Package 'spatstat.explore'

September 22, 2025

Version 3.5-3

Date 2025-09-22

Title Exploratory Data Analysis for the 'spatstat' Family

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

Depends R (>= 3.5.0), spatstat.data (>= 3.1-8), spatstat.univar (>= 3.1-4), spatstat.geom (>= 3.6), spatstat.random (>= 3.4), stats, graphics, grDevices, utils, methods, nlme

Imports spatstat.utils (>= 3.2), spatstat.sparse (>= 3.1), goftest (>= 1.2-2), Matrix, abind

Suggests sm, gsl, locfit, spatial, fftwtools (>= 0.9-8), spatstat.linnet (>= 3.3), spatstat.model (>= 3.4), spatstat (>= 3.4)

Description Functionality for exploratory data analysis and nonparametric analysis of spatial data, mainly spatial point patterns, in the 'spatstat' family of packages. (Excludes analysis of spatial data on a linear network, which is covered by the separate package 'spatstat.linnet'.) Methods include quadrat counts, K-functions and their simulation envelopes, nearest neighbour distance and empty space statistics, Fry plots, pair correlation function, kernel smoothed intensity, relative risk estimation with cross-validated bandwidth selection, mark correlation functions, segregation indices, mark dependence diagnostics, and kernel estimates of covariate effects. Formal hypothesis tests of random pattern (chi-squared, Kolmogorov-Smirnov, Monte Carlo, Diggle-Cressie-Loosmore-Ford, Dao-Genton, two-stage Monte Carlo) and tests for covariate effects (Cox-Berman-Waller-Lawson, Kolmogorov-Smirnov, ANOVA) are also supported.

URL http://spatstat.org/

NeedsCompilation yes

ByteCompile true

BugReports https://github.com/spatstat/spatstat.explore/issues

Author Adrian Baddeley [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0001-9499-8382>),
Rolf Turner [aut, cph] (ORCID: <https://orcid.org/0000-0001-5521-5218>),
Ege Rubak [aut, cph] (ORCID: <https://orcid.org/0000-0002-6675-533X>),
Kasper Klitgaard Berthelsen [ctb],
Warick Brown [cph],
Achmad Choiruddin [ctb],
Ya-Mei Chang [ctb],
Jean-Francois Coeurjolly [ctb],
Ottmar Cronie [ctb],
Tilman Davies [ctb, cph],
Julian Gilbey [ctb],
Jonatan Gonzalez [ctb],
Yongtao Guan [ctb],
Ute Hahn [ctb],
Martin Hazelton [ctb],
Kassel Hingee [ctb, cph],
Abdollah Jalilian [ctb],
Frederic Lavancier [ctb],
Marie-Colette van Lieshout [ctb, cph],
Greg McSwiggan [ctb],
Robin K Milne [cph],
Tuomas Rajala [ctb],
Suman Rakshit [ctb, cph],
Dominic Schuhmacher [ctb],
Rasmus Plenge Waagepetersen [ctb],
Hangsheng Wang [ctb],
Tingting Zhan [ctb]

Repository CRAN

Date/Publication 2025-09-22 05:10:20 UTC

Contents

`spatstat.explore-package`

The spatstat.explore Package

Description

The **spatstat.explore** package belongs to the **spatstat** family of packages. It contains the core functionality for statistical analysis and modelling of spatial data.

Details

spatstat is a family of R packages for the statistical analysis of spatial data. Its main focus is the analysis of spatial patterns of points in two-dimensional space.

The original **spatstat** package has now been split into several sub-packages.

This sub-package **spatstat.explore** contains the user-level functions that perform exploratory data analysis and nonparametric data analysis of spatial data.

(The main exception is that functions for linear networks are in the separate sub-package **spatstat.linnet**.)

Structure of the spatstat family

The orginal **spatstat** package grew to be very large. It has now been divided into several **sub-packages**:

- **spatstat.utils** containing basic utilities
- **spatstat.sparse** containing linear algebra utilities
- **spatstat.data** containing datasets
- **spatstat.univar** containing functions for estimating probability distributions of random variables
- **spatstat.geom** containing geometrical objects and geometrical operations
- **spatstat.explore** containing the functionality for exploratory data analysis and nonparametric analysis of spatial data.
- **spatstat.model** containing the functionality for statistical modelling, model-fitting, formal statistical inference and informal model diagnostics.
- **spatstat.linnet** containing functions for spatial data on a linear network
- **spatstat**, which simply loads the other sub-packages listed above, and provides documentation.

When you install **spatstat**, these sub-packages are also installed. Then if you load the **spatstat** package by typing `library(spatstat)`, the other sub-packages listed above will automatically be loaded or imported.

For an overview of all the functions available in the sub-packages of **spatstat**, see the help file for `"spatstat-package"` in the **spatstat** package.

Additionally there are several **extension packages:**

- **spatstat.gui** for interactive graphics
- **spatstat.local** for local likelihood (including geographically weighted regression)
- **spatstat.Knet** for additional, computationally efficient code for linear networks
- **spatstat.sphere** (under development) for spatial data on a sphere, including spatial data on the earth's surface

The extension packages must be installed separately and loaded explicitly if needed. They also have separate documentation.

Overview of Functionality in spatstat.explore

The **spatstat** family of packages 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.

For an overview, see the help file for `"spatstat-package"` in the **spatstat** package.

Following is a list of the functionality provided in the **spatstat.explore** package only.

To simulate a random point pattern:

Functions for generating random point patterns are now contained in the **spatstat.random** package.

To interrogate a point pattern:

`density.ppp`	kernel estimation of point pattern intensity
`densityHeat.ppp`	diffusion kernel estimation of point pattern intensity
`Smooth.ppp`	kernel smoothing of marks of point pattern
`sharpen.ppp`	data sharpening

Manipulation of pixel images:

An object of class `"im"` represents a pixel image.

`blur`	apply Gaussian blur to image
`Smooth.im`	apply Gaussian blur to image
`transect.im`	line transect of image
`pixelcentres`	extract centres of pixels
`rnoise`	random pixel noise

Line segment patterns

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

`density.psp`	kernel smoothing of line segments
`rpoisline`	generate a realisation of the Poisson line process inside a window

Tessellations

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

rpoislinetess	generate tessellation using Poisson line process

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".

runifpoint3	generate uniform random points in 3-D
rpoispp3	generate Poisson random points in 3-D
envelope.pp3	generate simulation envelopes for 3-D pattern

Multi-dimensional space-time point patterns

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

runifpointx	generate uniform random points
rpoisppx	generate Poisson random points

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.CvL	Cronie-Van Lieshout bandwidth selection for density estimation
bw.scott	Scott's rule of thumb for density estimation
bw.abram.ppp	Abramson's rule for adaptive bandwidths
bw.relrisk	cross-validated bandwidth selection for relrisk
bw.smoothppp	cross-validated bandwidth selection for Smooth.ppp
bw.frac	bandwidth selection using window geometry

Modern exploratory tools:

clusterset	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

sdr	Sufficient Data Reduction
thresholdSelect	optimal thresholding of a predictor

Summary statistics for a point pattern:

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

Selecting the bandwidth for kernel estimation of the summary function:

bw.stoyan	Stoyan's rule of thumb for bandwidth for pcf
bw.pcf	cross-validated bandwidth selection for pcf
bw.pcfinhom	cross-validated bandwidth selection for pcfinhom
bw.bdh	Adjusted Stoyan rule of thumb for bandwidth for pcfinhom

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
density.ppp	kernel smoothed density
densityHeat.ppp	diffusion 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.

relrisk	kernel estimation of relative risk
scan.test	spatial scan test of elevated risk
Gcross,Gdot,Gmulti	multitype nearest neighbour distributions $G_{ij}, G_{i\bullet}$
Kcross,Kdot, Kmulti	multitype K-functions $K_{ij}, K_{i\bullet}$
Lcross,Ldot	multitype L-functions $L_{ij}, L_{i\bullet}$
Jcross,Jdot,Jmulti	multitype J-functions $J_{ij}, J_{i\bullet}$
pcfcross	multitype pair correlation function g_{ij}
pcfdot	multitype pair correlation function $g_{i\bullet}$
pcfmulti	general pair correlation function
markconnect	marked connection function p_{ij}
alltypes	estimates of the above for all i, j pairs
Iest	multitype I-function
Kcross.inhom,Kdot.inhom	inhomogeneous counterparts of Kcross, Kdot
Lcross.inhom,Ldot.inhom	inhomogeneous counterparts of Lcross, Ldot
pcfcross.inhom,pcfdot.inhom	inhomogeneous counterparts of pcfcross, pcfdot
localKcross,localKdot	local counterparts of Kcross, Kdot
localLcross,localLdot	local counterparts of Lcross, Ldot
localKcross.inhom,localLcross.inhom	local counterparts of Kcross.inhom, Lcross.inhom

Summary statistics for a marked point pattern: A marked point pattern is represented by an object X of class "ppp" with a component X$marks. The entries in the vector X$marks may be numeric, complex, string or any other atomic type. For numeric marks, there are the following functions:

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

For marks of any type, there are the following:

Gmulti	multitype nearest neighbour distribution
Kmulti	multitype K-function
Jmulti	multitype J-function

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

Programming tools:

marktable	tabulate the marks of neighbours in a point pattern

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

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

Model fitting

Functions for fitting point process models are now contained in the **spatstat.model** package.

Simulation

There are many ways to generate a random point pattern, line segment pattern, pixel image or tessellation in **spatstat**.

Random point patterns: Functions for random generation are now contained in the **spatstat.random** package.

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.

Methods for simulating a fitted model are now contained in the **spatstat.model** package.

Other random patterns: Functions for random generation are now contained in the **spatstat.random** package.

Simulation-based inference

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

Manipulation of envelope objects:

as.data.frame.envelope	convert to data frame
with.fv	calculations with column(s) of data
eval.fv	calculations with all columns of data
plot.envelope	plot envelope
summary.envelope	print summary information
pool.envelope	pool data from several envelopes
ptwise.envelope	compute pointwise statistics
bias.envelope	pointwise bias
RMSE.envelope	pointwise root mean square error
MISE.envelope	mean integrated squared error
ISB.envelope	integrated squared bias
IV.envelope	integrated variance
ISE.envelope	integrated squared error

Hypothesis tests:

quadrat.test	χ^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

Model diagnostics:

Classical measures of model sensitivity such as leverage and influence, and classical model diagnostic tools such as residuals, partial residuals, and effect estimates, have been adapted to point process models. These capabilities are now provided in the **spatstat.model** package.

Resampling and randomisation procedures

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

`quadratresample`	block resampling
`rshift`	random shifting of (subsets of) points
`rthin`	random thinning

Licence

Acknowledgements

Kasper Klitgaard Berthelsen, Ottmar Cronie, Tilman Davies, Julian Gilbey, Yongtao Guan, Ute Hahn, Kassel Hingee, Abdollah Jalilian, Marie-Colette van Lieshout, Greg McSwiggan, Tuomas Rajala, Suman Rakshit, Dominic Schuhmacher, Rasmus Waagepetersen and Hangsheng Wang made substantial contributions of code.

For comments, corrections, bug alerts and suggestions, we thank Monsuru Adepeju, Corey Anderson, Ang Qi Wei, Ryan Arellano, Jens Åström, Robert Aue, Marcel Austenfeld, Sandro Azaele, Malissa Baddeley, Guy Bayegnak, Colin Beale, Melanie Bell, Thomas Bendtsen, Ricardo Bernhardt, Andrew Bevan, Brad Biggerstaff, Anders Bilgrau, Leanne Bischof, Christophe Biscio, Roger Bivand, Jose M. Blanco Moreno, Florent Bonneu, Jordan Brown, Ian Buller, Julian Burgos, Simon Byers, Ya-Mei Chang, Jianbao Chen, Igor Chernayavsky, Y.C. Chin, Bjarke Christensen, Lucía Cobo Sanchez, Jean-Francois Coeurjolly, Kim Colyvas, Hadrien Commenges, Rochelle Constantine, Robin Corria Ainslie, Richard Cotton, Marcelino de la Cruz, Peter Dalgaard, Mario D'Antuono, Sourav Das, Peter Diggle, Patrick Donnelly, Ian Dryden, Stephen Eglen, Ahmed El-Gabbas, Belarmain Fandohan, Olivier Flores, David Ford, Peter Forbes, Shane Frank, Janet Franklin, Funwi-Gabga Neba, Oscar Garcia, Agnes Gault, Jonas Geldmann, Marc Genton, Shaaban Ghalandarayeshi, Jason Goldstick, Pavel Grabarnik, C. Graf, Ute Hahn, Andrew Hardegen, Martin Bøgsted Hansen, Martin Hazelton, Juha Heikkinen, Mandy Hering, Markus Herrmann, Maximilian Hesselbarth, Paul Hewson, Hamidreza Heydarian, Kurt Hornik, Philipp Hunziker, Jack Hywood, Ross Ihaka, Čenk Içös, Aruna Jammalamadaka, Robert John-Chandran, Devin Johnson, Mahdieh Khanmohammadi, Bob Klaver, Lily Kozmian-Ledward, Peter Kovesi, Mike Kuhn, Jeff Laake, Robert Lamb, Frédéric Lavancier, Tom Lawrence, Tomas Lazauskas, Jonathan Lee, George Leser, Angela Li, Li Haitao, George Limitsios, Andrew Lister, Nestor Luambua, Ben Madin, Martin Maechler, Kiran Marchikanti, Jeff Marcus, Robert Mark, Peter McCullagh, Monia Mahling, Jorge Mateu Mahiques, Ulf Mehlig, Frederico Mestre, Sebastian Wastl Meyer, Mi Xiangcheng, Lore De Middeleer, Robin Milne, Enrique Miranda, Jesper Møller, Annie Mollié, Ines Moncada, Mehdi Moradi, Virginia Morera Pujol, Erika Mudrak, Gopalan Nair, Nader Najari, Nicoletta Nava, Linda Stougaard Nielsen, Felipe Nunes, Jens Randel Nyengaard, Jens Oehlschlägel, Thierry Onkelinx, Sean O'Riordan, Evgeni Parilov, Jeff Picka, Nicolas Picard, Tim Pollington, Mike Porter, Sergiy Protsiv, Adrian Raftery, Ben Ramage, Pablo Ramon, Xavier Raynaud, Nicholas

Read, Matt Reiter, Ian Renner, Tom Richardson, Brian Ripley, Ted Rosenbaum, Barry Rowlingson, Jason Rudokas, Tyler Rudolph, John Rudge, Christopher Ryan, Farzaneh Safavimanesh, Aila Särkkä, Cody Schank, Katja Schladitz, Sebastian Schutte, Bryan Scott, Olivia Semboli, François Sémécurbe, Vadim Shcherbakov, Shen Guochun, Shi Peijian, Harold-Jeffrey Ship, Tammy L Silva, Ida-Maria Sintorn, Yong Song, Malte Spiess, Mark Stevenson, Kaspar Stucki, Jan Sulavik, Michael Sumner, P. Surovy, Ben Taylor, Thordis Linda Thorarinsdottir, Leigh Torres, Berwin Turlach, Torben Tvedebrink, Kevin Ummer, Medha Uppala, Andrew van Burgel, Tobias Verbeke, Mikko Vihtakari, Alexendre Villers, Fabrice Vinatier, Maximilian Vogtland, Sasha Voss, Sven Wagner, Hao Wang, H. Wendrock, Jan Wild, Carl G. Witthoft, Selene Wong, Maxime Woringer, Luke Yates, Mike Zamboni and Achim Zeileis.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

`adaptive.density` *Adaptive Estimate of Intensity of Point Pattern*

Description

Computes an adaptive estimate of the intensity function of a point pattern.

Usage

```
adaptive.density(X, ..., method=c("voronoi","kernel", "nearest"))
```

Arguments

`X`	Point pattern (object of class `"ppp"` or `"lpp"`).
`method`	Character string specifying the estimation method
`...`	Additional arguments passed to `densityVoronoi`, `densityAdaptiveKernel.ppp` or `nndensity.ppp`.

Details

This function is an alternative to `density.ppp` and `density.lpp`. It computes an estimate of the intensity function of a point pattern dataset. The result is a pixel image giving the estimated intensity.

If `method="voronoi"` the data are passed to the function `densityVoronoi` which estimates the intensity using the Voronoi-Dirichlet tessellation.

If `method="kernel"` the data are passed to the function `densityAdaptiveKernel.ppp` which estimates the intensity using a variable-bandwidth kernel estimator. (This is not yet supported when `X` has class `"lpp"`.)

If `method="nearest"` the data are passed to the function `nndensity.ppp` which estimates the intensity using the distance to the k-th nearest data point. (This is not yet supported when `X` has class `"lpp"`.)

Value

A pixel image (object of class "im" or "linim") whose values are estimates of the intensity of X.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk> and Mehdi Moradi <m2.moradi@yahoo.com>.

See Also

density.ppp, densityVoronoi, densityAdaptiveKernel.ppp, nndensity.ppp, im.object.

Examples

```
plot(adaptive.density(nztrees, 1), main="Voronoi estimate")
```

---

allstats *Calculate four standard summary functions of a point pattern.*

Description

Calculates the F, G, J, and K summary functions for an unmarked point pattern. Returns them as a function array (of class "fasp", see fasp.object).

Usage

```
allstats(pp, ..., dataname=NULL, verb=FALSE)
```

Arguments

pp	The observed point pattern, for which summary function estimates are required. An object of class "ppp". It must not be marked.
...	Optional arguments passed to the summary functions Fest, Gest, Jest and Kest.
dataname	A character string giving an optional (alternative) name for the point pattern.
verb	A logical value meaning "verbose". If TRUE, progress reports are printed during calculation.

Details

This computes four standard summary statistics for a point pattern: the empty space function $F(r)$, nearest neighbour distance distribution function $G(r)$, van Lieshout-Baddeley function $J(r)$ and Ripley's function $K(r)$. The real work is done by Fest, Gest, Jest and Kest respectively. Consult the help files for these functions for further information about the statistical interpretation of F, G, J and K.

If verb is TRUE, then "progress reports" (just indications of completion) are printed out when the calculations are finished for each of the four function types.

The overall title of the array of four functions (for plotting by plot.fasp) will be formed from the argument dataname. If this is not given, it defaults to the expression for pp given in the call to allstats.

Value

A list of length 4 containing the F, G, J and K functions respectively.

The list can be plotted directly using plot (which dispatches to plot.anylist).

Each list entry retains the format of the output of the relevant estimating routine Fest, Gest, Jest or Kest. Thus each entry in the list is a function value table (object of class "fv", see fv.object).

The default formulae for plotting these functions are cbind(km,theo) ~ r for F, G, and J, and cbind(trans,theo) ~ r for K.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

plot.anylist, plot.fv, fv.object, Fest, Gest, Jest, Kest

Examples

```
a <- allstats(swedishpines,dataname="Swedish Pines")
if(interactive()) {
plot(a)
plot(a, subset=list("r<=15","r<=15","r<=15","r<=50"))
}
```

---

alltypes *Calculate Summary Statistic for All Types in a Multitype Point Pattern*

Description

Given a marked point pattern, this computes the estimates of a selected summary function (F,G, J, K etc) of the pattern, for all possible combinations of marks, and returns these functions in an array.

Usage

```
alltypes(X, fun="K", ...,
         dataname=NULL,verb=FALSE,envelope=FALSE,reuse=TRUE)
```

Arguments

X	The observed point pattern, for which summary function estimates are required. An object of class "ppp" or "lpp".
fun	The summary function. Either an R function, or a character string indicating the summary function required. Options for strings are "F", "G", "J", "K", "L", "pcf", "Gcross", "Jcross", "Kcross", "Lcross", "Gdot", "Jdot", "Kdot", "Ldot".
...	Arguments passed to the summary function (and to the function envelope if appropriate)
dataname	Character string giving an optional (alternative) name to the point pattern, different from what is given in the call. This name, if supplied, may be used by plot.fasp() in forming the title of the plot. If not supplied it defaults to the parsing of the argument supplied as X in the call.
verb	Logical value. If verb is true then terse "progress reports" (just the values of the mark indices) are printed out when the calculations for that combination of marks are completed.
envelope	Logical value. If envelope is true, then simulation envelopes of the summary function will also be computed. See Details.
reuse	Logical value indicating whether the envelopes in each panel should be based on the same set of simulated patterns (reuse=TRUE) or on different, independent sets of simulated patterns (reuse=FALSE).

Details

This routine is a convenient way to analyse the dependence between types in a multitype point pattern. It computes the estimates of a selected summary function of the pattern, for all possible combinations of marks. It returns these functions in an array (an object of class "fasp") amenable to plotting by plot.fasp().

The argument fun specifies the summary function that will be evaluated for each type of point, or for each pair of types. It may be either an R function or a character string.

Suppose that the points have possible types $1, 2, \ldots, m$ and let X_i denote the pattern of points of type i only.

If fun="F" then this routine calculates, for each possible type i, an estimate of the Empty Space Function $F_i(r)$ of X_i. See Fest for explanation of the empty space function. The estimate is computed by applying Fest to X_i with the optional arguments ....

If fun is "Gcross", "Jcross", "Kcross" or "Lcross", the routine calculates, for each pair of types (i, j), an estimate of the "i-toj" cross-type function $G_{ij}(r)$, $J_{ij}(r)$, $K_{ij}(r)$ or $L_{ij}(r)$ respectively describing the dependence between X_i and X_j. See Gcross, Jcross, Kcross or Lcross respectively for explanation of these functions. The estimate is computed by applying the relevant function (Gcross etc) to X using each possible value of the arguments i,j, together with the optional arguments ....

If fun is "pcf" the routine calculates the cross-type pair correlation function pcfcross between each pair of types.

If fun is "Gdot", "Jdot", "Kdot" or "Ldot", the routine calculates, for each type i, an estimate of the "i-to-any" dot-type function $G_{i\bullet}(r)$, $J_{i\bullet}(r)$ or $K_{i\bullet}(r)$ or $L_{i\bullet}(r)$ respectively describing the

dependence between X_i and X. See Gdot, Jdot, Kdot or Ldot respectively for explanation of these functions. The estimate is computed by applying the relevant function (Gdot etc) to X using each possible value of the argument i, together with the optional arguments ....

The letters "G", "J", "K" and "L" are interpreted as abbreviations for Gcross, Jcross, Kcross and Lcross respectively, assuming the point pattern is marked. If the point pattern is unmarked, the appropriate function Fest, Jest, Kest or Lest is invoked instead.

If envelope=TRUE, then as well as computing the value of the summary function for each combination of types, the algorithm also computes simulation envelopes of the summary function for each combination of types. The arguments ... are passed to the function envelope to control the number of simulations, the random process generating the simulations, the construction of envelopes, and so on.

When envelope=TRUE it is possible that errors could occur because the simulated point patterns do not satisfy the requirements of the summary function (for example, because the simulated pattern is empty and fun requires at least one point). If the number of such errors exceeds the maximum permitted number maxnerr, then the envelope algorithm will give up, and will return the empirical summary function for the data point pattern, fun(X), in place of the envelope.

Value

A function array (an object of class "fasp", see fasp.object). This can be plotted using plot.fasp.

If the pattern is not marked, the resulting "array" has dimensions 1×1. Otherwise the following is true:

If fun="F", the function array has dimensions $m \times 1$ where m is the number of different marks in the point pattern. The entry at position [i,1] in this array is the result of applying Fest to the points of type i only.

If fun is "Gdot", "Jdot", "Kdot" or "Ldot", the function array again has dimensions $m \times 1$. The entry at position [i,1] in this array is the result of Gdot(X, i), Jdot(X, i) Kdot(X, i) or Ldot(X, i) respectively.

If fun is "Gcross", "Jcross", "Kcross" or "Lcross" (or their abbreviations "G", "J", "K" or "L"), the function array has dimensions $m \times m$. The [i,j] entry of the function array (for $i \neq j$) is the result of applying the function Gcross, Jcross, Kcross orLcross to the pair of types (i,j). The diagonal [i,i] entry of the function array is the result of applying the univariate function Gest, Jest, Kest or Lest to the points of type i only.

If envelope=FALSE, then each function entry fns[[i]] retains the format of the output of the relevant estimating routine Fest, Gest, Jest, Kest, Lest, Gcross, Jcross ,Kcross, Lcross, Gdot, Jdot, Kdot or Ldot The default formulae for plotting these functions are cbind(km,theo) ~ r for F, G, and J functions, and cbind(trans,theo) ~ r for K and L functions.

If envelope=TRUE, then each function entry fns[[i]] has the same format as the output of the envelope command.

Note

Sizeable amounts of memory may be needed during the calculation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>.

See Also

plot.fasp, fasp.object, Fest, Gest, Jest, Kest, Lest, Gcross, Jcross, Kcross, Lcross, Gdot, Jdot, Kdot, envelope.

Examples

```
# bramblecanes (3 marks).
bram <- bramblecanes

bF <- alltypes(bram,"F",verb=TRUE)
plot(bF)
if(interactive()) {
  plot(alltypes(bram,"G"))
  plot(alltypes(bram,"Gdot"))
}

# Swedishpines (unmarked).
swed <- swedishpines

plot(alltypes(swed,"K"))

plot(alltypes(amacrine, "pcf"), ylim=c(0,1.3))

# envelopes
bKE <- alltypes(bram,"K",envelope=TRUE,nsim=19)
# global version:

  bFE <- alltypes(bram,"F",envelope=TRUE,nsim=19,global=TRUE)


# extract one entry
as.fv(bKE[1,1])
```

---

as.data.frame.envelope

Coerce Envelope to Data Frame

Description

Converts an envelope object to a data frame.

Usage

```
## S3 method for class 'envelope'
as.data.frame(x, ..., simfuns=FALSE)
```

Arguments

x	Envelope object (class "envelope").
...	Ignored.
simfuns	Logical value indicating whether the result should include the values of the simulated functions that were used to build the envelope.

Details

This is a method for the generic function as.data.frame for the class of envelopes (see envelope.

The result is a data frame with columns containing the values of the function argument (usually named r), the function estimate for the original point pattern data (obs), the upper and lower envelope limits (hi and lo), and possibly additional columns.

If simfuns=TRUE, the result also includes columns of values of the simulated functions that were used to compute the envelope. This is possible only when the envelope was computed with the argument savefuns=TRUE in the call to envelope.

Value

A data frame.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

Examples

```
E <- envelope(cells, nsim=5, savefuns=TRUE)
tail(as.data.frame(E))
tail(as.data.frame(E, simfuns=TRUE))
```

---

as.function.fv *Convert Function Value Table to Function*

Description

Converts an object of class "fv" to an R language function.

Usage

```
  ## S3 method for class 'fv'
as.function(x, ..., value=".y", extrapolate=FALSE)
```

Arguments

x	Object of class "fv" or "rhohat".
...	Ignored.
value	Optional. Character string or character vector selecting one or more of the columns of x for use as the function value. See Details.
extrapolate	Logical, indicating whether to extrapolate the function outside the domain of x. See Details.

Details

A function value table (object of class "fv") is a convenient way of storing and plotting several different estimates of the same function. Objects of this class are returned by many commands in **spatstat**, such as Kest, which returns an estimate of Ripley's K-function for a point pattern dataset.

Sometimes it is useful to convert the function value table to a function in the R language. This is done by as.function.fv. It converts an object x of class "fv" to an R function f.

If f <- as.function(x) then f is an R function that accepts a numeric argument and returns a corresponding value for the summary function by linear interpolation between the values in the table x.

Argument values lying outside the range of the table yield an NA value (if extrapolate=FALSE) or the function value at the nearest endpoint of the range (if extrapolate = TRUE). To apply different rules to the left and right extremes, use extrapolate=c(TRUE,FALSE) and so on.

Typically the table x contains several columns of function values corresponding to different edge corrections. Auxiliary information for the table identifies one of these columns as the *recommended value*. By default, the values of the function f <- as.function(x) are taken from this column of recommended values. This default can be changed using the argument value, which can be a character string or character vector of names of columns of x. Alternatively value can be one of the abbreviations used by fvnames.

If value specifies a single column of the table, then the result is a function f(r) with a single numeric argument r (with the same name as the orginal argument of the function table).

If value specifies several columns of the table, then the result is a function f(r,what) where r is the numeric argument and what is a character string identifying the column of values to be used.

The formal arguments of the resulting function are f(r, what=value), which means that in a call to this function f, the permissible values of what are the entries of the original vector value; the default value of what is the first entry of value.

The command as.function.fv is a method for the generic command as.function.

Value

A function(r) or function(r,what) where r is the name of the original argument of the function table.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

as.function.rhohat, fv, fv.object, fvnames, plot.fv, Kest

Examples

```
K <- Kest(cells)
f <- as.function(K)
f
f(0.1)
g <- as.function(K, value=c("iso", "trans"))
g
g(0.1, "trans")
```

---

as.function.rhohat *Convert Function Table to Function*

Description

Converts an object of class "rhohat" to an R language function.

Usage

```
## S3 method for class 'rhohat'
as.function(x, ..., value=".y", extrapolate=TRUE)
```

Arguments

x	Object of class "rhohat", produced by the function rhohat.
...	Ignored.
value	Optional. Character string or character vector selecting one or more of the columns of x for use as the function value. See Details.
extrapolate	Logical, indicating whether to extrapolate the function outside the domain of x. See Details.

Details

An object of class "rhohat" is essentially a data frame of estimated values of the function $rho(x)$ as described in the help file for rhohat.

Sometimes it is useful to convert the function value table to a function in the R language. This is done by as.function.rhohat. It converts an object x of class "rhohat" to an R function f.

The command as.function.rhohat is a method for the generic command as.function for the class "rhohat".

If f <- as.function(x) then f is an R function that accepts a numeric argument and returns a corresponding value for the summary function by linear interpolation between the values in the table x.

Argument values lying outside the range of the table yield an NA value (if extrapolate=FALSE) or the function value at the nearest endpoint of the range (if extrapolate = TRUE). To apply different rules to the left and right extremes, use extrapolate=c(TRUE,FALSE) and so on.

Typically the table x contains several columns of function values corresponding to different edge corrections. Auxiliary information for the table identifies one of these columns as the *recommended value*. By default, the values of the function f <- as.function(x) are taken from this column of recommended values. This default can be changed using the argument value, which can be a character string or character vector of names of columns of x. Alternatively value can be one of the abbreviations used by fvnames.

If value specifies a single column of the table, then the result is a function f(r) with a single numeric argument r (with the same name as the orginal argument of the function table).

If value specifies several columns of the table, then the result is a function f(r,what) where r is the numeric argument and what is a character string identifying the column of values to be used.

The formal arguments of the resulting function are f(r, what=value), which means that in a call to this function f, the permissible values of what are the entries of the original vector value; the default value of what is the first entry of value.

Value

A function(r) or function(r,what) where r is the name of the original argument of the function table.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

rhohat, methods.rhohat, as.function.fv.

Examples

```
g <- rhohat(cells, "x")
f <- as.function(g)
f
f(0.1)
```

---

as.fv *Convert Data To Class fv*

Description

Converts data into a function table (an object of class "fv").

Usage

```
  as.fv(x)

  ## S3 method for class 'fv'
as.fv(x)

  ## S3 method for class 'data.frame'
as.fv(x)

  ## S3 method for class 'matrix'
as.fv(x)

  ## S3 method for class 'fasp'
as.fv(x)

  ## S3 method for class 'bw.optim'
as.fv(x)
```

Arguments

x	Data which will be converted into a function table

Details

This command converts data x, that could be interpreted as the values of a function, into a function value table (object of the class "fv" as described in fv.object). This object can then be plotted easily using plot.fv.

The dataset x may be any of the following:

- an object of class "fv";
- a matrix or data frame with at least two columns;
- an object of class "fasp", representing an array of "fv" objects.
- an object of class "minconfit", giving the results of a minimum contrast fit by the command mincontrast. The
- an object of class "kppm", representing a fitted Cox or cluster point process model, obtained from the model-fitting command kppm;
- an object of class "dppm", representing a fitted determinantal point process model, obtained from the model-fitting command dppm;
- an object of class "bw.optim", representing an optimal choice of smoothing bandwidth by a cross-validation method, obtained from commands like bw.diggle.

The function as.fv is generic, with methods for each of the classes listed above. The behaviour is as follows:

- If x is an object of class "fv", it is returned unchanged.
- If x is a matrix or data frame, the first column is interpreted as the function argument, and subsequent columns are interpreted as values of the function computed by different methods.

- If x is an object of class "fasp" representing an array of "fv" objects, these are combined into a single "fv" object.
- If x is an object of class "minconfit", or an object of class "kppm" or "dppm", the result is a function table containing the observed summary function and the best fit summary function.
- If x is an object of class "bw.optim", the result is a function table of the optimisation criterion as a function of the smoothing bandwidth.

Value

An object of class "fv" (see fv.object).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>

Examples

```
r <- seq(0, 1, length=101)
x <- data.frame(r=r, y=r^2)
as.fv(x)
```

---

as.owin.quadrattest *Convert Data To Class owin*

Description

Converts data specifying an observation window in any of several formats, into an object of class "owin".

Usage

```
 ## S3 method for class 'quadrattest'
as.owin(W, ..., fatal=TRUE)
```

Arguments

W	Data specifying an observation window, in any of several formats described under *Details* below.
fatal	Logical value determining what to do if the data cannot be converted to an observation window. See Details.
...	Ignored.

Details

The class "owin" is a way of specifying the observation window for a point pattern. See owin.object for an overview.

The generic function as.owin converts data in any of several formats into an object of class "owin" for use by the **spatstat** package. The function as.owin is generic, with methods for different classes of objects, and a default method.

The argument W may be

- an object of class "owin"
- a structure with entries xrange, yrange specifying the x and y dimensions of a rectangle
- a structure with entries named xmin, xmax, ymin, ymax (in any order) specifying the x and y dimensions of a rectangle. This will accept objects of class bbox in the sf package.
- a numeric vector of length 4 (interpreted as (xmin, xmax, ymin, ymax) in that order) specifying the x and y dimensions of a rectangle
- a structure with entries named xl, xu, yl, yu (in any order) specifying the x and y dimensions of a rectangle as (xmin, xmax) = (xl, xu) and (ymin, ymax) = (yl, yu). This will accept objects of class spp used in the Venables and Ripley **spatial** package.
- an object of class "ppp" representing a point pattern. In this case, the object's window structure will be extracted.
- an object of class "psp" representing a line segment pattern. In this case, the object's window structure will be extracted.
- an object of class "tess" representing a tessellation. In this case, the object's window structure will be extracted.
- an object of class "quad" representing a quadrature scheme. In this case, the window of the data component will be extracted.
- an object of class "im" representing a pixel image. In this case, a window of type "mask" will be returned, with the same pixel raster coordinates as the image. An image pixel value of NA, signifying that the pixel lies outside the window, is transformed into the logical value FALSE, which is the corresponding convention for window masks.
- an object of class "ppm", "kppm", "slrm" or "dppm" representing a fitted point process model. In this case, if from="data" (the default), as.owin extracts the original point pattern data to which the model was fitted, and returns the observation window of this point pattern. If from="covariates" then as.owin extracts the covariate images to which the model was fitted, and returns a binary mask window that specifies the pixel locations.
- an object of class "lpp" representing a point pattern on a linear network. In this case, as.owin extracts the linear network and returns a window containing this network.
- an object of class "lppm" representing a fitted point process model on a linear network. In this case, as.owin extracts the linear network and returns a window containing this network.
- A data.frame with exactly three columns. Each row of the data frame corresponds to one pixel. Each row contains the x and y coordinates of a pixel, and a logical value indicating whether the pixel lies inside the window.
- A data.frame with exactly two columns. Each row of the data frame contains the x and y coordinates of a pixel that lies inside the window.

- an object of class "distfun", "nnfun" or "funxy" representing a function of spatial location, defined on a spatial domain. The spatial domain of the function will be extracted.
- an object of class "rmhmodel" representing a point process model that can be simulated using rmh. The window (spatial domain) of the model will be extracted. The window may be NULL in some circumstances (indicating that the simulation window has not yet been determined). This is not treated as an error, because the argument fatal defaults to FALSE for this method.
- an object of class "layered" representing a list of spatial objects. See layered. In this case, as.owin will be applied to each of the objects in the list, and the union of these windows will be returned.
- An object of another suitable class from another package. For full details, see vignette('shapefiles').

If the argument W is not in one of these formats and cannot be converted to a window, then an error will be generated (if fatal=TRUE) or a value of NULL will be returned (if fatal=FALSE).

When W is a data frame, the argument step can be used to specify the pixel grid spacing; otherwise, the spacing will be guessed from the data.

Value

An object of class "owin" (see owin.object) specifying an observation window.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

as.owin, as.owin.rmhmodel, as.owin.lpp.

owin.object, owin.

Additional methods for as.owin may be provided by other packages outside the **spatstat** family.

Examples

```
te <- quadrat.test(redwood, nx=3)
as.owin(te)
```

---

as.tess *Convert Data To Tessellation*

Description

Converts data specifying a tessellation, in any of several formats, into an object of class "tess".

Usage

```
 ## S3 method for class 'quadrattest'
as.tess(X)
```

Arguments

X	Data to be converted to a tessellation.

Details

A tessellation is a collection of disjoint spatial regions (called *tiles*) that fit together to form a larger spatial region. This command creates an object of class "tess" that represents a tessellation.

This function converts data in any of several formats into an object of class "tess" for use by the **spatstat** package. The argument X may be

- an object of class "tess". The object will be stripped of any extraneous attributes and returned.
- a pixel image (object of class "im") with pixel values that are logical or factor values. Each level of the factor will determine a tile of the tessellation.
- a window (object of class "owin"). The result will be a tessellation consisting of a single tile.
- a set of quadrat counts (object of class "quadratcount") returned by the command quadratcount. The quadrats used to generate the counts will be extracted and returned as a tessellation.
- a quadrat test (object of class "quadrattest") returned by the command quadrat.test. The quadrats used to perform the test will be extracted and returned as a tessellation.
- a list of windows (objects of class "owin") giving the tiles of the tessellation.

The function as.tess is generic, with methods for various classes, as listed above.

Value

An object of class "tess" specifying a tessellation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

tess

Examples

```
h <- quadrat.test(nztrees, nx=4, ny=3)
as.tess(h)
```

auc *Area Under ROC Curve for Point Pattern Data*

Description

Compute the AUC (area under the Receiver Operating Characteristic curve) for a point pattern or other data.

Usage

```
auc(X, ...)

## S3 method for class 'ppp'
auc(X, covariate, ..., high = TRUE,  subset=NULL)

## S3 method for class 'roc'
auc(X, ...)

## S3 method for class 'cdftest'
auc(X, ..., high=TRUE)

## S3 method for class 'bermantest'
auc(X, ..., high=TRUE)

## S3 method for class 'im'
auc(X, covariate, ..., high=TRUE)
```

Arguments

X	Point pattern (object of class "ppp" or "lpp") or fitted point process model (object of class "ppm" or "kppm" or "lppm") or fitted spatial logistic regression model (object of class "slrm") or an ROC curve (object of class "roc" computed by roc).
covariate	Spatial covariate. Either a function(x,y), a pixel image (object of class "im"), or one of the strings "x" or "y" indicating the Cartesian coordinates.
...	Arguments passed to roc, and arguments passed to as.mask controlling the pixel resolution for calculations,
high	Logical value indicating whether the threshold operation should favour high or low values of the covariate.
subset	Optional. A spatial window (object of class "owin") specifying a subset of the data, for which the AUC should be calculated.

Details

This command computes the AUC, the area under the Receiver Operating Characteristic curve. The ROC itself is computed by roc.

The function auc is generic. There are methods for point patterns, fitted point process models, and many other kinds of objects.

For a point pattern X and a covariate Z, the AUC is a numerical index that measures the ability of the covariate to separate the spatial domain into areas of high and low density of points. Let x_i be a randomly-chosen data point from X and U a randomly-selected location in the study region. The AUC is the probability that $Z(x_i) > Z(U)$ assuming high=TRUE. That is, AUC is the probability that a randomly-selected data point has a higher value of the covariate Z than does a randomly-selected spatial location. The AUC is a number between 0 and 1. A value of 0.5 indicates a complete lack of discriminatory power.

Methods for calculating AUC for a point process model or spatial logistic regression model are described in auc.ppm and auc.lpp.

Some other kinds of objects in **spatstat** contain sufficient data to compute the AUC. These include the objects returned by rhohat, cdf.test and berman.test. Methods are provided here to compute the AUC from these objects.

Value

Numeric. For auc.ppp, auc.cdftest, auc.bermantest and auc.im, the result is a single number giving the AUC value.

For auc.roc, the result is a numeric vector with one entry for each column of function values of X.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Ege Rubak <rubak@math.aau.dk> and Suman Rakshit <Suman.Rakshit@curtin.edu.au>.

References

Baddeley, A., Rubak, E., Rakshit, S. and Nair, G. (2025) ROC curves for spatial point patterns and presence-absence data. doi:10.48550/arXiv.2506.03414..

Lobo, J.M., Jiménez-Valverde, A. and Real, R. (2007) AUC: a misleading measure of the performance of predictive distribution models. *Global Ecology and Biogeography* **17**(2) 145–151.

Nam, B.-H. and D'Agostino, R. (2002) Discrimination index, the area under the ROC curve. Pages 267–279 in Huber-Carol, C., Balakrishnan, N., Nikulin, M.S. and Mesbah, M., *Goodness-of-fit tests and model validity*, Birkhäuser, Basel.

See Also

roc, auc.ppm, auc.lpp.

youden.

Examples

```
auc(swedishpines, "x")
```

berman.test *Berman's Tests for Point Process Model*

Description

Tests the goodness-of-fit of a Poisson point process model using methods of Berman (1986).

Usage

```
berman.test(...)

## S3 method for class 'ppp'
berman.test(X, covariate,
                          which = c("Z1", "Z2"),
        alternative = c("two.sided", "less", "greater"), ...)
```

Arguments

X	A point pattern (object of class "ppp" or "lpp").
covariate	The spatial covariate on which the test will be based. An image (object of class "im") or a function.
which	Character string specifying the choice of test.
alternative	Character string specifying the alternative hypothesis.
...	Additional arguments controlling the pixel resolution (arguments dimyx, eps and rule.eps passed to as.mask) or other undocumented features.

Details

These functions perform a goodness-of-fit test of a Poisson point process model fitted to point pattern data. The observed distribution of the values of a spatial covariate at the data points, and the predicted distribution of the same values under the model, are compared using either of two test statistics Z_1 and Z_2 proposed by Berman (1986). The Z_1 test is also known as the Lawson-Waller test.

The function berman.test is generic, with methods for point patterns ("ppp" or "lpp") and point process models ("ppm" or "lppm").

- If X is a point pattern dataset (object of class "ppp" or "lpp"), then berman.test(X, ...) performs a goodness-of-fit test of the uniform Poisson point process (Complete Spatial Randomness, CSR) for this dataset.
- If model is a fitted point process model (object of class "ppm" or "lppm") then berman.test(model, ...) performs a test of goodness-of-fit for this fitted model. In this case, model should be a Poisson point process.

The test is performed by comparing the observed distribution of the values of a spatial covariate at the data points, and the predicted distribution of the same covariate under the model. Thus, you must nominate a spatial covariate for this test.

The argument `covariate` should be either a `function(x,y)` or a pixel image (object of class `"im"` containing the values of a spatial function. If `covariate` is an image, it should have numeric values, and its domain should cover the observation window of the `model`. If `covariate` is a function, it should expect two arguments `x` and `y` which are vectors of coordinates, and it should return a numeric vector of the same length as `x` and `y`.

First the original data point pattern is extracted from `model`. The values of the `covariate` at these data points are collected.

Next the values of the `covariate` at all locations in the observation window are evaluated. The point process intensity of the fitted model is also evaluated at all locations in the window.

- If `which="Z1"`, the test statistic Z_1 is computed as follows. The sum S of the covariate values at all data points is evaluated. The predicted mean μ and variance σ^2 of S are computed from the values of the covariate at all locations in the window. Then we compute $Z_1 = (S - \mu)/\sigma$. Closely-related tests were proposed independently by Waller et al (1993) and Lawson (1993) so this test is often termed the Lawson-Waller test in epidemiological literature.
- If `which="Z2"`, the test statistic Z_2 is computed as follows. The values of the `covariate` at all locations in the observation window, weighted by the point process intensity, are compiled into a cumulative distribution function F. The probability integral transformation is then applied: the values of the `covariate` at the original data points are transformed by the predicted cumulative distribution function F into numbers between 0 and 1. If the model is correct, these numbers are i.i.d. uniform random numbers. The standardised sample mean of these numbers is the statistic Z_2.

In both cases the null distribution of the test statistic is the standard normal distribution, approximately.

The return value is an object of class `"htest"` containing the results of the hypothesis test. The print method for this class gives an informative summary of the test outcome.

Value

An object of class `"htest"` (hypothesis test) and also of class `"bermantest"`, containing the results of the test. The return value can be plotted (by `plot.bermantest`) or printed to give an informative summary of the test.

Warning

The meaning of a one-sided test must be carefully scrutinised: see the printed output.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

Berman, M. (1986) Testing for spatial association between a point process and another stochastic process. *Applied Statistics* **35**, 54–62.

Lawson, A.B. (1993) On the analysis of mortality events around a prespecified fixed point. *Journal of the Royal Statistical Society, Series A* **156** (3) 363–377.

Waller, L., Turnbull, B., Clark, L.C. and Nasca, P. (1992) Chronic Disease Surveillance and testing of clustering of disease and exposure: Application to leukaemia incidence and TCE-contaminated dumpsites in upstate New York. *Environmetrics* **3**, 281–300.

See Also

`cdf.test`, `quadrat.test`, `ppm`

Examples

```
# Berman's data
X <- copper$SouthPoints
L <- copper$SouthLines
D <- distmap(L, eps=1)
# test of CSR
berman.test(X, D)
berman.test(X, D, "Z2")
```

`bind.fv` *Combine Function Value Tables*

Description

Advanced Use Only. Combine objects of class `"fv"`, or glue extra columns of data onto an existing `"fv"` object.

Usage

```
## S3 method for class 'fv'
cbind(...)

bind.fv(x, y, labl = NULL, desc = NULL, preferred = NULL, clip=FALSE)
```

Arguments

`...`	Any number of arguments, which are objects of class `"fv"`, or other data. See Details.
`x`	An object of class `"fv"`.
`y`	Either an object of class `"fv"`, a data frame, or a function. See Details.
`labl`	Plot labels (see `fv`) for columns of `y`. A character vector.
`desc`	Descriptions (see `fv`) for columns of `y`. A character vector.

preferred	Character string specifying the column which is to be the new recommended value of the function.
clip	Logical value indicating whether each object must have exactly the same domain, that is, the same sequence of values of the function argument (clip=FALSE, the default) or whether objects with different domains are permissible and will be restricted to a common domain (clip=TRUE).

Details

This documentation is provided for experienced programmers who want to modify the internal behaviour of **spatstat**.

The function cbind.fv is a method for the generic R function cbind. It combines any number of objects of class "fv" into a single object of class "fv". The objects must be compatible, in the sense that they have identical values of the function argument.

The function bind.fv is a lower level utility which glues additional columns onto an existing object x of class "fv". It has three modes of use:

- If the additional dataset y is an object of class "fv", then x and y must be compatible as described above. Then the columns of y that contain function values will be appended to the object x.
- Alternatively if y is a data frame, then y must have the same number of rows as x. All columns of y will be appended to x.
- Alternatively if y is a function in the R language, then this function will be evaluated at the argument values stored in the object x, and these function values will be appended as a new column to x.

The arguments labl and desc provide plot labels and description strings (as described in fv) for the *new* columns. If y is an object of class "fv" then labl and desc are optional, and default to the relevant entries in the object y. If y is a data frame then labl and desc should be provided, but there is a default.

For additional flexibility, cbind.fv also accepts arguments which are data frames or functions.

Value

An object of class "fv".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

fv for creating objects of class "fv" from raw data.

collapse.fv for combining several "fv" objects with similar columns.

with.fv for evaluating expressions.

fvnames for extracting and assigning the column names of standard components of "fv" objects.

Undocumented functions for modifying an "fv" object include tweak.fv.entry and rebadge.fv.

Examples

```
K1 <- Kest(cells, correction="border")
K2 <- Kest(cells, correction="iso")

# remove column 'theo' to avoid duplication
K2 <- K2[, names(K2) != "theo"]

cbind(K1, K2)

bind.fv(K1, K2, preferred="iso")

# constrain border estimate to be monotonically increasing
bm <- cumsum(c(0, pmax(0, diff(K1$border))))
bind.fv(K1, data.frame(bmono=bm),
            "%s[bmo](r)",
            "monotone border-corrected estimate of %s",
            "bmono")

# add a column of values defined by a function
cbind(K1, upper=function(r) { pi * r^2 + 0.1 })
```

`bits.envelope` *Global Envelopes for Balanced Independent Two-Stage Test*

Description

Computes the global envelopes corresponding to the balanced independent two-stage Monte Carlo test of goodness-of-fit.

Usage

```
bits.envelope(X, ...,
            nsim = 19, nrank = 1,
            alternative=c("two.sided", "less", "greater"),
            leaveout=1, interpolate = FALSE,
            savefuns=FALSE, savepatterns=FALSE,
            verbose = TRUE)
```

Arguments

`X`	Either a point pattern dataset (object of class `"ppp"`, `"lpp"` or `"pp3"`) or a fitted point process model (object of class `"ppm"`, `"kppm"` or `"slrm"`).
`...`	Arguments passed to `mad.test` or `envelope` to control the conduct of the test. Useful arguments include `fun` to determine the summary function, `rinterval` to determine the range of r values used in the test, and `verbose=FALSE` to turn off the messages.

nsim	Number of simulated patterns to be generated in each stage. Number of simulations in each basic test. There will be nsim repetitions of the basic test, each involving nsim simulated realisations, together with one independent set of nsim realisations, so there will be a total of nsim * (nsim + 1) simulations.
nrank	Integer. Rank of the envelope value amongst the nsim simulated values. A rank of 1 means that the minimum and maximum simulated values will be used.
alternative	Character string determining whether the envelope corresponds to a two-sided test (alternative="two.sided", the default) or a one-sided test with a lower critical boundary (alternative="less") or a one-sided test with an upper critical boundary (alternative="greater").
leaveout	Optional integer 0, 1 or 2 indicating how to calculate the deviation between the observed summary function and the nominal reference value, when the reference value must be estimated by simulation. See Details.
interpolate	Logical value indicating whether to interpolate the distribution of the test statistic by kernel smoothing, as described in Dao and Genton (2014, Section 5).
savefuns	Logical flag indicating whether to save the simulated function values (from the first stage).
savepatterns	Logical flag indicating whether to save the simulated point patterns (from the first stage).
verbose	Logical value determining whether to print progress reports.

Details

Computes global simulation envelopes corresponding to the balanced independent two-stage Monte Carlo test of goodness-of-fit described by Baddeley et al (2017). The envelopes are described in Baddeley et al (2019).

If X is a point pattern, the null hypothesis is CSR.

If X is a fitted model, the null hypothesis is that model.

This command is similar to dg.envelope which corresponds to the Dao-Genton test of goodness-of-fit. It was shown in Baddeley et al (2017) that the Dao-Genton test is biased when the significance level is very small (small p-values are not reliable) and we recommend bits.envelope in this case.

Value

An object of class "fv".

Author(s)

Adrian Baddeley, Andrew Hardegen, Tom Lawrence, Robin Milne, Gopalan Nair and Suman Rakshit. Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

References

Dao, N.A. and Genton, M. (2014) A Monte Carlo adjusted goodness-of-fit test for parametric models describing spatial point patterns. *Journal of Graphical and Computational Statistics* **23**, 497–517.

Baddeley, A., Hardegen, A., Lawrence, T., Milne, R.K., Nair, G. and Rakshit, S. (2017) On two-stage Monte Carlo tests of composite hypotheses. *Computational Statistics and Data Analysis* **114**, 75–87.

Baddeley, A., Hardegen, A., Lawrence, L., Milne, R.K., Nair, G.M. and Rakshit, S. (2019) Pushing the envelope: extensions of graphical Monte Carlo tests. In preparation.

See Also

dg.envelope, bits.test, mad.test, envelope

Examples

```
ns <- if(interactive()) 19 else 4
E <- bits.envelope(swedishpines, Lest, nsim=ns)
E
plot(E)
Eo <- bits.envelope(swedishpines, Lest, alternative="less", nsim=ns)
Ei <- bits.envelope(swedishpines, Lest, interpolate=TRUE, nsim=ns)
```

---

bits.test *Balanced Independent Two-Stage Monte Carlo Test*

Description

Performs a Balanced Independent Two-Stage Monte Carlo test of goodness-of-fit for spatial pattern.

Usage

```
bits.test(X, ...,
        exponent = 2, nsim=19,
        alternative=c("two.sided", "less", "greater"),
        leaveout=1, interpolate = FALSE,
        savefuns=FALSE, savepatterns=FALSE,
        verbose = TRUE)
```

Arguments

X	Either a point pattern dataset (object of class "ppp", "lpp" or "pp3") or a fitted point process model (object of class "ppm", "kppm", "lppm" or "slrm").
...	Arguments passed to dclf.test or mad.test or envelope to control the conduct of the test. Useful arguments include fun to determine the summary function, rinterval to determine the range of r values used in the test, and use.theory described under Details.
exponent	Exponent used in the test statistic. Use exponent=2 for the Diggle-Cressie-Loosmore-Ford test, and exponent=Inf for the Maximum Absolute Deviation test.

nsim	Number of replicates in each stage of the test. A total of nsim * (nsim + 1) simulated point patterns will be generated, and the p-value will be a multiple of 1/(nsim+1).
alternative	Character string specifying the alternative hypothesis. The default (alternative="two.sided") is that the true value of the summary function is not equal to the theoretical value postulated under the null hypothesis. If alternative="less" the alternative hypothesis is that the true value of the summary function is lower than the theoretical value.
leaveout	Optional integer 0, 1 or 2 indicating how to calculate the deviation between the observed summary function and the nominal reference value, when the reference value must be estimated by simulation. See Details.
interpolate	Logical value indicating whether to interpolate the distribution of the test statistic by kernel smoothing, as described in Dao and Genton (2014, Section 5).
savefuns	Logical flag indicating whether to save the simulated function values (from the first stage).
savepatterns	Logical flag indicating whether to save the simulated point patterns (from the first stage).
verbose	Logical value indicating whether to print progress reports.

Details

Performs the Balanced Independent Two-Stage Monte Carlo test proposed by Baddeley et al (2017), an improvement of the Dao-Genton (2014) test.

If X is a point pattern, the null hypothesis is CSR.

If X is a fitted model, the null hypothesis is that model.

The argument use.theory passed to envelope determines whether to compare the summary function for the data to its theoretical value for CSR (use.theory=TRUE) or to the sample mean of simulations from CSR (use.theory=FALSE).

The argument leaveout specifies how to calculate the discrepancy between the summary function for the data and the nominal reference value, when the reference value must be estimated by simulation. The values leaveout=0 and leaveout=1 are both algebraically equivalent (Baddeley et al, 2014, Appendix) to computing the difference observed - reference where the reference is the mean of simulated values. The value leaveout=2 gives the leave-two-out discrepancy proposed by Dao and Genton (2014).

Value

A hypothesis test (object of class "htest" which can be printed to show the outcome of the test.

Author(s)

Adrian Baddeley, Andrew Hardegen, Tom Lawrence, Robin Milne, Gopalan Nair and Suman Rakshit. Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

Dao, N.A. and Genton, M. (2014) A Monte Carlo adjusted goodness-of-fit test for parametric models describing spatial point patterns. *Journal of Graphical and Computational Statistics* **23**, 497–517.

Baddeley, A., Diggle, P.J., Hardegen, A., Lawrence, T., Milne, R.K. and Nair, G. (2014) On tests of spatial pattern based on simulation envelopes. *Ecological Monographs* **84** (3) 477–489.

Baddeley, A., Hardegen, A., Lawrence, L., Milne, R.K., Nair, G.M. and Rakshit, S. (2017) On two-stage Monte Carlo tests of composite hypotheses. *Computational Statistics and Data Analysis* **114**, 75–87.

See Also

Simulation envelopes: `bits.envelope`.

Other tests: `dg.test`, `dclf.test`, `mad.test`.

Examples

```
ns <- if(interactive()) 19 else 4
bits.test(cells, nsim=ns)
bits.test(cells, alternative="less", nsim=ns)
bits.test(cells, nsim=ns, interpolate=TRUE)
```

`blur` *Apply Gaussian Blur to a Pixel Image*

Description

Applies a Gaussian blur to a pixel image.

Usage

```
blur(x, sigma = NULL, ...,
     kernel="gaussian", normalise=FALSE, bleed = TRUE, varcov=NULL)

## S3 method for class 'im'
Smooth(X, sigma = NULL, ...,
                   kernel="gaussian",
                   normalise=FALSE, bleed = TRUE, varcov=NULL)
```

Arguments

`x`, `X`	The pixel image. An object of class `"im"`.
`sigma`	Standard deviation of isotropic Gaussian smoothing kernel.
`...`	Ignored.
`kernel`	String (partially matched) specifying the smoothing kernel. Current options are `"gaussian"`, `"epanechnikov"`, `"quartic"` or `"disc"`.

normalise	Logical flag indicating whether the output values should be divided by the corresponding blurred image of the window itself. See Details.
bleed	Logical flag indicating whether to allow blur to extend outside the original domain of the image. See Details.
varcov	Variance-covariance matrix of anisotropic Gaussian kernel. Incompatible with sigma.

Details

This command applies a Gaussian blur to the pixel image x.

Smooth.im is a method for the generic Smooth for pixel images. It is currently identical to blur, apart from the name of the first argument.

The blurring kernel is the isotropic Gaussian kernel with standard deviation sigma, or the anisotropic Gaussian kernel with variance-covariance matrix varcov. The arguments sigma and varcov are incompatible. Also sigma may be a vector of length 2 giving the standard deviations of two independent Gaussian coordinates, thus equivalent to varcov = diag(sigma^2).

If the pixel values of x include some NA values (meaning that the image domain does not completely fill the rectangular frame) then these NA values are first reset to zero.

The algorithm then computes the convolution $x * G$ of the (zero-padded) pixel image x with the specified Gaussian kernel G.

If normalise=FALSE, then this convolution $x * G$ is returned. If normalise=TRUE, then the convolution $x * G$ is normalised by dividing it by the convolution $w * G$ of the image domain w with the same Gaussian kernel. Normalisation ensures that the result can be interpreted as a weighted average of input pixel values, without edge effects due to the shape of the domain.

If bleed=FALSE, then pixel values outside the original image domain are set to NA. Thus the output is a pixel image with the same domain as the input. If bleed=TRUE, then no such alteration is performed, and the result is a pixel image defined everywhere in the rectangular frame containing the input image.

Computation is performed using the Fast Fourier Transform.

Value

A pixel image with the same pixel array as the input image x.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

interp.im for interpolating a pixel image to a finer resolution, density.ppp for blurring a point pattern, Smooth.ppp for interpolating marks attached to points.

Examples

```
Z <- as.im(function(x,y) { 4 * x^2 + 3 * y }, letterR)
opa <- par(mfrow=c(1,3))
plot(Z)
plot(letterR, add=TRUE)
plot(blur(Z, 0.3, bleed=TRUE))
plot(letterR, add=TRUE)
plot(blur(Z, 0.3, bleed=FALSE))
plot(letterR, add=TRUE)
par(opa)
```

---

blurHeat *Diffusion Blur*

Description

Blur a Pixel Image by Applying Diffusion

Usage

```
blurHeat(X, ...)

## S3 method for class 'im'
blurHeat(X, sigma, ...,
     connect = 8, symmetric = FALSE, k= 1, show = FALSE)

## S3 method for class 'im'
SmoothHeat(X, sigma, ...)
```

Arguments

X	Pixel image (object of class "im").
sigma	Smoothing bandwidth. A numeric value, a pixel image or a function(x,y).
...	Ignored by blurHeat.im.
connect	Grid connectivity: either 4 or 8.
symmetric	Logical value indicating whether to *force* the algorithm to use a symmetric random walk.
k	Integer. Calculations will be performed by repeatedly multiplying the current state by the k-step transition matrix.
show	Logical value indicating whether to plot successive iterations.

Details

The function blurHeat is generic.

This help file documents the method blurHeat.im for pixel images (objects of class "im"). This is currently equivalent to SmoothHeat.im, which is also documented here.

If sigma is a numeric value, then the classical time-dependent heat equation is solved up to time t = sigma^2 starting with the initial condition given by the image X. This has the effect of blurring the input image X.

If sigma is a function or a pixel image, then it is treated as a spatially-variable diffusion rate, and the corresponding heat equation is solved.

This command can be used to calculate the expected value of the diffusion estimator of intensity (densityHeat) when the true intensity is known.

Value

A pixel image on the same raster as X.

Author(s)

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

See Also

densityHeat, blur.

Examples

```
Z <- as.im(function(x,y) { sin(10*x) + sin(9*y) }, letterR)
ZZ <- blurHeat(Z, 0.2)
plot(solist(original=Z, blurred=ZZ), main="")
```

---

boyce *Boyce Index*

Description

Calculate the discrete or continuous Boyce index for a spatial point pattern dataset.

Usage

```
boyce(X, Z, ..., breaks = NULL, halfwidth = NULL)
```

Arguments

X	A spatial point pattern (object of class "ppp").
Z	Habitat suitability classes or habitat suitability index. Either a tessellation (object of class "tess") or a spatial covariate such as a pixel image (object of class "im"), a function(x,y) or one of the letters "a", "b" representing the cartesian coordinates.
...	Additional arguments passed to rhohat.ppp.
breaks	The breakpoint values defining discrete bands of values of the covariate Z for which the discrete Boyce index will be calculated. Either a numeric vector of breakpoints for Z, or a single integer specifying the number of evenly-spaced breakpoints. Incompatible with halfwidth.
halfwidth	The half-width h of the interval $[z-h, z+h]$ which will be used to calculate the continuous Boyce index $B(z)$ for each possible value z of the covariate Z.

Details

Given a spatial point pattern X and some kind of explanatory information Z, this function computes either the index originally defined by Boyce et al (2002) or the 'continuous Boyce index' defined by Hirzel et al (2006).

Boyce et al (2002) defined an index of habitat suitability in which the study region W is first divided into separate subregions $C_1, \ldots, C_m$ based on appropriate scientific considerations. Then we count the number n_j of data points of X that fall in each subregion C_j, measure the area a_j of each subregion C_j, and calculate the index

$$B_j = \frac{n_j/n}{a_j/a}$$

where a is the total area and n is the total number of points in X.

Hirzel et al (2006) defined another version of this index which is based on a continuous spatial covariate. For each possible value z of the covariate Z, consider the region $C(z)$ where the value of the covariate lies between $z-h$ and $z+h$, where h is the chosen 'halfwidth'. The 'continuous Boyce index' is

$$B(z) = \frac{n(z)/n}{a(z)/a}$$

where $n(z)$ is the number of points of X falling in $C(z)$, and $a(z)$ is the area of $C(z)$.

If Z is a tessellation (object of class "tess"), the algorithm calculates the original ('discrete') Boyce index (Boyce et al, 2002) for each tile of the tessellation. The result is another tessellation, identical to Z except that the mark values are the values of the discrete Boyce index.

If Z is a pixel image whose values are categorical (i.e. factor values), then Z is treated as a tessellation, with one tile for each level of the factor. The discrete Boyce index is then calculated. The result is a tessellation with marks that are the values of the discrete Boyce index.

Otherwise, if Z is a spatial covariate such as a pixel image, a function(x,y) or one of the characters "x" or "y", then exactly one of the arguments breaks or halfwidth must be given.

- if halfwidth is given, it should be a single positive number. The continuous Boyce index (Hirzel et al, 2006) is computed using the specified halfwidth h. The result is an object of class "fv" that can be plotted to show $B(z)$ as a function of z.

- if breaks is given, it can be either a numeric vector of possible values of Z defining the breakpoints for the bands of values of Z, or a single integer specifying the number of evenly-spaced breakpoints that should be created. The discrete Boyce index (Boyce et al, 2002) is computed. The result is an object of class "fv" that can be plotted to show the discrete Boyce index as a function of z.

When Z is a spatial covariate (not factor-valued), the calculation is performed using rhohat.ppp (since the Boyce index is a special case of rhohat). Arguments ... passed to rhohat.ppp control the accuracy of the spatial discretisation and other parameters of the algorithm.

Value

A tessellation (object of class "tess") or a function value table (object of class "fv") as explained above.

Author(s)

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

References

Boyce, M.S., Vernier, P.R., Nielsen, S.E. and Schmiegelow, F.K.A. (2002) Evaluating resource selection functions. *Ecological modelling* **157**, 281–300.

Hirzel, A.H., Le Lay, V., Helfer, V., Randin, C. and Guisan, A. (2006) Evaluating the ability of habitat suitability models to predict species presences. *Ecological Modelling* **199**, 142–152.

See Also

rhohat

Examples

```
online <- interactive()
## a simple tessellation
V <- quadrats(Window(bei), 4, 3)
if(online) plot(V)

## discrete Boyce index for a simple tessellation
A <- boyce(bei, V)

if(online) {
 plot(A, do.col=TRUE)
 marks(A)
 tilenames(A)
}

## spatial covariate: terrain elevation
Z <- bei.extra$elev

## continuous Boyce index for terrain elevation
BC <- boyce(bei, Z, halfwidth=10)
```

```
if(online) plot(BC)

## discrete Boyce index for terrain elevation steps of height 5 metres
bk <- c(seq(min(Z), max(Z), by=5), Inf)
BD <- boyce(bei, Z, breaks=bk)

if(online) plot(BD)
```

---

bw.abram.ppp *Abramson's Adaptive Bandwidths For Spatial Point Pattern*

Description

Computes adaptive smoothing bandwidths for a spatial point pattern, according to the inverse-square-root rule of Abramson (1982).

Usage

```
## S3 method for class 'ppp'
bw.abram(X, h0,
         ...,
         at=c("points", "pixels"),
         hp = h0, pilot = NULL, trim=5, smoother=density.ppp)
```

Arguments

X	A point pattern (object of class "ppp") for which the variable bandwidths should be computed.
h0	A scalar value giving the global smoothing bandwidth in the same units as the coordinates of X. The default is h0=bw.ppl(X).
...	Additional arguments passed to as.im to control the pixel resolution, or passed to density.ppp or smoother to control the type of smoothing, when computing the pilot estimate.
at	Character string (partially matched) specifying whether to compute bandwidth values at the points of X (at="points", the default) or to compute bandwidths at every pixel in a fine pixel grid (at="pixels").
hp	Optional. A scalar pilot bandwidth, used for estimation of the pilot density if required. Ignored if pilot is a pixel image (object of class "im"); see below.
pilot	Optional. Specification of a pilot density (possibly unnormalised). If pilot=NULL the pilot density is computed by applying fixed-bandwidth density estimation to X using bandwidth hp. If pilot is a point pattern, the pilot density is is computed using a fixed-bandwidth estimate based on pilot and hp. If pilot is a pixel image (object of class "im"), this is taken to be the (possibly unnormalised) pilot density, and hp is ignored.

trim	A trimming value required to curb excessively large bandwidths. See Details. The default is sensible in most cases.
smoother	Smoother for the pilot. A function or character string, specifying the function to be used to compute the pilot estimate when pilot is NULL or is a point pattern.

Details

This function computes adaptive smoothing bandwidths using the methods of Abramson (1982) and Hall and Marron (1988).

The function bw.abram is generic. The function bw.abram.ppp documented here is the method for spatial point patterns (objects of class "ppp").

If at="points" (the default) a smoothing bandwidth is computed for each point in the pattern X. Alternatively if at="pixels" a smoothing bandwidth is computed for each spatial location in a pixel grid.

Under the Abramson-Hall-Marron rule, the bandwidth at location u is

$$h(u) = \texttt{h0} * \min[\frac{\tilde{f}(u)^{-1/2}}{\gamma}, \texttt{trim}]$$

where $\tilde{f}(u)$ is a pilot estimate of the spatially varying probability density. The variable bandwidths are rescaled by γ, the geometric mean of the $\tilde{f}(u)^{-1/2}$ terms evaluated at the data; this allows the global bandwidth h0 to be considered on the same scale as a corresponding fixed bandwidth. The trimming value trim has the same interpretation as the required 'clipping' of the pilot density at some small nominal value (see Hall and Marron, 1988), to necessarily prevent extreme bandwidths (which can occur at very isolated observations).

The pilot density or intensity is determined as follows:

- If pilot is a pixel image, this is taken as the pilot density or intensity.
- If pilot is NULL, then the pilot intensity is computed as a fixed-bandwidth kernel intensity estimate using density.ppp applied to the data pattern X using the pilot bandwidth hp.
- If pilot is a different point pattern on the same spatial domain as X, then the pilot intensity is computed as a fixed-bandwidth kernel intensity estimate using density.ppp applied to pilot using the pilot bandwidth hp.

In each case the pilot density or intensity is renormalised to become a probability density, and then the Abramson rule is applied.

Instead of calculating the pilot as a fixed-bandwidth density estimate, the user can specify another density estimation procedure using the argument smoother. This should be either a function or the character string name of a function. It will replace density.ppp as the function used to calculate the pilot estimate. The pilot estimate will be computed as smoother(X, sigma=hp, ...) if pilot is NULL, or smoother(pilot, sigma=hp, ...) if pilot is a point pattern. If smoother does not recognise the argument name sigma for the smoothing bandwidth, then hp is effectively ignored, as shown in the Examples.

Value

Either a numeric vector of length npoints(X) giving the Abramson bandwidth for each point (when at = "points", the default), or the entire pixel image of the Abramson bandwidths over the relevant spatial domain (when at = "pixels").

Author(s)

Tilman Davies <Tilman.Davies@otago.ac.nz>. Adapted by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

References

Abramson, I. (1982) On bandwidth variation in kernel estimates — a square root law. *Annals of Statistics*, **10**(4), 1217-1223.

Davies, T.M. and Baddeley, A. (2018) Fast computation of spatially adaptive kernel estimates. *Statistics and Computing*, **28**(4), 937-956.

Davies, T.M., Marshall, J.C., and Hazelton, M.L. (2018) Tutorial on kernel estimation of continuous spatial and spatiotemporal relative risk. *Statistics in Medicine*, **37**(7), 1191-1221.

Hall, P. and Marron, J.S. (1988) Variable window width kernel density estimates of probability densities. *Probability Theory and Related Fields*, **80**, 37-49.

Silverman, B.W. (1986) *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, New York.

See Also

bw.abram

Examples

```
# 'ch' just 58 laryngeal cancer cases
ch <- split(chorley)[[1]]

h <- bw.abram(ch,h0=1,hp=0.7)
length(h)
summary(h)
if(interactive()) hist(h)

# calculate pilot based on all 1036 observations
h.pool <- bw.abram(ch,h0=1,hp=0.7,pilot=chorley)
length(h.pool)
summary(h.pool)
if(interactive()) hist(h.pool)

# get full image used for 'h' above
him <- bw.abram(ch,h0=1,hp=0.7,at="pixels")
plot(him);points(ch,col="grey")

# use Voronoi-Dirichlet pilot ('hp' is ignored)
hvo <- bw.abram(ch, h0=1, smoother=densityVoronoi)
```

bw.bdh *Adjust Bandwidth Selection Rule to Account for Inhomogeneity*

Description

Calculate a bandwidth for kernel estimation of the inhomogeneous pair correlation function. Given a bandwidth selection rule or bandwidth value which would be appropriate for a homogeneous point pattern, this function adjusts the bandwidth to account for inhomogeneity.

Usage

```
bw.bdh(X, lambda=NULL, ..., base=bw.stoyan, k=2)
```

Arguments

X	A point pattern (object of class "ppp").
lambda	Optional. Values of the estimated intensity function of X. Either a vector giving the intensity values at the points of the pattern X, a pixel image (object of class "im") giving the intensity values at all locations, a fitted point process model (object of class "ppm", "kppm" or "dppm") or a function(x,y) which can be evaluated to give the intensity value at any location. If lambda is missing or NULL, the intensity will be estimated from X using density.ppp.
...	Arguments passed to density.ppp to control the estimation of intensity (if lambda=NULL), or arguments passed to base if it is a function.
base	Bandwidth selection rule, or bandwidth value, that will be adjusted. Either a single numeric value, or a function that will be applied to the pattern X to obtain a bandwidth value, or a character string giving the name of such a function.
k	Integer exponent for calculating the adjustment factor.

Details

This function calculates a bandwidth value for kernel estimation of the inhomogeneous pair correlation function using pcfinhom.

It takes a bandwidth value or bandwidth selection rule that would be appropriate if the point process were homogeneous, adjusts the bandwidth to account for inhomogeneity, and returns the adjusted bandwidth value.

The adjusted (inhomogeneous) bandwidth is the original (homogeneous) bandwidth multiplied by the Baddeley, Davies and Hazelton (2025) variance controlling adjustment factor.

First a numerical bandwidth value, appropriate for a homogeneous process, is calculated. The default is to apply the extrapolated Stoyan rule-of-thumb bw.stoyan to the point pattern X. If base is specified, it may be either a numeric value for the bandwidth, or another function that will be applied to X to calculate a bandwidth value.

Next the intensity value at each point of X is evaluated. The argument lambda may be:

- a numeric vector giving the intensity values at the points of X;

- a pixel image (object of class "im") giving the intensity values at all locations;
- a fitted point process model (object of class "ppm", "kppm" or "dppm"). The intensity of the fitted model will be evaluated at each point of X. By default, the fitted model is updated by re-fitting it to X before the intensity is evaluated. Updating can be disabled by setting update=FALSE.
- a function(x,y) which can be evaluated to give the intensity value at any location;
- missing or NULL. In this case, the intensity will be estimated from X using density.ppp. Arguments ... controlling the kernel estimation include sigma, varcov and kernel.

Finally the bandwidth is adjusted by multiplying it by the Baddeley, Davies and Hazelton (2025) variance-controlling factor

$$a = (n^{-1}\sum_i \lambda_i)\,(n^{-1}\sum_i \lambda_i^{-k})^{1/k}$$

where λ_i is the value of lambda for the ith data point X[i].

When k=2 (the default), the adjustment factor is

$$a = (n^{-1}\sum_i \lambda_i)\sqrt{n^{-1}\sum_i \lambda_i^{-2}}$$

Value

A finite positive numerical value giving the selected bandwidth (the standard deviation of the smoothing kernel). The result has an attribute "adjust" giving the adjustment factor a.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Tilman Davies <Tilman.Davies@otago.ac.nz> and Martin Hazelton <Martin.Hazelton@otago.ac.nz>.

References

Baddeley, A., Davies, T.M. and Hazelton, M.L. (2025) An improved estimator of the pair correlation function of a spatial point process. *Biometrika*, to appear.

See Also

bw.stoyan, bw.pcf, pcfinhom

Examples

```
if(require(spatstat.model)) {
  fit <- ppm(japanesepines ~ x)
  (b <- bw.bdh(japanesepines, fit))
  attr(b, "adjust")
}
```

bw.CvL *Cronie and van Lieshout's Criterion for Bandwidth Selection for Kernel Density*

Description

Uses Cronie and van Lieshout's criterion based on Cambell's formula to select a smoothing bandwidth for the kernel estimation of point process intensity.

Usage

```
bw.CvL(X, ..., srange = NULL, ns = 16, sigma = NULL, warn=TRUE)
```

Arguments

X	A point pattern (object of class "ppp").
...	Ignored.
srange	Optional numeric vector of length 2 giving the range of values of bandwidth to be searched.
ns	Optional integer giving the number of values of bandwidth to search.
sigma	Optional. Vector of values of the bandwidth to be searched. Overrides the values of ns and srange.
warn	Logical. If TRUE, a warning is issued if the optimal value of the cross-validation criterion occurs at one of the ends of the search interval.

Details

This function selects an appropriate bandwidth sigma for the kernel estimator of point process intensity computed by density.ppp.

The bandwidth σ is chosen to minimise the discrepancy between the area of the observation window and the sum of reciprocal estimated intensity values at the points of the point process

$$\mathrm{CvL}(\sigma) = (|W| - \sum_i 1/\hat{\lambda}(x_i))^2$$

where the sum is taken over all the data points x_i, and where $\hat{\lambda}(x_i)$ is the kernel-smoothing estimate of the intensity at x_i with smoothing bandwidth σ.

The value of $\mathrm{CvL}(\sigma)$ is computed directly, using density.ppp, for ns different values of σ between srange[1] and srange[2].

Value

A single numerical value giving the selected bandwidth. The result also belongs to the class "bw.optim" (see bw.optim.object) which can be plotted to show the bandwidth selection criterion as a function of sigma.

Author(s)

Ottmar Cronie <ottmar@chalmers.se> and Marie-Colette van Lieshout <Marie-Colette.van.Lieshout@cwi.nl>. Adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

Cronie, O and Van Lieshout, M N M (2018) A non-model-based approach to bandwidth selection for kernel estimators of spatial intensity functions, *Biometrika*, **105**, 455-462.

See Also

density.ppp, bw.optim.object.

Alternative methods: bw.diggle, bw.scott, bw.ppl, bw.frac.

For adaptive smoothing bandwidths, use bw.CvL.adaptive.

Examples

```
if(interactive()) {
  b <- bw.CvL(redwood)
  b
  plot(b, main="Cronie and van Lieshout bandwidth criterion for redwoods")
  plot(density(redwood, b))
  plot(density(redwood, bw.CvL))
}
```

---

bw.CvL.adaptive — *Select Adaptive Bandwidth for Kernel Estimation Using Cronie-Van Lieshout Criterion*

Description

Uses the Cronie-Van Lieshout criterion to select the global smoothing bandwidth for adaptive kernel estimation of point process intensity.

Usage

```
bw.CvL.adaptive(X, ...,
            hrange = NULL, nh = 16, h=NULL,
            bwPilot = bw.scott.iso(X),
            edge = FALSE, diggle = TRUE)
```

Arguments

X	A point pattern (object of class "ppp").
...	Additional arguments passed to densityAdaptiveKernel.ppp.
hrange	Optional numeric vector of length 2 giving the range of values of global bandwidth h to be searched.
nh	Optional integer giving the number of values of bandwidth h to search.
h	Optional. Vector of values of the bandwidth to be searched. Overrides the values of nh and hrange.
bwPilot	Pilot bandwidth. A scalar value in the same units as the coordinates of X. The smoothing bandwidth for computing an initial estimate of intensity using density.ppp.
edge	Logical value indicating whether to apply edge correction.
diggle	Logical. If TRUE, use the Jones-Diggle improved edge correction, which is more accurate but slower to compute than the default correction.

Details

This function selects an appropriate value of global bandwidth h0 for adaptive kernel estimation of the intensity function for the point pattern X.

In adaptive estimation, each point in the point pattern is subjected to a different amount of smoothing, controlled by data-dependent or spatially-varying bandwidths. The global bandwidth h0 is a scale factor which is used to adjust all of the data-dependent bandwidths according to the Abramson (1982) square-root rule.

This function considers each candidate value of bandwidth h, performs the smoothing steps described above, extracts the adaptively-estimated intensity values $\hat{\lambda}(x_i)$ at each data point x_i, and calculates the Cronie-Van Lieshout criterion

$$\mathrm{CvL}(h) = \sum_{i=1}^{n} \frac{1}{\hat{\lambda}(x_i)}.$$

The value of h which minimises the squared difference

$$LP2(h) = (CvL(h) - |W|)^2$$

(where |W| is the area of the window of X) is selected as the optimal global bandwidth.

Bandwidths h are physical distance values expressed in the same units as the coordinates of X.

Value

A single numerical value giving the selected global bandwidth. The result also belongs to the class "bw.optim" (see bw.optim.object) which can be plotted to show the bandwidth selection criterion as a function of sigma.

Author(s)

Marie-Colette Van Lieshout. Modified by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

References

Abramson, I. (1982) On bandwidth variation in kernel estimates — a square root law. *Annals of Statistics*, **10**(4), 1217-1223.

Cronie, O and Van Lieshout, M N M (2018) A non-model-based approach to bandwidth selection for kernel estimators of spatial intensity functions, *Biometrika*, **105**, 455-462.

Van Lieshout, M.N.M. (2021) Infill asymptotics for adaptive kernel estimators of spatial intensity. *Australian and New Zealand Journal of Statistics* **63** (1) 159–181.

See Also

bw.optim.object.

adaptive.density, densityAdaptiveKernel.ppp, bw.abram.ppp, density.ppp.

To select a *fixed* smoothing bandwidth using the Cronie-Van Lieshout criterion, use bw.CvL.

Examples

```
online <- interactive()
if(online) {
  h0 <- bw.CvL.adaptive(redwood3)
} else {
  ## faster computation for package checker
  h0 <- bw.CvL.adaptive(redwood3, nh=8,
                        hrange=c(1/4, 4) * bw.diggle(redwood3))
}
plot(h0)
plot(as.fv(h0), CvL ~ h)
if(online) {
  Z <- densityAdaptiveKernel(redwood3, h0)
  plot(Z)
}
```

---

bw.CvLHeat — *Bandwidth Selection for Diffusion Smoother by Cronie-van Lieshout Rule*

Description

Selects an optimal bandwidth for diffusion smoothing using the Cronie-van Lieshout rule.

Usage

```
bw.CvLHeat(X, ..., srange=NULL, ns=16, sigma=NULL,
         leaveoneout=TRUE, verbose = TRUE)
```

Arguments

X	Point pattern (object of class "ppp").
...	Arguments passed to densityHeat.ppp.
srange	Numeric vector of length 2 specifying a range of bandwidths to be considered.
ns	Integer. Number of candidate bandwidths to be considered.
sigma	Maximum smoothing bandwidth. A numeric value, or a pixel image, or a function(x,y). Alternatively a numeric vector containing a sequence of candidate bandwidths.
leaveoneout	Logical value specifying whether intensity values at data points should be estimated using the leave-one-out rule.
verbose	Logical value specifying whether to print progress reports.

Details

This algorithm selects the optimal global bandwidth for kernel estimation of intensity for the dataset X using diffusion smoothing densityHeat.ppp.

If sigma is a numeric value, the algorithm finds the optimal bandwidth tau <= sigma.

If sigma is a pixel image or function, the algorithm finds the optimal fraction 0 < f <= 1 such that smoothing with f * sigma would be optimal.

Value

A numerical value giving the selected bandwidth (if sigma was a numeric value) or the selected fraction of the maximum bandwidth (if sigma was a pixel image or function). The result also belongs to the class "bw.optim" which can be plotted.

Author(s)

Adrian Baddeley.

See Also

bw.pplHeat for an alternative method.

densityHeat.ppp

Examples

```
online <- interactive()
if(!online) op <- spatstat.options(npixel=32)
f <- function(x,y) { dnorm(x, 2.3, 0.1) * dnorm(y, 2.0, 0.2) }
X <- rpoint(15, f, win=letterR)
plot(X)
b <- bw.CvLHeat(X, sigma=0.25)
b
plot(b)
if(!online) spatstat.options(op)
```

bw.diggle *Cross Validated Bandwidth Selection for Kernel Density*

Description

Uses cross-validation to select a smoothing bandwidth for the kernel estimation of point process intensity.

Usage

```
bw.diggle(X, ..., correction="good", hmax=NULL, nr=512, warn=TRUE)
```

Arguments

X	A point pattern (object of class "ppp").
...	Ignored.
correction	Character string passed to Kest determining the edge correction to be used to calculate the K function.
hmax	Numeric. Maximum value of bandwidth that should be considered.
nr	Integer. Number of steps in the distance value r to use in computing numerical integrals.
warn	Logical. If TRUE, issue a warning if the minimum of the cross-validation criterion occurs at one of the ends of the search interval.

Details

This function selects an appropriate bandwidth sigma for the kernel estimator of point process intensity computed by density.ppp.

The bandwidth σ is chosen to minimise the mean-square error criterion defined by Diggle (1985). The algorithm uses the method of Berman and Diggle (1989) to compute the quantity

$$M(\sigma) = \frac{\mathrm{MSE}(\sigma)}{\lambda^2} - g(0)$$

as a function of bandwidth σ, where $\mathrm{MSE}(\sigma)$ is the mean squared error at bandwidth σ, while λ is the mean intensity, and g is the pair correlation function. See Diggle (2003, pages 115-118) for a summary of this method.

The result is a numerical value giving the selected bandwidth. The result also belongs to the class "bw.optim" which can be plotted to show the (rescaled) mean-square error as a function of sigma.

Value

A single numerical value giving the selected bandwidth. The result also belongs to the class "bw.optim" (see bw.optim.object) which can be plotted to show the bandwidth selection criterion as a function of sigma.

Definition of bandwidth

The smoothing parameter `sigma` returned by `bw.diggle` (and displayed on the horizontal axis of the plot) corresponds to `h/2`, where `h` is the smoothing parameter described in Diggle (2003, pages 116-118) and Berman and Diggle (1989). In those references, the smoothing kernel is the uniform density on the disc of radius `h`. In `density.ppp`, the smoothing kernel is the isotropic Gaussian density with standard deviation `sigma`. When replacing one kernel by another, the usual practice is to adjust the bandwidths so that the kernels have equal variance (cf. Diggle 2003, page 118). This implies that `sigma = h/2`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

Berman, M. and Diggle, P. (1989) Estimating weighted integrals of the second-order intensity of a spatial point process. *Journal of the Royal Statistical Society, series B* **51**, 81–92.

Diggle, P.J. (1985) A kernel method for smoothing point process data. *Applied Statistics* (Journal of the Royal Statistical Society, Series C) **34** (1985) 138–147.

Diggle, P.J. (2003) *Statistical analysis of spatial point patterns*, Second edition. Arnold.

See Also

`density.ppp`, `bw.optim.object`.

Alternative methods: `bw.ppl`, `bw.scott`, `bw.CvL`, `bw.frac`.

Examples

```
attach(split(lansing))
b <- bw.diggle(hickory)
plot(b, ylim=c(-2, 0), main="Cross validation for hickories")
if(interactive()) {
 plot(density(hickory, b))
}
```

---

bw.frac *Bandwidth Selection Based on Window Geometry*

Description

Select a smoothing bandwidth for smoothing a point pattern, based only on the geometry of the spatial window. The bandwidth is a specified quantile of the distance between two independent random points in the window.

Usage

```
bw.frac(X, ..., f=1/4)
```

Arguments

X	A window (object of class "owin") or point pattern (object of class "ppp") or other data which can be converted to a window using as.owin.
...	Arguments passed to distcdf.
f	Probability value (between 0 and 1) determining the quantile of the distribution.

Details

This function selects an appropriate bandwidth sigma for the kernel estimator of point process intensity computed by density.ppp.

The bandwidth σ is computed as a quantile of the distance between two independent random points in the window. The default is the lower quartile of this distribution.

If $F(r)$ is the cumulative distribution function of the distance between two independent random points uniformly distributed in the window, then the value returned is the quantile with probability f. That is, the bandwidth is the value r such that $F(r) = f$.

The cumulative distribution function $F(r)$ is computed using distcdf. We then we compute the smallest number r such that $F(r) \geq f$.

Value

A numerical value giving the selected bandwidth. The result also belongs to the class "bw.frac" which can be plotted to show the cumulative distribution function and the selected quantile.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

For estimating point process intensity, see density.ppp, bw.diggle, bw.ppl, bw.scott, bw.CvL.

For other smoothing purposes, see bw.stoyan, bw.smoothppp, bw.relrisk.

Examples

```
h <- bw.frac(letterR)
h
plot(h, main="bw.frac(letterR)")
```

bw.optim.object *Class of Optimized Bandwidths*

Description

An object of the class "bw.optim" represents a tuning parameter (usually a smoothing bandwidth) that has been selected automatically. The object can be used as if it were a numerical value, but it can also be plotted to show the optimality criterion.

Details

An object of the class "bw.optim" represents the numerical value of a smoothing bandwidth, a threshold, or a similar tuning parameter, that has been selected by optimising a criterion such as cross-validation.

The object is a numerical value, with some attributes that retain information about how the value was selected.

Attributes include the vector of candidate values that were examined, the corresponding values of the optimality criterion, the name of the parameter, the name of the optimality criterion, and the units in which the parameter is measured.

There are methods for print, plot, summary, as.data.frame and as.fv for the class "bw.optim".

The print method simply prints the numerical value of the parameter. The summary method prints this value, and states how this value was selected.

The plot method produces a plot of the optimisation criterion against the candidate value of the parameter. The as.data.frame and as.fv methods extract this graphical information as a data frame or function table, respectively.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

Functions which produce objects of class bw.optim include bw.CvL, bw.CvL.adaptive, bw.diggle, bw.lppl, bw.pcf, bw.ppl, bw.relrisk, bw.relrisk.lpp, bw.smoothppp and bw.voronoi

Examples

```
Ns <- if(interactive()) 32 else 3
b <- bw.ppl(redwood, srange=c(0.02, 0.07), ns=Ns)
b
summary(b)
plot(b)
```

bw.pcf *Cross Validated Bandwidth Selection for Pair Correlation Function*

Description

Uses composite likelihood or generalized least squares cross-validation to select a smoothing bandwidth for the kernel estimation of pair correlation function.

Usage

```
bw.pcf(X, ..., rmax=NULL, nr=10000,
         cv.method=c("compLik", "leastSQ", "oracle"),
         leaveoneout=TRUE, simple=TRUE,
         fast=TRUE, srange=NULL, ns=32, use.count=TRUE,
         gtrue=NULL,
         verbose=FALSE, warn=TRUE)
```

Arguments

X	A point pattern (object of class "ppp").
...	Additional arguments passed to pcf.
rmax	Optional. Numeric value. Maximum value of the spatial lag distance r for which $g(r)$ should be evaluated.
nr	Integer. Number of subintervals for discretization of [0, rmax] to use in computing numerical integrals.
cv.method	Choice of cross validation method: either "compLik", "leastSQ" or "oracle" (partially matched).
leaveoneout	Logical value specifying whether to use leave-one-out estimators. See Details.
simple	Logical. Whether to use simple removal of spatial lag distances. See Details.
fast	Logical value indicating whether to find the optimal value by an optimization algorithm (fast=TRUE, the default) or by evaluating the objective function on an equally-spaced grid of bandwidth values (fast=FALSE).
srange	Optional. Numeric vector of length 2 giving the range of bandwidth values that should be searched to find the optimum bandwidth.
ns	Integer. Number of values of bandwidths at which to evaluate the objective function, when fast=FALSE.
use.count	Logical value specifying the benchmark for the calculation when cv.method="martin". In this calculation, the sum of values $1/g(d_{i,j})$ over all pairwise distances is compared to a benchmark. If use.count=TRUE (the default), the benchmark is simply the total number of pairs of points contributing to the sum. If use.count=FALSE, the theoretical expected value is used as the benchmark.
gtrue	Function in the R language giving the true pair correlation function, when cv.method="oracle".
verbose	Logical value indicating whether to print progress reports during the optimization procedure.

warn Logical. If TRUE, issue a warning if the optimum value of the cross-validation criterion occurs at one of the ends of the search interval.

Details

This function selects an appropriate bandwidth bw for the kernel estimator of the pair correlation function of a point process intensity computed by pcf.

With cv.method="leastSQ", the bandwidth h is chosen to minimise an unbiased estimate of the integrated mean-square error criterion $M(h)$ defined in equation (4) in Guan (2007a). The code implements the fast algorithm of Jalilian and Waagepetersen (2018).

With cv.method="compLik", the bandwidth h is chosen to maximise a likelihood cross-validation criterion $CV(h)$ defined in equation (6) of Guan (2007b).

$$M(b) = \frac{\mathrm{MSE}(\sigma)}{\lambda^2} - g(0)$$

With cv.method="oracle", the true pair correlation function must be provided as the argument gtrue. The bandwidth h is chosen to minimise the integrated squared difference between the pcf estimate and the true pcf,

$$M(h) = \int_0^{\mathrm{rmax}} (\hat{g}(r) - g(r))^2 dr$$

The result is a numerical value giving the selected bandwidth.

Value

A numerical value giving the selected bandwidth. The result also belongs to the class "bw.optim" which can be plotted.

Definition of bandwidth

The bandwidth bw returned by bw.pcf is the standard deviation of the smoothing kernel, following the standard convention in R. As mentioned in the documentation for density.default and pcf.ppp, this differs from other definitions of bandwidth that can be found in the literature. The scale parameter h, which is called the bandwidth in some literature, is defined differently. For example for the Epanechnikov kernel, h is the half-width of the kernel, and bw=h/sqrt(5).

Author(s)

Rasmus Waagepetersen and Abdollah Jalilian. Adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu.a Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>. Further hacked by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Martin Hazelton <Martin.Hazelton@otago.ac.nz> and Tilman Davies <Tilman.Davies@otago.ac.nz>.

References

Baddeley, A., Davies, T.M. and Hazelton, M.L. (2025) An improved estimator of the pair correlation function of a spatial point process. *Biometrika*, to appear.

Guan, Y. (2007a). A composite likelihood cross-validation approach in selecting bandwidth for the estimation of the pair correlation function. *Scandinavian Journal of Statistics*, **34**(2), 336–346.

Guan, Y. (2007b). A least-squares cross-validation bandwidth selection approach in pair correlation function estimations. *Statistics & Probability Letters*, **77**(18), 1722–1729.

Jalilian, A. and Waagepetersen, R. (2018) Fast bandwidth selection for estimation of the pair correlation function. *Journal of Statistical Computation and Simulation*, **88**(10), 2001–2011. https://www.tandfonline.com/doi/full/10.1080/00949655.2018.1428606

See Also

pcf

Examples

```
b <- bw.pcf(redwood3)
plot(pcf(redwood3, bw=b))
```

`bw.pcfinhom` *Cross Validated Bandwidth Selection for Inhomogeneous Pair Correlation Function*

Description

Uses composite likelihood or generalized least squares cross-validation to select a smoothing bandwidth for the kernel estimation of the inhomogeneous pair correlation function.

Usage

```
bw.pcfinhom(X, lambda=NULL, ..., rmax=NULL, nr=10000,
          cv.method=c("compLik", "leastSQ", "oracle"),
          leaveoneout=TRUE, simple=TRUE,
          fast=TRUE, srange=NULL, ns=32, use.count=TRUE,
          gtrue=NULL,
          verbose=FALSE, warn=TRUE)
```

Arguments

`X`	A point pattern (object of class `"ppp"`).
`lambda`	Optional. Values of the estimated intensity function. Either a vector giving the intensity values at the points of the pattern `X`, a pixel image (object of class `"im"`) giving the intensity values at all locations, a fitted point process model (object of class `"ppm"`, `"kppm"` or `"dppm"`) or a `function(x,y)` which can be evaluated to give the intensity value at any location.
`...`	Additional arguments passed to pcfinhom.
`rmax`	Optional. Numeric value. Maximum value of the spatial lag distance r for which $g(r)$ should be evaluated.

nr	Integer. Number of subintervals for discretization of [0, rmax] to use in computing numerical integrals.
cv.method	Choice of cross validation method: either "compLik", "leastSQ" or "oracle" (partially matched).
leaveoneout	Logical value specifying whether to use leave-one-out estimators. See Details.
simple	Logical. Whether to use simple removal of spatial lag distances. See Details.
fast	Logical value indicating whether to find the optimal value by an optimization algorithm (fast=TRUE, the default) or by evaluating the objective function on an equally-spaced grid of bandwidth values (fast=FALSE).
srange	Optional. Numeric vector of length 2 giving the range of bandwidth values that should be searched to find the optimum bandwidth.
ns	Integer. Number of values of bandwidths at which to evaluate the objective function, when fast=FALSE.
use.count	Logical value specifying the benchmark for the calculation when cv.method="martin". In this calculation, the sum of values $1/g(d_{i,j})$ over all pairwise distances is compared to a benchmark. If use.count=TRUE (the default), the benchmark is simply the total number of pairs of points contributing to the sum. If use.count=FALSE, the theoretical expected value is used as the benchmark.
gtrue	Function in the R language giving the true pair correlation function, when cv.method="oracle".
verbose	Logical value indicating whether to print progress reports during the optimization procedure.
warn	Logical. If TRUE, issue a warning if the optimum value of the cross-validation criterion occurs at one of the ends of the search interval.

Details

This function selects an appropriate bandwidth bw for the kernel estimator of the pair correlation function of a point process intensity computed by pcfinhom.

With cv.method="leastSQ", the bandwidth h is chosen to minimise an unbiased estimate of the integrated mean-square error criterion $M(h)$ defined in equation (4) in Guan (2007a). The code implements the fast algorithm of Jalilian and Waagepetersen (2018).

With cv.method="compLik", the bandwidth h is chosen to maximise a likelihood cross-validation criterion $CV(h)$ defined in equation (6) of Guan (2007b).

$$M(b) = \frac{\mathrm{MSE}(\sigma)}{\lambda^2} - g(0)$$

With cv.method="oracle", the true pair correlation function must be provided as the argument gtrue. The bandwidth h is chosen to minimise the integrated squared difference between the pcf estimate and the true pcf,

$$M(h) = \int_0^{\mathrm{rmax}} (\hat{g}(r) - g(r))^2 dr$$

The result is a numerical value giving the selected bandwidth.

Value

A numerical value giving the selected bandwidth. The result also belongs to the class "bw.optim" which can be plotted.

Definition of bandwidth

The bandwidth bw returned by bw.pcfinhom is the standard deviation of the smoothing kernel, following the standard convention in R. As mentioned in the documentation for density.default and pcf.ppp, this differs from other definitions of bandwidth that can be found in the literature. The scale parameter h, which is called the bandwidth in some literature, is defined differently. For example for the Epanechnikov kernel, h is the half-width of the kernel, and bw=h/sqrt(5).

Author(s)

Rasmus Waagepetersen and Abdollah Jalilian. Adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu.a Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>. Hacked by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Martin Hazelton <Martin.Hazelton@otago.ac.nz> and Tilman Davies <Tilman.Davies@otago.ac.nz>.

References

Baddeley, A., Davies, T.M. and Hazelton, M.L. (2025) An improved estimator of the pair correlation function of a spatial point process. *Biometrika*, to appear.

Guan, Y. (2007a). A composite likelihood cross-validation approach in selecting bandwidth for the estimation of the pair correlation function. *Scandinavian Journal of Statistics*, **34**(2), 336–346.

Guan, Y. (2007b). A least-squares cross-validation bandwidth selection approach in pair correlation function estimations. *Statistics & Probability Letters*, **77**(18), 1722–1729.

Jalilian, A. and Waagepetersen, R. (2018) Fast bandwidth selection for estimation of the pair correlation function. *Journal of Statistical Computation and Simulation*, **88**(10), 2001–2011. https://www.tandfonline.com/doi/full/10.1080/00949655.2018.1428606

See Also

pcfinhom

Examples

```
b <- bw.pcfinhom(japanesepines)
plot(pcfinhom(japanesepines, bw=b))
```

bw.ppl *Likelihood Cross Validation Bandwidth Selection for Kernel Density*

Description

Uses likelihood cross-validation to select a smoothing bandwidth for the kernel estimation of point process intensity.

Usage

```
bw.ppl(X, ..., srange=NULL, ns=16, sigma=NULL, varcov1=NULL,
       weights=NULL, shortcut=TRUE, warn=TRUE)
```

Arguments

X	A point pattern (object of class "ppp").
srange	Optional numeric vector of length 2 giving the range of values of bandwidth to be searched.
ns	Optional integer giving the number of values of bandwidth to search.
sigma	Optional. Vector of values of the bandwidth to be searched. Overrides the values of ns and srange.
varcov1	Optional. Variance-covariance matrix matrix of the kernel with bandwidth $h = 1$. See section on Anisotropic Smoothing.
weights	Optional. Numeric vector of weights for the points of X. Argument passed to density.ppp.
...	Additional arguments passed to density.ppp.
shortcut	Logical value indicating whether to speed up the calculation by omitting the integral term in the cross-validation criterion.
warn	Logical. If TRUE, issue a warning if the maximum of the cross-validation criterion occurs at one of the ends of the search interval.

Details

This function selects an appropriate bandwidth sigma for the kernel estimator of point process intensity computed by density.ppp.

The bandwidth σ is chosen to maximise the point process likelihood cross-validation criterion

$$\mathrm{LCV}(\sigma) = \sum_i \log \hat{\lambda}_{-i}(x_i) - \int_W \hat{\lambda}(u)\,\mathrm{d}u$$

where the sum is taken over all the data points x_i, where $\hat{\lambda}_{-i}(x_i)$ is the leave-one-out kernel-smoothing estimate of the intensity at x_i with smoothing bandwidth σ, and $\hat{\lambda}(u)$ is the kernel-smoothing estimate of the intensity at a spatial location u with smoothing bandwidth σ. See Loader(1999, Section 5.3).

The value of LCV(σ) is computed directly, using density.ppp, for ns different values of σ between srange[1] and srange[2].

The result is a numerical value giving the selected bandwidth. The result also belongs to the class "bw.optim" which can be plotted to show the (rescaled) mean-square error as a function of sigma.

If shortcut=TRUE (the default), the computation is accelerated by omitting the integral term in the equation above. This is valid because the integral is approximately constant.

Value

A numerical value giving the selected bandwidth. The result also belongs to the class "bw.optim" which can be plotted.

Anisotropic Smoothing

Anisotropic kernel smoothing is available in density.ppp using the argument varcov to specify the variance-covariance matrix of the anisotropic kernel. In order to choose the matrix varcov, the user can call bw.ppl using the argument varcov1 to specify a 'template' matrix. Scalar multiples of varcov1 will be considered and the optimal scale factor will be determined. That is, bw.ppl will try smoothing the data using varcov = h^2 * varcov1 for different values of h. The result of bw.ppl will be the optimal value of h.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

Loader, C. (1999) *Local Regression and Likelihood*. Springer, New York.

See Also

density.ppp, bw.diggle, bw.scott, bw.CvL, bw.frac.

Examples

```
if(interactive()) {
  b <- bw.ppl(redwood)
  plot(b, main="Likelihood cross validation for redwoods")
  plot(density(redwood, b))
}
```

bw.pplHeat *Bandwidth Selection for Diffusion Smoother by Likelihood Cross-Validation*

Description

Selects an optimal bandwidth for diffusion smoothing by point process likelihood cross-validation.

Usage

```
bw.pplHeat(X, ..., srange=NULL, ns=16, sigma=NULL,
         leaveoneout=TRUE, verbose = TRUE)
```

Arguments

X	Point pattern (object of class "ppp").
...	Arguments passed to densityHeat.ppp.
srange	Numeric vector of length 2 specifying a range of bandwidths to be considered.
ns	Integer. Number of candidate bandwidths to be considered.
sigma	Maximum smoothing bandwidth. A numeric value, or a pixel image, or a function(x,y). Alternatively a numeric vector containing a sequence of candidate bandwidths.
leaveoneout	Logical value specifying whether intensity values at data points should be estimated using the leave-one-out rule.
verbose	Logical value specifying whether to print progress reports.

Details

This algorithm selects the optimal global bandwidth for kernel estimation of intensity for the dataset X using diffusion smoothing densityHeat.ppp.

If sigma is a numeric value, the algorithm finds the optimal bandwidth tau <= sigma.

If sigma is a pixel image or function, the algorithm finds the optimal fraction 0 < f <= 1 such that smoothing with f * sigma would be optimal.

Value

A numerical value giving the selected bandwidth (if sigma was a numeric value) or the selected fraction of the maximum bandwidth (if sigma was a pixel image or function). The result also belongs to the class "bw.optim" which can be plotted.

Author(s)

Adrian Baddeley and Tilman Davies.

See Also

bw.CvLHeat for an alternative method.

densityHeat.ppp

Examples

```
online <- interactive()
if(!online) op <- spatstat.options(npixel=32)
f <- function(x,y) { dnorm(x, 2.3, 0.1) * dnorm(y, 2.0, 0.2) }
X <- rpoint(15, f, win=letterR)
plot(X)
b <- bw.pplHeat(X, sigma=0.25)
b
plot(b)
if(!online) spatstat.options(op)
```

---

bw.relrisk *Cross Validated Bandwidth Selection for Relative Risk Estimation*

Description

Uses cross-validation to select a smoothing bandwidth for the estimation of relative risk.

Usage

```
   bw.relrisk(X, ...)

   ## S3 method for class 'ppp'
bw.relrisk(X, method = "likelihood", ...,
           nh = spatstat.options("n.bandwidth"),
           hmin=NULL, hmax=NULL, warn=TRUE)
```

Arguments

X	A multitype point pattern (object of class "ppp" which has factor valued marks).
method	Character string determining the cross-validation method. Current options are "likelihood", "leastsquares" or "weightedleastsquares".
nh	Number of trial values of smoothing bandwith sigma to consider. The default is 32.
hmin, hmax	Optional. Numeric values. Range of trial values of smoothing bandwith sigma to consider. There is a sensible default.
warn	Logical. If TRUE, issue a warning if the minimum of the cross-validation criterion occurs at one of the ends of the search interval.
...	Additional arguments passed to density.ppp or to other methods for bw.relrisk.

Details

This function selects an appropriate bandwidth for the nonparametric estimation of relative risk using relrisk.

Consider the indicators y_{ij} which equal 1 when data point x_i belongs to type j, and equal 0 otherwise. For a particular value of smoothing bandwidth, let $\hat{p}_j(u)$ be the estimated probabilities that a point at location u will belong to type j. Then the bandwidth is chosen to minimise either the negative likelihood, the squared error, or the approximately standardised squared error, of the indicators y_{ij} relative to the fitted values $\hat{p}_j(x_i)$. See Diggle (2003) or Baddeley et al (2015).

The result is a numerical value giving the selected bandwidth sigma. The result also belongs to the class "bw.optim" allowing it to be printed and plotted. The plot shows the cross-validation criterion as a function of bandwidth.

The range of values for the smoothing bandwidth sigma is set by the arguments hmin, hmax. There is a sensible default, based on multiples of Stoyan's rule of thumb bw.stoyan.

If the optimal bandwidth is achieved at an endpoint of the interval [hmin, hmax], the algorithm will issue a warning (unless warn=FALSE). If this occurs, then it is probably advisable to expand the interval by changing the arguments hmin, hmax.

Computation time depends on the number nh of trial values considered, and also on the range [hmin, hmax] of values considered, because larger values of sigma require calculations involving more pairs of data points.

Value

A single numerical value giving the selected bandwidth. The result also belongs to the class "bw.optim" (see bw.optim.object) which can be plotted to show the bandwidth selection criterion as a function of sigma.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>.

References

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. Chapman and Hall/CRC Press.

Diggle, P.J. (2003) *Statistical analysis of spatial point patterns*, Second edition. Arnold.

Kelsall, J.E. and Diggle, P.J. (1995) Kernel estimation of relative risk. *Bernoulli* **1**, 3–16.

See Also

relrisk, bw.stoyan.

bw.optim.object.

Examples

```
b <- bw.relrisk(urkiola)
b
plot(b)
b <- bw.relrisk(urkiola, hmax=20)
plot(b)
```

`bw.relriskHeatppp` *Bandwidth Selection for Relative Risk using Diffusion*

Description

Performs data-based bandwidth selection for the diffusion estimate of relative risk `relriskHeat.ppp` using either likelihood cross-validation or least squares

Usage

```
bw.relriskHeatppp(X, ..., method = c("likelihood", "leastsquares"),
            weights = NULL, srange = NULL, ns = 16, sigma = NULL,
            leaveoneout = TRUE, verbose = TRUE)
```

Arguments

`X`	A multitype point pattern (object of class `"ppp"`).
`...`	Arguments passed to `relriskHeat.ppp`.
`method`	Character string specifying the cross-validation method. Partially matched to `"likelihood"` for binary likelihood cross-validation or `"leastsquares"` for least squares cross-validation.
`weights`	Optional numeric vector of weights associated with each point of `X`.
`srange`	Numeric vector of length 2 specifying a range of bandwidths to be considered.
`ns`	Integer. Number of candidate bandwidths to be considered.
`sigma`	Maximum smoothing bandwidth. A numeric value, or a pixel image, or a `function(x,y)`. Alternatively a numeric vector containing a sequence of candidate bandwidths.
`leaveoneout`	Logical value specifying whether intensity values at data points should be estimated using the leave-one-out rule.
`verbose`	Logical value specifying whether to print progress reports.

Details

This algorithm selects the optimal global bandwidth for kernel estimation of relative risk for the dataset `X` using diffusion smoothing `relriskHeat`.

If `sigma` is a numeric value, the algorithm finds the optimal bandwidth `tau <= sigma`.

If `sigma` is a pixel image or function, the algorithm finds the optimal fraction `0 < f <= 1` such that smoothing with `f * sigma` would be optimal.

Value

A numerical value giving the selected bandwidth (if `sigma` was a numeric value) or the selected fraction of the maximum bandwidth (if `sigma` was a pixel image or function). The result also belongs to the class `"bw.optim"` which can be plotted.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Tilman Davies <Tilman.Davies@otago.ac.nz> and Suman Rakshit.

See Also

`relriskHeat.ppp`

Examples

```
## bovine tuberculosis data
X <- subset(btb, select=spoligotype)
if(interactive()) {
  smax <- 40
  ns <- 16
  dimyx <- NULL
} else {
  ## reduce data and resolution to speed up
  X <- X[c(TRUE, rep(FALSE, 7))]
  smax <- 9
  ns <- 8
  dimyx <- 32
}
b <- bw.relriskHeatppp(X, sigma=smax, ns=ns, dimyx=dimyx)
b
plot(b)
```

`bw.scott` *Scott's Rule for Bandwidth Selection for Kernel Density*

Description

Use Scott's rule of thumb to determine the smoothing bandwidth for the kernel estimation of point process intensity.

Usage

```
bw.scott(X, isotropic=FALSE, d=NULL)

bw.scott.iso(X)
```

Arguments

X	A point pattern (object of class "ppp", "lpp", "pp3" or "ppx").
isotropic	Logical value indicating whether to compute a single bandwidth for an isotropic Gaussian kernel (isotropic=TRUE) or separate bandwidths for each coordinate axis (isotropic=FALSE, the default).
d	Advanced use only. An integer value that should be used in Scott's formula instead of the true number of spatial dimensions.

Details

These functions select a bandwidth sigma for the kernel estimator of point process intensity computed by density.ppp or other appropriate functions. They can be applied to a point pattern belonging to any class "ppp", "lpp", "pp3" or "ppx".

The bandwidth σ is computed by the rule of thumb of Scott (1992, page 152, equation 6.42). The bandwidth is proportional to $n^{-1/(d+4)}$ where n is the number of points and d is the number of spatial dimensions.

This rule is very fast to compute. It typically produces a larger bandwidth than bw.diggle. It is useful for estimating gradual trend.

If isotropic=FALSE (the default), bw.scott provides a separate bandwidth for each coordinate axis, and the result of the function is a vector, of length equal to the number of coordinates. If isotropic=TRUE, a single bandwidth value is computed and the result is a single numeric value.

bw.scott.iso(X) is equivalent to bw.scott(X, isotropic=TRUE).

The default value of d is as follows:

class	dimension
"ppp"	2
"lpp"	1
"pp3"	3
"ppx"	number of spatial coordinates

The use of d=1 for point patterns on a linear network (class "lpp") was proposed by McSwiggan et al (2016) and Rakshit et al (2019).

Value

A numerical value giving the selected bandwidth, or a numerical vector giving the selected bandwidths for each coordinate.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

Scott, D.W. (1992) *Multivariate Density Estimation. Theory, Practice and Visualization*. New York: Wiley.

See Also

density.ppp, bw.diggle, bw.ppl, bw.CvL, bw.frac.

Examples

```
hickory <- split(lansing)[["hickory"]]
b <- bw.scott(hickory)
b
if(interactive()) {
 plot(density(hickory, b))
}
bw.scott.iso(hickory)
bw.scott(osteo$pts[[1]])
```

---

bw.smoothppp *Cross Validated Bandwidth Selection for Spatial Smoothing*

Description

Uses least-squares cross-validation to select a smoothing bandwidth for spatial smoothing of marks.

Usage

```
bw.smoothppp(X, ..., nh = spatstat.options("n.bandwidth"),
             hmin=NULL, hmax=NULL, warn=TRUE, kernel="gaussian",
             varcov1=NULL, train=NULL, test=NULL)
```

Arguments

X	A marked point pattern with numeric marks.
...	Ignored.
nh	Number of trial values of smoothing bandwith sigma to consider. The default is 32.
hmin, hmax	Optional. Numeric values. Range of trial values of smoothing bandwith sigma to consider. There is a sensible default.
warn	Logical. If TRUE, issue a warning if the minimum of the cross-validation criterion occurs at one of the ends of the search interval.
kernel	The smoothing kernel. A character string specifying the smoothing kernel (current options are "gaussian", "epanechnikov", "quartic" or "disc").
varcov1	Optional. Variance-covariance matrix matrix of the kernel with bandwidth $h = 1$. See section on Anisotropic Smoothing.
train, test	Optional. Training and testing subsets for cross-validation. Each argument is either a valid subset index in the usual R sense, or a window (an object of class "owin") delineating a subset of the original observation window, or a pixel image with logical values defining a subset of the original observation window.

Details

This function selects an appropriate bandwidth for the nonparametric smoothing of mark values using Smooth.ppp.

The argument X must be a marked point pattern with a vector or data frame of marks. All mark values must be numeric.

The bandwidth is selected by least-squares cross-validation. Let y_i be the mark value at the ith data point. For a particular choice of smoothing bandwidth, let $\hat{y}_i$ be the smoothed value at the ith data point. Then the bandwidth is chosen to minimise the squared error of the smoothed values $\sum_i (y_i - \hat{y}_i)^2$.

If the argument train is given, then spatial smoothing is calculated using only the data from the subset X[train]. If the argument test is given, then smoothed values are calculated only at the locations in the subset X[test], and squared errors are summed over the locations in X[test].

(If test is given, then this sum of squared errors is computed only over the specified subset of locations.)

The result of bw.smoothppp is a numerical value giving the selected bandwidth sigma. The result also belongs to the class "bw.optim" allowing it to be printed and plotted. The plot shows the cross-validation criterion as a function of bandwidth.

The range of values for the smoothing bandwidth sigma is set by the arguments hmin, hmax. There is a sensible default, based on the nearest neighbour distances.

If the optimal bandwidth is achieved at an endpoint of the interval [hmin, hmax], the algorithm will issue a warning (unless warn=FALSE). If this occurs, then it is probably advisable to expand the interval by changing the arguments hmin, hmax.

Computation time depends on the number nh of trial values considered, and also on the range [hmin, hmax] of values considered, because larger values of sigma require calculations involving more pairs of data points.

Value

A single numerical value giving the selected bandwidth. The result also belongs to the class "bw.optim" (see bw.optim.object) which can be plotted to show the bandwidth selection criterion as a function of sigma.

Anisotropic Smoothing

Anisotropic smoothing is available in Smooth.ppp using the argument varcov to specify the variance-covariance matrix of the anisotropic kernel. In order to choose the matrix varcov, the user can call bw.smoothppp using the argument varcov1 to specify a 'template' matrix. Scalar multiples of varcov1 will be considered and the optimal scale factor will be determined. That is, bw.smoothppp will try smoothing the data using varcov = h^2 * varcov1 for different values of h ranging from hmin to hmax. The result of bw.smoothppp will be the optimal value of the standard deviation scale factor h.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

Smooth.ppp, bw.optim.object

Examples

```
b <- bw.smoothppp(longleaf)
b
plot(b)
```

---

bw.stoyan *Stoyan's Rule of Thumb for Bandwidth for Estimating Pair Correlation*

Description

Computes a rough estimate of the appropriate bandwidth for kernel smoothing estimators of the pair correlation function.

Usage

```
bw.stoyan(X, co=0.15, extrapolate=FALSE, ...)
```

Arguments

X	A point pattern (object of class "ppp").
co	Coefficient appearing in the rule of thumb. See Details.
extrapolate	Logical value specifying whether to use the extrapolated version of the rule. See Details.
...	Ignored.

Details

Estimation of the pair correlation function (and similar quantities) by smoothing methods requires a choice of the smoothing bandwidth. Stoyan and Stoyan (1995, equation (15.16), page 285) proposed a rule of thumb for choosing the smoothing bandwidth.

For the Epanechnikov kernel, the rule of thumb is to set the kernel's half-width h to $0.15/\sqrt{\lambda}$ where λ is the estimated intensity of the point pattern, typically computed as the number of points of X divided by the area of the window containing X.

For a general kernel, the corresponding rule is to set the standard deviation of the kernel to $\sigma = 0.15/\sqrt{5\lambda}$.

The coefficient 0.15 can be tweaked using the argument co.

To ensure the bandwidth is finite, an empty point pattern is treated as if it contained 1 point.

The original version of Stoyan's rule, stated above, was developed by experience with patterns of 30 to 100 points. For patterns with larger numbers of points, the bandwidth should be smaller: the theoretically optimal bandwidth decreases in proportion to $n^{-1/5}$ where n is the number of points

in the pattern. In the 'extrapolated' version of Stoyan's rule proposed by Baddeley, Davies and Hazelton (2025), the value σ calculated above is multiplied by $(100/n)^{1/5}$. The extrapolated rule is applied if `extrapolate=TRUE`.

Value

A finite positive numerical value giving the selected bandwidth (the standard deviation of the smoothing kernel).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net>, Tilman Davies <Tilman.Davies@otago.ac.nz>, Martin Hazelton <Martin.Hazelton@otago.ac.nz> and Ya-Mei Chang <yamei628@gmail.com>.

References

Baddeley, A., Davies, T.M. and Hazelton, M.L. (2025) An improved estimator of the pair correlation function of a spatial point process. *Biometrika*, to appear.

Stoyan, D. and Stoyan, H. (1995) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

See Also

`pcf`, `bw.relrisk`

Examples

```
bw.stoyan(shapley)
bw.stoyan(shapley, extrapolate=TRUE)
```

---

cdf.test *Spatial Distribution Test for Point Pattern or Point Process Model*

Description

Performs a test of goodness-of-fit of a point process model. The observed and predicted distributions of the values of a spatial covariate are compared using either the Kolmogorov-Smirnov test, Cramér-von Mises test or Anderson-Darling test. For non-Poisson models, a Monte Carlo test is used.

Usage

```
cdf.test(...)

## S3 method for class 'ppp'
cdf.test(X, covariate, test=c("ks", "cvm", "ad"), ...,
                       interpolate=TRUE, jitter=TRUE)
```

Arguments

X	A point pattern (object of class "ppp" or "lpp").
covariate	The spatial covariate on which the test will be based. A function, a pixel image (object of class "im"), a list of pixel images, or one of the characters "x" or "y" indicating the Cartesian coordinates.
test	Character string identifying the test to be performed: "ks" for Kolmogorov-Smirnov test, "cvm" for Cramér-von Mises test or "ad" for Anderson-Darling test.
...	Arguments passed to ks.test (from the **stats** package) or cvm.test or ad.test (from the **goftest** package) to control the test; and arguments passed to as.mask to control the pixel resolution.
interpolate	Logical flag indicating whether to interpolate pixel images. If interpolate=TRUE, the value of the covariate at each point of X will be approximated by interpolating the nearby pixel values. If interpolate=FALSE, the nearest pixel value will be used.
jitter	Logical flag. If jitter=TRUE, values of the covariate will be slightly perturbed at random, to avoid tied values in the test.

Details

These functions perform a goodness-of-fit test of a Poisson or Gibbs point process model fitted to point pattern data. The observed distribution of the values of a spatial covariate at the data points, and the predicted distribution of the same values under the model, are compared using the Kolmogorov-Smirnov test, the Cramér-von Mises test or the Anderson-Darling test. For Gibbs models, a Monte Carlo test is performed using these test statistics.

The function cdf.test is generic, with methods for point patterns ("ppp" or "lpp"), point process models ("ppm" or "lppm") and spatial logistic regression models ("slrm").

- If X is a point pattern dataset (object of class "ppp"), then cdf.test(X, ...) performs a goodness-of-fit test of the uniform Poisson point process (Complete Spatial Randomness, CSR) for this dataset. For a multitype point pattern, the uniform intensity is assumed to depend on the type of point (sometimes called Complete Spatial Randomness and Independence, CSRI).
- If model is a fitted point process model (object of class "ppm" or "lppm") then cdf.test(model, ...) performs a test of goodness-of-fit for this fitted model.

- If model is a fitted spatial logistic regression (object of class "slrm") then cdf.test(model, ...) performs a test of goodness-of-fit for this fitted model.

The test is performed by comparing the observed distribution of the values of a spatial covariate at the data points, and the predicted distribution of the same covariate under the model, using a classical goodness-of-fit test. Thus, you must nominate a spatial covariate for this test.

If X is a point pattern that does not have marks, the argument covariate should be either a function(x,y) or a pixel image (object of class "im" containing the values of a spatial function, or one of the characters "x" or "y" indicating the Cartesian coordinates. If covariate is an image, it should have numeric values, and its domain should cover the observation window of the model. If covariate is a function, it should expect two arguments x and y which are vectors of coordinates, and it should return a numeric vector of the same length as x and y.

If X is a multitype point pattern, the argument covariate can be either a function(x,y,marks), or a pixel image, or a list of pixel images corresponding to each possible mark value, or one of the characters "x" or "y" indicating the Cartesian coordinates.

First the original data point pattern is extracted from model. The values of the covariate at these data points are collected.

The predicted distribution of the values of the covariate under the fitted model is computed as follows. The values of the covariate at all locations in the observation window are evaluated, weighted according to the point process intensity of the fitted model, and compiled into a cumulative distribution function F using ewcdf.

The probability integral transformation is then applied: the values of the covariate at the original data points are transformed by the predicted cumulative distribution function F into numbers between 0 and 1. If the model is correct, these numbers are i.i.d. uniform random numbers. The A goodness-of-fit test of the uniform distribution is applied to these numbers using stats::ks.test, goftest::cvm.test or goftest::ad.test.

This test was apparently first described (in the context of spatial data, and using Kolmogorov-Smirnov) by Berman (1986). See also Baddeley et al (2005).

If model is not a Poisson process, then a Monte Carlo test is performed, by generating nsim point patterns which are simulated realisations of the model, re-fitting the model to each simulated point pattern, and calculating the test statistic for each fitted model. The Monte Carlo p value is determined by comparing the simulated values of the test statistic with the value for the original data.

The return value is an object of class "htest" containing the results of the hypothesis test. The print method for this class gives an informative summary of the test outcome.

The return value also belongs to the class "cdftest" for which there is a plot method plot.cdftest. The plot method displays the empirical cumulative distribution function of the covariate at the data points, and the predicted cumulative distribution function of the covariate under the model, plotted against the value of the covariate.

The argument jitter controls whether covariate values are randomly perturbed, in order to avoid ties. If the original data contains any ties in the covariate (i.e. points with equal values of the covariate), and if jitter=FALSE, then the Kolmogorov-Smirnov test implemented in ks.test will issue a warning that it cannot calculate the exact p-value. To avoid this, if jitter=TRUE each value of the covariate will be perturbed by adding a small random value. The perturbations are normally distributed with standard deviation equal to one hundredth of the range of values of the covariate. This prevents ties, and the p-value is still correct. There is a very slight loss of power.

Value

An object of class "htest" containing the results of the test. See ks.test for details. The return value can be printed to give an informative summary of the test.

The value also belongs to the class "cdftest" for which there is a plot method.

Warning

The outcome of the test involves a small amount of random variability, because (by default) the coordinates are randomly perturbed to avoid tied values. Hence, if cdf.test is executed twice, the p-values will not be exactly the same. To avoid this behaviour, set jitter=FALSE.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

References

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.

Berman, M. (1986) Testing for spatial association between a point process and another stochastic process. *Applied Statistics* **35**, 54–62.

See Also

plot.cdftest, quadrat.test, berman.test, ks.test, cvm.test, ad.test, ppm

Examples

```
op <- options(useFancyQuotes=FALSE)

# test of CSR using x coordinate
cdf.test(nztrees, "x")
cdf.test(nztrees, "x", "cvm")
cdf.test(nztrees, "x", "ad")

# test of CSR using a function of x and y
fun <- function(x,y){2* x + y}
cdf.test(nztrees, fun)

# test of CSR using an image covariate
funimage <- as.im(fun, W=Window(nztrees))
cdf.test(nztrees, funimage)


# multitype point pattern
cdf.test(amacrine, "x")

options(op)
```

circdensity *Density Estimation for Circular Data*

Description

Computes a kernel smoothed estimate of the probability density for angular data.

Usage

```
circdensity(x, sigma = "nrd0", ...,
               bw = NULL,
               weights=NULL, unit = c("degree", "radian"))
```

Arguments

x	Numeric vector, containing angular data.
sigma	Smoothing bandwidth, or bandwidth selection rule, passed to density.default.
bw	Alternative to sigma for consistency with other functions.
...	Additional arguments passed to density.default, such as kernel and weights.
weights	Optional numeric vector of weights for the data in x.
unit	The unit of angle in which x is expressed.

Details

The angular values x are smoothed using (by default) the wrapped Gaussian kernel with standard deviation sigma.

Value

An object of class "density" (produced by density.default) which can be plotted by plot or by rose.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

density.default), rose.

Examples

```
ang <- runif(1000, max=360)
rose(circdensity(ang, 12))
```

clarkevans *Clark and Evans Aggregation Index*

Description

Computes the Clark and Evans aggregation index R for a spatial point pattern.

Usage

```
clarkevans(X, correction=c("none", "Donnelly", "cdf"),
              clipregion=NULL)
```

Arguments

X	A spatial point pattern (object of class "ppp").
correction	Character vector. The type of edge correction(s) to be applied.
clipregion	Clipping region for the guard area correction. A window (object of class "owin"). See Details.

Details

The Clark and Evans (1954) aggregation index R is a crude measure of clustering or ordering of a point pattern. It is the ratio of the observed mean nearest neighbour distance in the pattern to that expected for a Poisson point process of the same intensity. A value $R > 1$ suggests ordering, while $R < 1$ suggests clustering.

Without correction for edge effects, the value of R will be positively biased. Edge effects arise because, for a point of X close to the edge of the window, the true nearest neighbour may actually lie outside the window. Hence observed nearest neighbour distances tend to be larger than the true nearest neighbour distances.

The argument correction specifies an edge correction or several edge corrections to be applied. It is a character vector containing one or more of the options "none", "Donnelly", "guard" and "cdf" (which are recognised by partial matching). These edge corrections are:

"none": No edge correction is applied.

"Donnelly": Edge correction of Donnelly (1978), available for rectangular windows only. The theoretical expected value of mean nearest neighbour distance under a Poisson process is adjusted for edge effects by the edge correction of Donnelly (1978). The value of R is the ratio of the observed mean nearest neighbour distance to this adjusted theoretical mean.

"guard": Guard region or buffer area method. The observed mean nearest neighbour distance for the point pattern X is re-defined by averaging only over those points of X that fall inside the sub-window clipregion.

"cdf": Cumulative Distribution Function method. The nearest neighbour distance distribution function $G(r)$ of the stationary point process is estimated by Gest using the Kaplan-Meier type edge correction. Then the mean of the distribution is calculated from the cdf.

Alternatively `correction="all"` selects all options.

If the argument `clipregion` is given, then the selected edge corrections will be assumed to include `correction="guard"`.

To perform a test based on the Clark-Evans index, see `clarkevans.test`.

Value

A numeric value, or a numeric vector with named components

`naive`	R without edge correction
`Donnelly`	R using Donnelly edge correction
`guard`	R using guard region
`cdf`	R using cdf method

(as selected by `correction`). The value of the `Donnelly` component will be `NA` if the window of `X` is not a rectangle.

Author(s)

John Rudge `<rudge@esc.cam.ac.uk>` with modifications by Adrian Baddeley `<Adrian.Baddeley@curtin.edu.au>`

References

Clark, P.J. and Evans, F.C. (1954) Distance to nearest neighbour as a measure of spatial relationships in populations *Ecology* **35**, 445–453.

Donnelly, K. (1978) Simulations to determine the variance and edge-effect of total nearest neighbour distance. In I. Hodder (ed.) *Simulation studies in archaeology*, Cambridge/New York: Cambridge University Press, pp 91–95.

See Also

`clarkevans.test`, `hopskel`, `nndist`, `Gest`

Examples

```
# Example of a clustered pattern
clarkevans(redwood)

# Example of an ordered pattern
clarkevans(cells)

# Random pattern
X <- rpoispp(100)
clarkevans(X)

# How to specify a clipping region
clip1 <- owin(c(0.1,0.9),c(0.1,0.9))
clip2 <- erosion(Window(cells), 0.1)
clarkevans(cells, clipregion=clip1)
clarkevans(cells, clipregion=clip2)
```

clarkevans.test *Clark and Evans Test*

Description

Performs the Clark-Evans test of aggregation for a spatial point pattern.

Usage

```
clarkevans.test(X, ...,
                correction,
                clipregion=NULL,
                alternative=c("two.sided", "less", "greater",
                              "clustered", "regular"),
                method=c("asymptotic", "MonteCarlo"),
                nsim=999)
```

Arguments

X	A spatial point pattern (object of class "ppp").
...	Ignored.
correction	Character string. The type of edge correction to be applied. See clarkevans and Details below.
clipregion	Clipping region for the guard area correction. A window (object of class "owin"). See clarkevans
alternative	String indicating the type of alternative for the hypothesis test. Partially matched.
method	Character string (partially matched) specifying how to calculate the p-value of the test. See Details.
nsim	Number of Monte Carlo simulations to perform, if a Monte Carlo p-value is required.

Details

This command uses the Clark and Evans (1954) aggregation index R as the basis for a crude test of clustering or ordering of a point pattern.

The Clark-Evans *aggregation index* R is computed by the separate function clarkevans.

This command clarkevans.text performs a hypothesis test of clustering or ordering of the point pattern X based on the Clark-Evans index R. The null hypothesis is Complete Spatial Randomness, i.e.\ a uniform Poisson process. The alternative hypothesis is specified by the argument alternative:

- alternative="less" or alternative="clustered": the alternative hypothesis is that $R <$ 1 corresponding to a clustered point pattern;
- alternative="greater" or alternative="regular": the alternative hypothesis is that $R >$ 1 corresponding to a regular or ordered point pattern;

- `alternative="two.sided"`: the alternative hypothesis is that $R \neq 1$ corresponding to a clustered or regular pattern.

The Clark-Evans index R is first computed for the point pattern dataset `X` using the edge correction determined by the arguments `correction` and `clipregion`. These arguments are documented in the help file for `clarkevans`.

If `method="asymptotic"` (the default), the p-value for the test is computed by standardising R as proposed by Clark and Evans (1954) and referring the standardised statistic to the standard Normal distribution. For this asymptotic test, the default edge correction is `correction="Donnelly"` if the window of `X` is a rectangle, and `correction="cdf"` otherwise. It is strongly recommended to avoid using `correction="none"` which would lead to a severely biased test.

If `method="MonteCarlo"`, the p-value for the test is computed by comparing the observed value of R to the results obtained from `nsim` simulated realisations of Complete Spatial Randomness conditional on the observed number of points. This test is theoretically exact for any choice of edge correction, but may have lower power than the asymptotic test. For this Monte Carlo test, the default edge correction is `correction="none"` for computational efficiency.

Value

An object of class `"htest"` representing the result of the test.

Author(s)

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

References

Clark, P.J. and Evans, F.C. (1954) Distance to nearest neighbour as a measure of spatial relationships in populations. *Ecology* **35**, 445–453.

Donnelly, K. (1978) Simulations to determine the variance and edge-effect of total nearest neighbour distance. In *Simulation methods in archaeology*, Cambridge University Press, pp 91–95.

See Also

`clarkevans`, `hopskel.test`

Examples

```
# Redwood data - clustered
clarkevans.test(redwood)
clarkevans.test(redwood, alternative="clustered")
clarkevans.test(redwood, correction="cdf", method="MonteCarlo", nsim=39)
```

clusterset *Allard-Fraley Estimator of Cluster Feature*

Description

Detect high-density features in a spatial point pattern using the (unrestricted) Allard-Fraley estimator.

Usage

```
clusterset(X, what=c("marks", "domain"),
           ..., verbose=TRUE,
           fast=FALSE,
           exact=!fast)
```

Arguments

X	A dimensional spatial point pattern (object of class "ppp").
what	Character string or character vector specifying the type of result. See Details.
verbose	Logical value indicating whether to print progress reports.
fast	Logical. If FALSE (the default), the Dirichlet tile areas will be computed exactly using polygonal geometry, so that the optimal choice of tiles will be computed exactly. If TRUE, the Dirichlet tile areas will be approximated using pixel counting, so the optimal choice will be approximate.
exact	Logical. If TRUE, the Allard-Fraley estimator of the domain will be computed exactly using polygonal geometry. If FALSE, the Allard-Fraley estimator of the domain will be approximated by a binary pixel mask. The default is initially set to FALSE.
...	Optional arguments passed to as.mask to control the pixel resolution if exact=FALSE.

Details

Allard and Fraley (1997) developed a technique for recognising features of high density in a spatial point pattern in the presence of random clutter.

This algorithm computes the *unrestricted* Allard-Fraley estimator. The Dirichlet (Voronoi) tessellation of the point pattern X is computed. The smallest m Dirichlet cells are selected, where the number m is determined by a maximum likelihood criterion.

- If fast=FALSE (the default), the areas of the tiles of the Dirichlet tessellation will be computed exactly using polygonal geometry. This ensures that the optimal selection of tiles is computed exactly.
- If fast=TRUE, the Dirichlet tile areas will be approximated by counting pixels. This is faster, and is usually correct (depending on the pixel resolution, which is controlled by the arguments ...).

The type of result depends on the character vector what.

- If what="marks" the result is the point pattern X with a vector of marks labelling each point with a value yes or no depending on whether the corresponding Dirichlet cell is selected by the Allard-Fraley estimator. In other words each point of X is labelled as either a cluster point or a non-cluster point.
- If what="domain", the result is the Allard-Fraley estimator of the cluster feature set, which is the union of all the selected Dirichlet cells, represented as a window (object of class "owin").
- If what=c("marks", "domain") the result is a list containing both of the results described above.

Computation of the Allard-Fraley set estimator depends on the argument exact.

- If exact=TRUE (the default), the Allard-Fraley set estimator will be computed exactly using polygonal geometry. The result is a polygonal window.
- If exact=FALSE, the Allard-Fraley set estimator will be approximated by a binary pixel mask. This is faster than the exact computation. The result is a binary mask.

Value

If what="marks", a multitype point pattern (object of class "ppp").

If what="domain", a window (object of class "owin").

If what=c("marks", "domain") (the default), a list consisting of a multitype point pattern and a window.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

References

Allard, D. and Fraley, C. (1997) Nonparametric maximum likelihood estimation of features in spatial point processes using Voronoi tessellation. *Journal of the American Statistical Association* **92**, 1485–1493.

See Also

nnclean, sharpen

Examples

```
opa <- par(mfrow=c(1,2))
W <- grow.rectangle(as.rectangle(letterR), 1)
X <- superimpose(runifpoint(300, letterR),
                 runifpoint(50, W), W=W)
plot(W, main="clusterset(X, 'm')")
plot(clusterset(X, "marks", fast=TRUE), add=TRUE, chars=c(1, 3), cols=1:2)
plot(letterR, add=TRUE)
plot(W, main="clusterset(X, 'd')")
plot(clusterset(X, "domain", exact=FALSE), add=TRUE)
```

```
plot(letterR, add=TRUE)
par(opa)
```

`collapse.fv` *Collapse Several Function Tables into One*

Description

Combines several function tables (objects of class `"fv"`) into a single function table, merging columns that are identical and relabelling columns that are different.

Usage

```
## S3 method for class 'fv'
collapse(object, ..., same = NULL, different = NULL)

## S3 method for class 'anylist'
collapse(object, ..., same = NULL, different = NULL)
```

Arguments

`object`	An object of class `"fv"`, or a list of such objects.
`...`	Additional objects of class `"fv"`.
`same`	Character string or character vector specifying a column or columns of function values that are identical in different `"fv"` objects. These columns will be included only once in the result.
`different`	Character string or character vector specifying a column or columns of function values, that are different in different `"fv"` objects. Each of these columns of data will be included, with labels that distinguish them from each other.

Details

This is a method for the generic function `collapse`.

It combines the data in several function tables (objects of class `"fv"`, see `fv.object`) to make a single function table. It is essentially a smart wrapper for `cbind.fv`.

A typical application is to calculate the same summary statistic (such as the K function) for different point patterns, and then to use `collapse.fv` to combine the results into a single object that can easily be plotted. See the Examples.

The arguments `object` and `...` should be function tables (objects of class `"fv"`, see `fv.object`) that are compatible in the sense that they have the same values of the function argument. (This can be ensured by applying `harmonise.fv` to them.)

The argument `same` identifies any columns that are present in some or all of the function tables, and which are known to contain exactly the same values in each table that includes them. This column or columns will be included only once in the result.

The argument different identifies any columns that are present in some or all of the function tables, and which may contain different numerical values in different tables. Each of these columns will be included, with labels to distinguish them.

Columns that are not named in same or different will not be included.

The function argument is always included and does not need to be specified.

The arguments same and different can be NULL, or they can be character vectors containing the names of columns of object. The argument different can be one of the abbreviations recognised by fvnames.

Value

Object of class "fv".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

fv.object, cbind.fv

Examples

```
# generate simulated data
X <- replicate(3, rpoispp(100), simplify=FALSE)
names(X) <- paste("Simulation", 1:3)
# compute K function estimates
Klist <- anylapply(X, Kest)
# collapse
K <- collapse(Klist, same="theo", different="iso")
K
```

---

compatible.fasp *Test Whether Function Arrays Are Compatible*

Description

Tests whether two or more function arrays (class "fasp") are compatible.

Usage

```
  ## S3 method for class 'fasp'
compatible(A, B, ...)
```

Arguments

A, B, ...	Two or more function arrays (object of class "fasp").

Details

An object of class "fasp" can be regarded as an array of functions. Such objects are returned by the command alltypes.

This command tests whether such objects are compatible (so that, for example, they could be added or subtracted). It is a method for the generic command compatible.

The function arrays are compatible if the arrays have the same dimensions, and the corresponding elements in each cell of the array are compatible as defined by compatible.fv.

Value

Logical value: TRUE if the objects are compatible, and FALSE if they are not.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

eval.fasp

---

compatible.fv *Test Whether Function Objects Are Compatible*

Description

Tests whether two or more function objects (class "fv") are compatible.

Usage

```
  ## S3 method for class 'fv'
compatible(A, B, ..., samenames=TRUE)
```

Arguments

A, B, ...	Two or more function value objects (class "fv").
samenames	Logical value indicating whether to check for complete agreement between the column names of the objects (samenames=TRUE, the default) or just to check that the name of the function argument is the same (samenames=FALSE).

Details

An object of class "fv" is essentially a data frame containing several different statistical estimates of the same function. Such objects are returned by Kest and its relatives.

This command tests whether such objects are compatible (so that, for example, they could be added or subtracted). It is a method for the generic command compatible.

The functions are compatible if they have been evaluated at the same sequence of values of the argument r, and if the statistical estimates have the same names.

Value

Logical value: TRUE if the objects are compatible, and FALSE if they are not.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

eval.fv

---

compileCDF *Generic Calculation of Cumulative Distribution Function of Distances*

Description

A low-level function which calculates the estimated cumulative distribution function of a distance variable.

Usage

```
compileCDF(D, B, r, ..., han.denom=NULL, check=TRUE)
```

Arguments

D	A vector giving the distances from each data point to the target.
B	A vector giving the distances from each data point to the window boundary, or censoring distances.
r	An equally spaced, finely spaced sequence of distance values at which the CDF should be estimated.
...	Ignored.
han.denom	Denominator for the Hanisch-Chiu-Stoyan estimator. A single number, or a numeric vector with the same length as r.
check	Logical value specifying whether to check validity of the data, for example, that the vectors D and B have the same length, and contain non-negative numbers.

Details

This low-level function calculates estimates of the cumulative distribution function

$$F(r) = P(D \le r)$$

of a distance variable D, given a vector of observed values of D and other information. Examples of this concept include the empty space distance function computed by Fest and the nearest-neighbour distance distribution function Gest.

This function compileCDF and its siblings compileK and compilepcf are useful for code development and for teaching, because they perform a common task, and do the housekeeping required to make an object of class "fv" that represents the estimated function. However, they are not very efficient.

The argument D should be a numeric vector of shortest distances measured from each 'query' point to the 'target' set. The argument B should be a numeric vector of shortest distances measured from each 'query' point to the boundary of the window of observation. All entries of D and B should be non-negative.

compileCDF calculates estimates of the cumulative distribution function $F(r)$ using the border method (reduced sample estimator), the Kaplan-Meier estimator and, if han.denom is given, the Hanisch-Chiu-Stoyan estimator. See Chapter 8 of Baddeley, Rubak and Turner (2015).

The result is an object of class "fv" representing the estimated function. Additional columns (such as a column giving the theoretical value) must be added by the user, with the aid of bind.fv.

Value

An object of class "fv" representing the estimated function.

Author(s)

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

References

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. Chapman and Hall/CRC Press.

See Also

compileK.

bind.fv to add more columns.

Examples

```
## Equivalent to Gest(japanesepines)
X <- japanesepines
D <- nndist(X)
B <- bdist.points(X)
r <- seq(0, 0.25, by=0.01)
H <- eroded.areas(Window(X), r)
G <- compileCDF(D=D, B=B, r=r, han.denom=H)
G <- rebadge.fv(G, new.fname="G", new.ylab=quote(G(r)))
plot(G)
```

compileK *Generic Calculation of K Function and Pair Correlation Function*

Description

Low-level functions which calculate the estimated K function and estimated pair correlation function (or any similar functions) from a matrix of pairwise distances and optional weights.

Usage

```
compileK(D, r, weights = NULL, denom = 1,
         check = TRUE, ratio = FALSE, fname = "K",
         samplesize=denom)

compilepcf(D, r, weights = NULL, denom = 1,
         check = TRUE, endcorrect = TRUE, ratio=FALSE,
 ..., fname = "g", samplesize=denom)
```

Arguments

D	A square matrix giving the distances between all pairs of points.
r	An equally spaced, finely spaced sequence of distance values.
weights	Optional numerical weights for the pairwise distances. A numeric matrix with the same dimensions as D. If absent, the weights are taken to equal 1.
denom	Denominator for the estimator. A single number, or a numeric vector with the same length as r. See Details.
check	Logical value specifying whether to check that D is a valid matrix of pairwise distances.
ratio	Logical value indicating whether to store ratio information. See Details.
...	Optional arguments passed to density.default controlling the kernel smoothing.
endcorrect	Logical value indicating whether to apply End Correction of the pair correlation estimate at r=0.
fname	Character string giving the name of the function being estimated.
samplesize	The sample size that should be used as the denominator when ratio=TRUE.

Details

These low-level functions construct estimates of the K function or pair correlation function, or any similar functions, given only the matrix of pairwise distances and optional weights associated with these distances.

These functions are useful for code development and for teaching, because they perform a common task, and do the housekeeping required to make an object of class "fv" that represents the estimated function. However, they are not very efficient.

compileK calculates the weighted estimate of the K function,

$$\hat{K}(r) = (1/v(r)) \sum_i \sum_j 1\{d_{ij} \le r\} w_{ij}$$

and compilepcf calculates the weighted estimate of the pair correlation function,

$$\hat{g}(r) = (1/v(r)) \sum_i \sum_j \kappa(d_{ij} - r) w_{ij}$$

where d_{ij} is the distance between spatial points i and j, with corresponding weight w_{ij}, and $v(r)$ is a specified denominator. Here κ is a fixed-bandwidth smoothing kernel.

For a point pattern in two dimensions, the usual denominator $v(r)$ is constant for the K function, and proportional to r for the pair correlation function. See the Examples.

The result is an object of class "fv" representing the estimated function. This object has only one column of function values. Additional columns (such as a column giving the theoretical value) must be added by the user, with the aid of bind.fv.

If ratio=TRUE, the result also belongs to class "rat" and has attributes containing the numerator and denominator of the function estimate. (If samplesize is given, the numerator and denominator are rescaled by a common factor so that the denominator is equal to samplesize.) This allows function estimates from several datasets to be pooled using pool.

Value

An object of class "fv" representing the estimated function.

Author(s)

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

See Also

Kest, pcf for definitions of the K function and pair correlation function.

bind.fv to add more columns.

compileCDF for the corresponding low-level utility for estimating a cumulative distribution function.

Examples

```
## Equivalent to Kest(japanesepines) and pcf(japanesepines)
X <- japanesepines
D <- pairdist(X)
Wt <- edge.Ripley(X, D)
lambda <- intensity(X)
a <- (npoints(X)-1) * lambda
r <- seq(0, 0.25, by=0.01)
K <- compileK(D=D, r=r, weights=Wt, denom=a)
g <- compilepcf(D=D, r=r, weights=Wt, denom= a * 2 * pi * r)
```

cov.im *Covariance and Correlation between Images*

Description

Compute the covariance or correlation between (the corresponding pixel values in) several images.

Usage

```
cov.im(..., use = "complete.obs", method = c("pearson", "kendall", "spearman"))
cor.im(..., use = "complete.obs", method = c("pearson", "kendall", "spearman"))
```

Arguments

...	Any number of arguments, each of which is a pixel image (object of class "im"). Alternatively, a single argument which is a list of pixel images.
use	Argument passed to cov or cor determining how to handle NA values in the data.
method	Argument passed to cov or cor determining the type of correlation that will be computed.

Details

The arguments ... should be pixel images (objects of class "im"). Their spatial domains must overlap, but need not have the same pixel dimensions.

These functions compute the covariance or correlation between the corresponding pixel values in the images given.

The pixel images are converted to a common pixel resolution (by resampling). Then the corresponding pixel values of each image are extracted. Finally the correlation or covariance between the pixel values of each pair of images, at corresponding pixels, is computed.

The result is a symmetric matrix with one row and column for each image. The [i,j] entry is the correlation or covariance between the ith and jth images in the argument list. The row names and column names of the matrix are copied from the argument names if they were given (i.e. if the arguments were given as name=value).

The argument use specifies how to handle NA values. A pixel value of NA is assigned to any pixel falling outside the spatial domain of an image (i.e. outside the window in which the image is defined). If any one of the image arguments ... is defined on a non-rectangular window, or if the image arguments are not all defined on the same window, then the data will contain NA values. Options for the argument use are documented in the help file for cov and cor.

- use="complete.obs" (the default): calculations are based on those pixels which lie inside the intersection of the windows of all the images. An error occurs if the intersection is empty.
- use="na.or.complete": calculations are based on those pixels which lie inside the intersection of the windows of all the images. If the intersection is empty, a matrix of NA values is returned.

- use="pairwise.complete.obs": the calculation of the covariance or correlation between each pair of images is based on the pixels which lie in the intersection of the windows of those two images. Only available when method="pearson". The resulting matrix may not be positive definite.
- use="everything": the calculation is based on all pixels, but any calculation of variance or covariance or correlation that includes an NA value gives an NA result in the corresponding entry in the matrix.
- use="all.obs": the calculation is based on all pixels, and an error occurs if any pixel has an NA value in any image.

Note that cor and cov are not generic, so you have to type cor.im, cov.im.

Value

A symmetric matrix.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

cor, cov

pairs.im

Examples

```
cor.im(bei.extra)
```

---

dclf.progress *Progress Plot of Test of Spatial Pattern*

Description

Generates a progress plot (envelope representation) of the Diggle-Cressie-Loosmore-Ford test or the Maximum Absolute Deviation test for a spatial point pattern.

Usage

```
dclf.progress(X, ...)
mad.progress(X, ...)
mctest.progress(X, fun = Lest, ...,
                exponent = 1, nrank = 1,
                interpolate = FALSE, alpha, rmin=0)
```

Arguments

X	Either a point pattern (object of class "ppp", "lpp" or other class), a fitted point process model (object of class "ppm", "kppm" or other class) or an envelope object (class "envelope").
...	Arguments passed to mctest.progress or to envelope. Useful arguments include fun to determine the summary function, nsim to specify the number of Monte Carlo simulations, alternative to specify one-sided or two-sided envelopes, and verbose=FALSE to turn off the messages.
fun	Function that computes the desired summary statistic for a point pattern.
exponent	Positive number. The exponent of the L^p distance. See Details.
nrank	Integer. The rank of the critical value of the Monte Carlo test, amongst the nsim simulated values. A rank of 1 means that the minimum and maximum simulated values will become the critical values for the test.
interpolate	Logical value indicating how to compute the critical value. If interpolate=FALSE (the default), a standard Monte Carlo test is performed, and the critical value is the largest simulated value of the test statistic (if nrank=1) or the nrank-th largest (if nrank is another number). If interpolate=TRUE, kernel density estimation is applied to the simulated values, and the critical value is the upper alpha quantile of this estimated distribution.
alpha	Optional. The significance level of the test. Equivalent to nrank/(nsim+1) where nsim is the number of simulations.
rmin	Optional. Left endpoint for the interval of r values on which the test statistic is calculated.

Details

The Diggle-Cressie-Loosmore-Ford test and the Maximum Absolute Deviation test for a spatial point pattern are described in dclf.test. These tests depend on the choice of an interval of distance values (the argument rinterval). A *progress plot* or *envelope representation* of the test (Baddeley et al, 2014) is a plot of the test statistic (and the corresponding critical value) against the length of the interval rinterval.

The command dclf.progress performs dclf.test on X using all possible intervals of the form $[0, R]$, and returns the resulting values of the test statistic, and the corresponding critical values of the test, as a function of R.

Similarly mad.progress performs mad.test using all possible intervals and returns the test statistic and critical value.

More generally, mctest.progress performs a test based on the L^p discrepancy between the curves. The deviation between two curves is measured by the pth root of the integral of the pth power of the absolute value of the difference between the two curves. The exponent p is given by the argument exponent. The case exponent=2 is the Cressie-Loosmore-Ford test, while exponent=Inf is the MAD test.

If the argument rmin is given, it specifies the left endpoint of the interval defining the test statistic: the tests are performed using intervals $[r_{\min}, R]$ where $R \geq r_{\min}$.

The result of each command is an object of class "fv" that can be plotted to obtain the progress plot. The display shows the test statistic (solid black line) and the Monte Carlo acceptance region (grey shading).

The significance level for the Monte Carlo test is nrank/(nsim+1). Note that nsim defaults to 99, so if the values of nrank and nsim are not given, the default is a test with significance level 0.01.

If X is an envelope object, then some of the data stored in X may be re-used:

- If X is an envelope object containing simulated functions, and fun=NULL, then the code will re-use the simulated functions stored in X.
- If X is an envelope object containing simulated point patterns, then fun will be applied to the stored point patterns to obtain the simulated functions. If fun is not specified, it defaults to Lest.
- Otherwise, new simulations will be performed, and fun defaults to Lest.

Value

An object of class "fv" that can be plotted to obtain the progress plot.

Author(s)

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

, Andrew Hardegen, Tom Lawrence, Gopal Nair and Robin Milne.

References

Baddeley, A., Diggle, P., Hardegen, A., Lawrence, T., Milne, R. and Nair, G. (2014) On tests of spatial pattern based on simulation envelopes. *Ecological Monographs* **84** (3) 477–489.

See Also

dclf.test and mad.test for the tests.

See plot.fv for information on plotting objects of class "fv".

Examples

```
plot(dclf.progress(cells, nsim=19))
```

---

dclf.sigtrace *Significance Trace of Cressie-Loosmore-Ford or Maximum Absolute Deviation Test*

Description

Generates a Significance Trace of the Diggle(1986)/ Cressie (1991)/ Loosmore and Ford (2006) test or the Maximum Absolute Deviation test for a spatial point pattern.

Usage

```
dclf.sigtrace(X, ...)
mad.sigtrace(X, ...)
mctest.sigtrace(X, fun=Lest, ...,
                exponent=1, interpolate=FALSE, alpha=0.05,
                confint=TRUE, rmin=0)
```

Arguments

X	Either a point pattern (object of class "ppp", "lpp" or other class), a fitted point process model (object of class "ppm", "kppm" or other class) or an envelope object (class "envelope").
...	Arguments passed to envelope or mctest.progress. Useful arguments include fun to determine the summary function, nsim to specify the number of Monte Carlo simulations, alternative to specify a one-sided test, and verbose=FALSE to turn off the messages.
fun	Function that computes the desired summary statistic for a point pattern.
exponent	Positive number. The exponent of the L^p distance. See Details.
interpolate	Logical value specifying whether to calculate the p-value by interpolation. If interpolate=FALSE (the default), a standard Monte Carlo test is performed, yielding a p-value of the form $(k+1)/(n+1)$ where n is the number of simulations and k is the number of simulated values which are more extreme than the observed value. If interpolate=TRUE, the p-value is calculated by applying kernel density estimation to the simulated values, and computing the tail probability for this estimated distribution.
alpha	Significance level to be plotted (this has no effect on the calculation but is simply plotted as a reference value).
confint	Logical value indicating whether to compute a confidence interval for the 'true' p-value.
rmin	Optional. Left endpoint for the interval of r values on which the test statistic is calculated.

Details

The Diggle (1986)/ Cressie (1991)/Loosmore and Ford (2006) test and the Maximum Absolute Deviation test for a spatial point pattern are described in dclf.test. These tests depend on the choice of an interval of distance values (the argument rinterval). A *significance trace* (Bowman and Azzalini, 1997; Baddeley et al, 2014, 2015; Baddeley, Rubak and Turner, 2015) of the test is a plot of the p-value obtained from the test against the length of the interval rinterval.

The command dclf.sigtrace performs dclf.test on X using all possible intervals of the form $[0, R]$, and returns the resulting p-values as a function of R.

Similarly mad.sigtrace performs mad.test using all possible intervals and returns the p-values.

More generally, mctest.sigtrace performs a test based on the L^p discrepancy between the curves. The deviation between two curves is measured by the pth root of the integral of the pth power of the absolute value of the difference between the two curves. The exponent p is given by the argument

exponent. The case exponent=2 is the Cressie-Loosmore-Ford test, while exponent=Inf is the MAD test.

If the argument rmin is given, it specifies the left endpoint of the interval defining the test statistic: the tests are performed using intervals $[r_{\min}, R]$ where $R \geq r_{\min}$.

The result of each command is an object of class "fv" that can be plotted to obtain the significance trace. The plot shows the Monte Carlo p-value (solid black line), the critical value 0.05 (dashed red line), and a pointwise 95% confidence band (grey shading) for the 'true' (Neyman-Pearson) p-value. The confidence band is based on the Agresti-Coull (1998) confidence interval for a binomial proportion (when interpolate=FALSE) or the delta method and normal approximation (when interpolate=TRUE).

If X is an envelope object and fun=NULL then the code will re-use the simulated functions stored in X.

Value

An object of class "fv" that can be plotted to obtain the significance trace.

Author(s)

Adrian Baddeley, Andrew Hardegen, Tom Lawrence, Robin Milne, Gopalan Nair and Suman Rakshit. Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

Agresti, A. and Coull, B.A. (1998) Approximate is better than "Exact" for interval estimation of binomial proportions. *American Statistician* **52**, 119–126.

Baddeley, A., Diggle, P., Hardegen, A., Lawrence, T., Milne, R. and Nair, G. (2014) On tests of spatial pattern based on simulation envelopes. *Ecological Monographs* **84**(3) 477–489.

Baddeley, A., Hardegen, A., Lawrence, L., Milne, R.K., Nair, G.M. and Rakshit, S. (2015) Pushing the envelope: extensions of graphical Monte Carlo tests. Unpublished manuscript.

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. Chapman and Hall/CRC Press.

Bowman, A.W. and Azzalini, A. (1997) *Applied smoothing techniques for data analysis: the kernel approach with S-Plus illustrations*. Oxford University Press, Oxford.

See Also

dclf.test for the tests; dclf.progress for progress plots.

See plot.fv for information on plotting objects of class "fv".

See also dg.sigtrace.

Examples

```
plot(dclf.sigtrace(cells, Lest, nsim=19))
```

dclf.test *Diggle-Cressie-Loosmore-Ford and Maximum Absolute Deviation Tests*

Description

Perform the Diggle (1986) / Cressie (1991) / Loosmore and Ford (2006) test or the Maximum Absolute Deviation test for a spatial point pattern.

Usage

```
dclf.test(X, ..., alternative=c("two.sided", "less", "greater"),
                  rinterval = NULL, leaveout=1,
                  scale=NULL, clamp=FALSE, interpolate=FALSE)

mad.test(X, ...,  alternative=c("two.sided", "less", "greater"),
                  rinterval = NULL, leaveout=1,
                  scale=NULL, clamp=FALSE, interpolate=FALSE)
```

Arguments

X	Data for the test. Either a point pattern (object of class "ppp", "lpp" or other class), a fitted point process model (object of class "ppm", "kppm" or other class), a simulation envelope (object of class "envelope") or a previous result of dclf.test or mad.test.
...	Arguments passed to envelope. Useful arguments include fun to determine the summary function, nsim to specify the number of Monte Carlo simulations, verbose=FALSE to turn off the messages, savefuns or savepatterns to save the simulation results, and use.theory described under Details.
alternative	The alternative hypothesis. A character string. The default is a two-sided alternative. See Details.
rinterval	Interval of values of the summary function argument r over which the maximum absolute deviation, or the integral, will be computed for the test. A numeric vector of length 2.
leaveout	Optional integer 0, 1 or 2 indicating how to calculate the deviation between the observed summary function and the nominal reference value, when the reference value must be estimated by simulation. See Details.
scale	Optional. A function in the R language which determines the relative scale of deviations, as a function of distance r. Summary function values for distance r will be *divided* by scale(r) before the test statistic is computed.
clamp	Logical value indicating how to compute deviations in a one-sided test. Deviations of the observed summary function from the theoretical summary function are initially evaluated as signed real numbers, with large positive values indicating consistency with the alternative hypothesis. If clamp=FALSE (the default), these values are not changed. If clamp=TRUE, any negative values are replaced by zero.

`interpolate` Logical value specifying whether to calculate the p-value by interpolation. If `interpolate=FALSE` (the default), a standard Monte Carlo test is performed, yielding a p-value of the form $(k+1)/(n+1)$ where n is the number of simulations and k is the number of simulated values which are more extreme than the observed value. If `interpolate=TRUE`, the p-value is calculated by applying kernel density estimation to the simulated values, and computing the tail probability for this estimated distribution.

Details

These functions perform hypothesis tests for goodness-of-fit of a point pattern dataset to a point process model, based on Monte Carlo simulation from the model.

`dclf.test` performs the test advocated by Loosmore and Ford (2006) which is also described in Diggle (1986), Cressie (1991, page 667, equation (8.5.42)) and Diggle (2003, page 14). See Baddeley et al (2014) for detailed discussion.

`mad.test` performs the 'global' or 'Maximum Absolute Deviation' test described by Ripley (1977, 1981). See Baddeley et al (2014).

The type of test depends on the type of argument `X`.

- If `X` is some kind of point pattern, then a test of Complete Spatial Randomness (CSR) will be performed. That is, the null hypothesis is that the point pattern is completely random.
- If `X` is a fitted point process model, then a test of goodness-of-fit for the fitted model will be performed. The model object contains the data point pattern to which it was originally fitted. The null hypothesis is that the data point pattern is a realisation of the model.
- If `X` is an envelope object generated by `envelope`, then it should have been generated with `savefuns=TRUE` or `savepatterns=TRUE` so that it contains simulation results. These simulations will be treated as realisations from the null hypothesis.
- Alternatively `X` could be a previously-performed test of the same kind (i.e. the result of calling `dclf.test` or `mad.test`). The simulations used to perform the original test will be re-used to perform the new test (provided these simulations were saved in the original test, by setting `savefuns=TRUE` or `savepatterns=TRUE`).

The argument `alternative` specifies the alternative hypothesis, that is, the direction of deviation that will be considered statistically significant. If `alternative="two.sided"` (the default), both positive and negative deviations (between the observed summary function and the theoretical function) are significant. If `alternative="less"`, then only negative deviations (where the observed summary function is lower than the theoretical function) are considered. If `alternative="greater"`, then only positive deviations (where the observed summary function is higher than the theoretical function) are considered.

In all cases, the algorithm will first call `envelope` to generate or extract the simulated summary functions. The number of simulations that will be generated or extracted, is determined by the argument `nsim`, and defaults to 99. The summary function that will be computed is determined by the argument `fun` (or the first unnamed argument in the list `...`) and defaults to `Kest` (except when `X` is an envelope object generated with `savefuns=TRUE`, when these functions will be taken).

The choice of summary function `fun` affects the power of the test. It is normally recommended to apply a variance-stabilising transformation (Ripley, 1981). If you are using the K function, the normal practice is to replace this by the L function (Besag, 1977) computed by `Lest`. If you

are using the F or G functions, the recommended practice is to apply Fisher's variance-stabilising transformation $\sin^{-1}\sqrt{x}$ using the argument `transform`. See the Examples.

The argument `rinterval` specifies the interval of distance values r which will contribute to the test statistic (either maximising over this range of values for `mad.test`, or integrating over this range of values for `dclf.test`). This affects the power of the test. General advice and experiments in Baddeley et al (2014) suggest that the maximum r value should be slightly larger than the maximum possible range of interaction between points. The `dclf.test` is quite sensitive to this choice, while the `mad.test` is relatively insensitive.

It is also possible to specify a pointwise test (i.e. taking a single, fixed value of distance r) by specifing `rinterval = c(r,r)`.

The argument `use.theory` passed to `envelope` determines whether to compare the summary function for the data to its theoretical value for CSR (`use.theory=TRUE`) or to the sample mean of simulations from CSR (`use.theory=FALSE`). The test statistic T is defined in equations (10.21) and (10.22) respectively on page 394 of Baddeley, Rubak and Turner (2015).

The argument `leaveout` specifies how to calculate the discrepancy between the summary function for the data and the nominal reference value, when the reference value must be estimated by simulation. The values `leaveout=0` and `leaveout=1` are both algebraically equivalent (Baddeley et al, 2014, Appendix) to computing the difference `observed - reference` where the `reference` is the mean of simulated values. The value `leaveout=2` gives the leave-two-out discrepancy proposed by Dao and Genton (2014).

Value

An object of class `"htest"`. Printing this object gives a report on the result of the test. The p-value is contained in the component `p.value`.

Handling Ties

If the observed value of the test statistic is equal to one or more of the simulated values (called a *tied value*), then the tied values will be assigned a random ordering, and a message will be printed.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Andrew Hardegen and Suman Rakshit.

References

Baddeley, A., Diggle, P.J., Hardegen, A., Lawrence, T., Milne, R.K. and Nair, G. (2014) On tests of spatial pattern based on simulation envelopes. *Ecological Monographs* **84**(3) 477–489.

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. Chapman and Hall/CRC Press.

Besag, J. (1977) Discussion of Dr Ripley's paper. *Journal of the Royal Statistical Society, Series B*, **39**, 193–195.

Cressie, N.A.C. (1991) *Statistics for spatial data*. John Wiley and Sons, 1991.

Dao, N.A. and Genton, M. (2014) A Monte Carlo adjusted goodness-of-fit test for parametric models describing spatial point patterns. *Journal of Graphical and Computational Statistics* **23**, 497–517.

Diggle, P. J. (1986). Displaced amacrine cells in the retina of a rabbit : analysis of a bivariate spatial point pattern. *J. Neuroscience Methods* **18**, 115–125.

Diggle, P.J. (2003) *Statistical analysis of spatial point patterns*, Second edition. Arnold.

Loosmore, N.B. and Ford, E.D. (2006) Statistical inference using the G or K point pattern spatial statistics. *Ecology* **87**, 1925–1931.

Ripley, B.D. (1977) Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, Series B*, **39**, 172 – 212.

Ripley, B.D. (1981) *Spatial statistics*. John Wiley and Sons.

See Also

envelope, dclf.progress

Examples

```
dclf.test(cells, Lest, nsim=39)
m <- mad.test(cells, Lest, verbose=FALSE, rinterval=c(0, 0.1), nsim=19)
m
# extract the p-value
m$p.value
# variance stabilised G function
dclf.test(cells, Gest, transform=expression(asin(sqrt(.))),
                  verbose=FALSE, nsim=19)

## one-sided test
ml <- mad.test(cells, Lest, verbose=FALSE, nsim=19, alternative="less")

## scaled
mad.test(cells, Kest, verbose=FALSE, nsim=19,
         rinterval=c(0.05, 0.2),
         scale=function(r) { r })
```

---

density.ppp *Kernel Smoothed Intensity of Point Pattern*

Description

Compute a kernel smoothed intensity function from a point pattern.

Usage

```
  ## S3 method for class 'ppp'
density(x, sigma=NULL, ...,
        weights=NULL, edge=TRUE, varcov=NULL,
        at="pixels", leaveoneout=TRUE,
        adjust=1, diggle=FALSE,
        se=FALSE, wtype=c("value", "multiplicity"),
```

```
          kernel="gaussian",
          scalekernel=is.character(kernel),
          positive=FALSE, verbose=TRUE, sameas)
```

Arguments

x	Point pattern (object of class "ppp").
sigma	The smoothing bandwidth (the amount of smoothing). The standard deviation of the isotropic smoothing kernel. Either a numerical value, or a function that computes an appropriate value of sigma.
weights	Optional weights to be attached to the points. A numeric vector, numeric matrix, an expression, or a pixel image.
...	Additional arguments passed to pixellate.ppp and as.mask to determine the pixel resolution, or passed to sigma if it is a function.
edge	Logical value indicating whether to apply edge correction.
varcov	Variance-covariance matrix of anisotropic smoothing kernel. Incompatible with sigma.
at	String specifying whether to compute the intensity values at a grid of pixel locations (at="pixels") or only at the points of x (at="points").
leaveoneout	Logical value indicating whether to compute a leave-one-out estimator. Applicable only when at="points".
adjust	Optional. Adjustment factor for the smoothing parameter.
diggle	Logical. If TRUE, use the Jones-Diggle improved edge correction, which is more accurate but slower to compute than the default correction.
kernel	The smoothing kernel. A character string specifying the smoothing kernel (current options are "gaussian", "epanechnikov", "quartic" or "disc"), or a pixel image (object of class "im") containing values of the kernel, or a function(x,y) which yields values of the kernel.
scalekernel	Logical value. If scalekernel=TRUE, then the kernel will be rescaled to the bandwidth determined by sigma and varcov: this is the default behaviour when kernel is a character string. If scalekernel=FALSE, then sigma and varcov will be ignored: this is the default behaviour when kernel is a function or a pixel image.
se	Logical value indicating whether to compute standard errors as well.
wtype	Character string (partially matched) specifying how the weights should be interpreted for the calculation of standard error. See Details.
positive	Logical value indicating whether to force all density values to be positive numbers. Default is FALSE.
verbose	Logical value indicating whether to issue warnings about numerical problems and conditions.
sameas	Optional. The result of a previous evaluation of density.ppp. Smoothing will be performed using the same kernel and bandwidth that were used to produce sameas. Namely the values of the arguments kernel, sigma, varcov, scalekernel and adjust will be overwritten by the values that were used to produce sameas.

Details

This is a method for the generic function density.

It computes a fixed-bandwidth kernel estimate (Diggle, 1985) of the intensity function of the point process that generated the point pattern x.

The amount of smoothing is controlled by sigma if it is specified.

By default, smoothing is performed using a Gaussian kernel. The resulting density estimate is the convolution of the isotropic Gaussian kernel, of standard deviation sigma, with point masses at each of the data points in x.

Anisotropic kernels, and non-Gaussian kernels, are also supported. Each point has unit weight, unless the argument weights is given.

If edge=TRUE (the default), the intensity estimate is corrected for edge effect bias.

If at="pixels" (the default), the result is a pixel image giving the estimated intensity at each pixel in a grid. If at="points", the result is a numeric vector giving the estimated intensity at each of the original data points in x.

Value

By default, the result is a pixel image (object of class "im"). Pixel values are estimated intensity values, expressed in "points per unit area".

If at="points", the result is a numeric vector of length equal to the number of points in x. Values are estimated intensity values at the points of x.

In either case, the return value has attributes "sigma" and "varcov" which report the smoothing bandwidth that was used.

If weights is a matrix with more than one column, then the result is a list of images (if at="pixels") or a matrix of numerical values (if at="points").

If se=TRUE, the result is a list with two elements named estimate and SE, each of the format described above.

Amount of smoothing

The amount of smoothing is determined by the arguments sigma, varcov and adjust.

- if sigma is a single numerical value, this is taken as the standard deviation of the isotropic Gaussian kernel.
- alternatively sigma may be a function that computes an appropriate bandwidth from the data point pattern by calling sigma(x). To perform automatic bandwidth selection using cross-validation, it is recommended to use the functions bw.diggle, bw.CvL, bw.scott or bw.ppl.
- The smoothing kernel may be made anisotropic by giving the variance-covariance matrix varcov. The arguments sigma and varcov are incompatible.
- Alternatively sigma may be a vector of length 2 giving the standard deviations of the x and y coordinates, thus equivalent to varcov = diag(rep(sigma^2, 2)).
- if neither sigma nor varcov is specified, an isotropic Gaussian kernel will be used, with a default value of sigma calculated by a simple rule of thumb that depends only on the size of the window.

- The argument adjust makes it easy for the user to change the bandwidth specified by any of the rules above. The value of sigma will be multiplied by the factor adjust. The matrix varcov will be multiplied by adjust^2. To double the smoothing bandwidth, set adjust=2.
- An infinite bandwidth, sigma=Inf or adjust=Inf, is permitted, and yields an intensity estimate which is constant over the spatial domain.

Edge correction

If edge=TRUE, the intensity estimate is corrected for edge effect bias in one of two ways:

- If diggle=FALSE (the default) the intensity estimate is correted by dividing it by the convolution of the Gaussian kernel with the window of observation. This is the approach originally described in Diggle (1985). Thus the intensity value at a point u is

$$\hat{\lambda}(u) = e(u) \sum_i k(x_i - u) w_i$$

  where k is the Gaussian smoothing kernel, $e(u)$ is an edge correction factor, and w_i are the weights.
- If diggle=TRUE then the code uses the improved edge correction described by Jones (1993) and Diggle (2010, equation 18.9). This has been shown to have better performance (Jones, 1993) but is slightly slower to compute. The intensity value at a point u is

$$\hat{\lambda}(u) = \sum_i k(x_i - u) w_i e(x_i)$$

  where again k is the Gaussian smoothing kernel, $e(x_i)$ is an edge correction factor, and w_i are the weights.

In both cases, the edge correction term $e(u)$ is the reciprocal of the kernel mass inside the window:

$$\frac{1}{e(u)} = \int_W k(v - u) \, \mathrm{d}v$$

where W is the observation window.

Smoothing kernel

By default, smoothing is performed using a Gaussian kernel.

The choice of smoothing kernel is determined by the argument kernel. This should be a character string giving the name of a recognised two-dimensional kernel (current options are "gaussian", "epanechnikov", "quartic" or "disc"), or a pixel image (object of class "im") containing values of the kernel, or a function(x,y) which yields values of the kernel. The default is a Gaussian kernel.

If scalekernel=TRUE then the kernel values will be rescaled according to the arguments sigma, varcov and adjust as explained above, effectively treating kernel as the template kernel with standard deviation equal to 1. This is the default behaviour when kernel is a character string. If scalekernel=FALSE, the kernel values will not be altered, and the arguments sigma, varcov and adjust are ignored. This is the default behaviour when kernel is a pixel image or a function.

Desired output

If `at="pixels"` (the default), intensity values are computed at every location u in a fine grid, and are returned as a pixel image. The point pattern is first discretised using `pixellate.ppp`, then the intensity is computed using the Fast Fourier Transform. Accuracy depends on the pixel resolution and the discretisation rule. The pixel resolution is controlled by the arguments `...` passed to `as.mask` (specify the number of pixels by `dimyx` or the pixel size by `eps`). The discretisation rule is controlled by the arguments `...` passed to `pixellate.ppp` (the default rule is that each point is allocated to the nearest pixel centre; this can be modified using the arguments `fractional` and `preserve`).

If `at="points"`, the intensity values are computed to high accuracy at the points of `x` only. Computation is performed by directly evaluating and summing the kernel contributions without discretising the data. The result is a numeric vector giving the density values. The intensity value at a point x_i is (if `diggle=FALSE`)

$$\hat{\lambda}(x_i) = e(x_i) \sum_j k(x_j - x_i) w_j$$

or (if `diggle=TRUE`)

$$\hat{\lambda}(x_i) = \sum_j k(x_j - x_i) w_j e(x_j)$$

If `leaveoneout=TRUE` (the default), then the sum in the equation is taken over all j not equal to i, so that the intensity value at a data point is the sum of kernel contributions from all *other* data points. If `leaveoneout=FALSE` then the sum is taken over all j, so that the intensity value at a data point includes a contribution from the same point.

Weights

If `weights` is a matrix with more than one column, then the calculation is effectively repeated for each column of weights. The result is a list of images (if `at="pixels"`) or a matrix of numerical values (if `at="points"`).

The argument `weights` can also be an `expression`. It will be evaluated in the data frame `as.data.frame(x)` to obtain a vector or matrix of weights. The expression may involve the symbols `x` and `y` representing the Cartesian coordinates, the symbol `marks` representing the mark values if there is only one column of marks, and the names of the columns of marks if there are several columns.

The argument `weights` can also be a pixel image (object of class `"im"`). numerical weights for the data points will be extracted from this image (by looking up the pixel values at the locations of the data points in `x`).

Standard error

If `se=TRUE`, the standard error of the estimate will also be calculated. The calculation assumes a Poisson point process.

If `weights` are given, then the calculation of standard error depends on the interpretation of the weights. This is controlled by the argument `wtype`.

- If `wtype="value"` (the default), the weights are interpreted as numerical values observed at the data locations. Roughly speaking, standard errors are proportional to the absolute values of the weights.

- If wtype="multiplicity" the weights are interpreted as multiplicities so that a weight of 2 is equivalent to having a pair of duplicated points at the data location. Roughly speaking, standard errors are proportional to the square roots of the weights. Negative weights are not permitted.

The default rule is now wtype="value" but previous versions of density.ppp (in **spatstat.explore** versions 3.1-0 and earlier) effectively used wtype="multiplicity".

The meaning of density.ppp

This function is often misunderstood.

The result of density.ppp is not a spatial smoothing of the marks or weights attached to the point pattern. To perform spatial interpolation of values that were observed at the points of a point pattern, use Smooth.ppp.

The result of density.ppp is not a probability density. It is an estimate of the *intensity function* of the point process that generated the point pattern data. Intensity is the expected number of random points per unit area. The units of intensity are "points per unit area". Intensity is usually a function of spatial location, and it is this function which is estimated by density.ppp. The integral of the intensity function over a spatial region gives the expected number of points falling in this region.

Inspecting an estimate of the intensity function is usually the first step in exploring a spatial point pattern dataset. For more explanation, see Baddeley, Rubak and Turner (2015) or Diggle (2003, 2010).

If you have two (or more) types of points, and you want a probability map or relative risk surface (the spatially-varying probability of a given type), use relrisk.

Technical issue: Negative Values

Negative and zero values of the density estimate are possible when at="pixels" because of numerical errors in finite-precision arithmetic.

By default, density.ppp does not try to repair such errors. This would take more computation time and is not always needed. (Also it would not be appropriate if weights include negative values.)

To ensure that the resulting density values are always positive, set positive=TRUE.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>

References

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. Chapman and Hall/CRC Press.

Bithell, J.F. (1990) An application of density estimation to geographical epidemiology. *Statistics in Medicine* **9**, 691–701.

Diggle, P.J. (1985) A kernel method for smoothing point process data. *Applied Statistics* (Journal of the Royal Statistical Society, Series C) **34** (1985) 138–147.

Diggle, P.J. (2003) *Statistical analysis of spatial point patterns*, Second edition. Arnold.

Diggle, P.J. (2010) Nonparametric methods. Chapter 18, pp. 299–316 in A.E. Gelfand, P.J. Diggle, M. Fuentes and P. Guttorp (eds.) *Handbook of Spatial Statistics*, CRC Press, Boca Raton, FL.

Jones, M.C. (1993) Simple boundary corrections for kernel density estimation. *Statistics and Computing* **3**, 135–146.

See Also

To select the bandwidth `sigma` automatically by cross-validation, use `bw.diggle`, `bw.CvL`, `bw.scott` or `bw.ppl`.

To perform spatial interpolation of values that were observed at the points of a point pattern, use `Smooth.ppp`.

For adaptive nonparametric estimation, see `adaptive.density`. For data sharpening, see `sharpen.ppp`.

To compute a relative risk surface or probability map for two (or more) types of points, use `relrisk`.

For information about the data structures, see `ppp.object`, `im.object`.

Examples

```
if(interactive()) {
  opa <- par(mfrow=c(1,2))
  plot(density(cells, 0.05))
  plot(density(cells, 0.05, diggle=TRUE))
  par(opa)
  v <- diag(c(0.05, 0.07)^2)
  plot(density(cells, varcov=v))
}
# automatic bandwidth selection
plot(density(cells, sigma=bw.diggle(cells)))
# equivalent:
plot(density(cells, bw.diggle))
# evaluate intensity at points
density(cells, 0.05, at="points")

# non-Gaussian kernel
plot(density(cells, sigma=0.4, kernel="epanechnikov"))

if(interactive()) {
  # see effect of changing pixel resolution
  opa <- par(mfrow=c(1,2))
  plot(density(cells, sigma=0.4))
  plot(density(cells, sigma=0.4, eps=0.05))
  par(opa)
}

# relative risk calculation by hand (see relrisk.ppp)
lung <- split(chorley)$lung
larynx <- split(chorley)$larynx
D <- density(lung, sigma=2)
plot(density(larynx, sigma=2, weights=1/D))
```

density.psp *Kernel Smoothing of Line Segment Pattern*

Description

Compute a kernel smoothed intensity function from a line segment pattern.

Usage

```
  ## S3 method for class 'psp'
density(x, sigma, ..., weights=NULL, edge=TRUE,
                  method=c("FFT", "C", "interpreted"),
                  at=NULL)
```

Arguments

x	Line segment pattern (object of class "psp") to be smoothed.
sigma	Standard deviation of isotropic Gaussian smoothing kernel.
...	Extra arguments, including arguments passed to as.mask to determine the resolution of the resulting image.
weights	Optional. Numerical weights for each line segment. A numeric vector, of length equal to the number of segments in x.
edge	Logical flag indicating whether to apply edge correction.
method	Character string (partially matched) specifying the method of computation. Option "FFT" is the fastest, while "C" is the most accurate.
at	Optional. An object specifying the locations where density values should be computed. Either a window (object of class "owin") or a point pattern (object of class "ppp" or "lpp").

Details

This is the method for the generic function density for the class "psp" (line segment patterns).

A kernel estimate of the intensity of the line segment pattern is computed. The result is the convolution of the isotropic Gaussian kernel, of standard deviation sigma, with the line segments. The result is computed as follows:

- if method="FFT" (the default), the line segments are discretised using pixellate.psp, then the Fast Fourier Transform is used to calculate the convolution. This method is the fastest, but is slightly less accurate. Accuracy can be improved by increasing pixel resolution.
- if method="C" the exact value of the convolution at the centre of each pixel is computed analytically using C code;
- if method="interpreted", the exact value of the convolution at the centre of each pixel is computed analytically using R code. This method is the slowest.

If edge=TRUE this result is adjusted for edge effects by dividing it by the convolution of the same Gaussian kernel with the observation window.

If weights are given, then the contribution from line segment i is multiplied by the value of weights[i].

If the argument at is given, then it specifies the locations where density values should be computed.

- If at is a window, then the window is converted to a binary mask using the arguments ..., and density values are computed at the centre of each pixel in this mask. The result is a pixel image.
- If at is a point pattern, then density values are computed at each point location, and the result is a numeric vector.

Value

A pixel image (object of class "im") or a numeric vector.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

psp.object, im.object, density

Examples

```
L <- psp(runif(20),runif(20),runif(20),runif(20), window=owin())
D <- density(L, sigma=0.03)
plot(D, main="density(L)")
plot(L, add=TRUE)
```

---

density.splitppp *Kernel Smoothed Intensity of Split Point Pattern*

Description

Compute a kernel smoothed intensity function for each of the components of a split point pattern, or each of the point patterns in a list.

Usage

```
  ## S3 method for class 'splitppp'
density(x, ..., weights=NULL, se=FALSE)

  ## S3 method for class 'ppplist'
density(x, ..., weights=NULL, se=FALSE)
```

Arguments

x	Split point pattern (object of class "splitppp" created by split.ppp) to be smoothed. Alternatively a list of point patterns, of class "ppplist".
...	Arguments passed to density.ppp to control the smoothing, pixel resolution, edge correction etc.
weights	Numerical weights for the points. See Details.
se	Logical value indicating whether to compute standard errors as well.

Details

This is a method for the generic function density.

The argument x should be a list of point patterns, and should belong to one of the classes "ppplist" or "splitppp".

Typically x is obtained by applying the function split.ppp to a point pattern y by calling split(y). This splits the points of y into several sub-patterns.

A kernel estimate of the intensity function of each of the point patterns is computed using density.ppp.

The return value is usually a list, each of whose entries is a pixel image (object of class "im"). The return value also belongs to the class "solist" and can be plotted or printed.

If the argument at="points" is given, the result is a list of numeric vectors giving the intensity values at the data points.

If se=TRUE, the result is a list with two elements named estimate and SE, each of the format described above.

The argument weights specifies numerical case weights for the data points. Normally it should be a list, with the same length as x. The entry weights[[i]] will determine the case weights for the pattern x[[i]], and may be given in any format acceptable to density.ppp. For example, weights[[i]] can be a numeric vector of length equal to npoints(x[[i]]), a single numeric value, a numeric matrix, a pixel image (object of class "im"), an expression, or a function of class "funxy".

For convenience, weights can also be a single expression, or a single pixel image (object of class "im"), or a single function of class "funxy".

Value

A list of pixel images (objects of class "im") which can be plotted or printed; or a list of numeric vectors giving the values at specified points.

If se=TRUE, the result is a list with two elements named estimate and SE, each of the format described above.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

ppp.object, im.object

Examples

```
Z <- density(split(amacrine), 0.05)
plot(Z)
```

`densityAdaptiveKernel.ppp` *Adaptive Kernel Estimate of Intensity of Point Pattern*

Description

Computes an adaptive estimate of the intensity function of a point pattern using a variable-bandwidth smoothing kernel.

Usage

```
## S3 method for class 'ppp'
densityAdaptiveKernel(X, bw, ...,
       weights=NULL,
       at=c("pixels", "points"),
       edge=TRUE, ngroups)
```

Arguments

`X`	Point pattern (object of class `"ppp"`).
`bw`	Numeric vector of smoothing bandwidths for each point in `X`, or a pixel image giving the smoothing bandwidth at each spatial location, or a spatial function of class `"funxy"` giving the smoothing bandwidth at each location. The default is to compute bandwidths using `bw.abram.ppp`.
`...`	Arguments passed to `bw.abram.ppp` to compute the smoothing bandwidths if `bw` is missing, or passed to `as.mask` to control the spatial resolution of the result.
`weights`	Optional vector of numeric weights for the points of `X`.
`at`	String specifying whether to compute the intensity values at a grid of pixel locations (`at="pixels"`) or only at the points of `x` (`at="points"`).
`edge`	Logical value indicating whether to perform edge correction.
`ngroups`	Number of groups into which the bandwidth values should be partitioned and discretised.

Details

This function computes a spatially-adaptive kernel estimate of the spatially-varying intensity from the point pattern `X` using the partitioning technique of Davies and Baddeley (2018).

The function `densityAdaptiveKernel` is generic. This file documents the method for point patterns, `densityAdaptiveKernel.ppp`.

The argument bw specifies the smoothing bandwidths to be applied to each of the points in X. It may be a numeric vector of bandwidth values, or a pixel image or function yielding the bandwidth values.

If the points of X are $x_1, \ldots, x_n$ and the corresponding bandwidths are $\sigma_1, \ldots, \sigma_n$ then the adaptive kernel estimate of intensity at a location u is

$$\hat{\lambda}(u) = \sum_{i=1}^{n} k(u, x_i, \sigma_i)$$

where $k(u, v, \sigma)$ is the value at u of the (possibly edge-corrected) smoothing kernel with bandwidth σ induced by a data point at v.

Exact computation of the estimate above can be time-consuming: it takes n times longer than fixed-bandwidth smoothing.

The partitioning method of Davies and Baddeley (2018) accelerates this computation by partitioning the range of bandwidths into ngroups intervals, correspondingly subdividing the points of the pattern X into ngroups sub-patterns according to bandwidth, and applying fixed-bandwidth smoothing to each sub-pattern.

The default value of ngroups is the integer part of the square root of the number of points in X, so that the computation time is only about $\sqrt{n}$ times slower than fixed-bandwidth smoothing. Any positive value of ngroups can be specified by the user. Specifying ngroups=Inf enforces exact computation of the estimate without partitioning. Specifying ngroups=1 is the same as fixed-bandwidth smoothing with bandwidth sigma=median(bw).

Value

If at="pixels" (the default), the result is a pixel image. If at="points", the result is a numeric vector with one entry for each data point in X.

Bandwidths and Bandwidth Selection

The function densityAdaptiveKernel computes one adaptive estimate of the intensity, determined by the smoothing bandwidth values bw.

Typically the bandwidth values are computed by first computing a pilot estimate of the intensity, then using bw.abram.ppp to compute the vector of bandwidths according to Abramson's rule. This involves specifying a global bandwidth h0.

The default bandwidths may work well in many contexts, but for optimal bandwidth selection, this calculation should be performed repeatedly with different values of h0 to optimise the value of h0. This can be computationally demanding; we recommend the function multiscale.density in the **sparr** package which supports much faster bandwidth selection, using the FFT method of Davies and Baddeley (2018).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Tilman Davies.

References

Davies, T.M. and Baddeley, A. (2018) Fast computation of spatially adaptive kernel estimates. *Statistics and Computing*, **28**(4), 937-956.

Hall, P. and Marron, J.S. (1988) Variable window width kernel density estimates of probability densities. *Probability Theory and Related Fields*, **80**, 37-49.

Silverman, B.W. (1986) *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, New York.

See Also

bw.abram.ppp, density.ppp, adaptive.density, densityVoronoi, im.object.

See the function bivariate.density in the **sparr** package for a more flexible implementation, and multiscale.density for an implementation that is more efficient for bandwidth selection.

Examples

```
Z <- densityAdaptiveKernel(redwood, h0=0.1)
plot(Z, main="Adaptive kernel estimate")
points(redwood, col="white")
```

---

densityAdaptiveKernel.splitppp

Adaptive Kernel Estimate of Intensity for Split Point Pattern

---

Description

Computes an adaptive estimate of the intensity function (using a variable-bandwidth smoothing kernel) for each of the components of a split point pattern, or each of the point patterns in a list.

Usage

```
## S3 method for class 'splitppp'
densityAdaptiveKernel(X, bw=NULL, ..., weights=NULL)

## S3 method for class 'ppplist'
densityAdaptiveKernel(X, bw=NULL, ..., weights=NULL)
```

Arguments

X	Split point pattern (object of class "splitppp" created by split.ppp) to be smoothed. Alternatively a list of point patterns, of class "ppplist".
bw	Smoothing bandwidths. See Details.
...	Additional arguments passed to densityAdaptiveKernel.ppp. These may include arguments that will be passed to bw.abram.ppp to compute the smoothing bandwidths if bw is missing, and arguments passed to as.mask to control the spatial resolution of the result.
weights	Numerical weights for the points. See Details.

Details

This function computes a spatially-adaptive kernel estimate of the spatially-varying intensity for each of the point patterns in the list X, using densityAdaptiveKernel.ppp.

The argument bw specifies smoothing bandwidths for the data points. Normally it should be a list, with the same length as x. The entry bw[[i]] will determine the smoothing bandwidths for the pattern x[[i]], and may be given in any format acceptable to densityAdaptiveKernel.ppp. For example, bw[[i]] can be a numeric vector of length equal to npoints(x[[i]]), a single numeric value, a pixel image (object of class "im"), an expression, or a function of class "funxy". For convenience, bw can also be a single expression, or a single pixel image, or a single function. If bw is missing or NULL, the default is to compute bandwidths using bw.abram.ppp.

The argument weights specifies numerical case weights for the data points. Normally it should be a list, with the same length as x. The entry weights[[i]] will determine the case weights for the pattern x[[i]], and may be given in any format acceptable to density.ppp. For example, weights[[i]] can be a numeric vector of length equal to npoints(x[[i]]), a single numeric value, a numeric matrix, a pixel image (object of class "im"), an expression, or a function of class "funxy". For convenience, weights can also be a single expression, or a single pixel image (object of class "im"), or a single function of class "funxy". If weights is missing or NULL, all weights are assumed to be equal to 1.

Value

A list of pixel images (objects of class "im") which can be plotted or printed; or a list of numeric vectors giving the values at specified points.

Author(s)

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

See Also

densityAdaptiveKernel.ppp, bw.abram.ppp.

Examples

```
X <- amacrine
if(!interactive()) X <- X[c(TRUE,FALSE,FALSE,FALSE)]
Z <- densityAdaptiveKernel(split(X), h0=0.15)
plot(Z, main="Adaptive kernel estimate")
```

---

densityfun.ppp *Kernel Estimate of Intensity as a Spatial Function*

Description

Compute a kernel estimate of intensity for a point pattern, and return the result as a function of spatial location.

Usage

```
densityfun(X, ...)

## S3 method for class 'ppp'
densityfun(X, sigma = NULL, ...,
                        weights = NULL, edge = TRUE, diggle = FALSE)
```

Arguments

X	Point pattern (object of class "ppp").
sigma	Smoothing bandwidth, or bandwidth selection function, passed to density.ppp.
...	Additional arguments passed to density.ppp.
weights	Optional vector of weights associated with the points of X.
edge, diggle	Logical arguments controlling the edge correction. Arguments passed to density.ppp.

Details

The commands densityfun and density both perform kernel estimation of the intensity of a point pattern. The difference is that density returns a pixel image, containing the estimated intensity values at a grid of locations, while densityfun returns a function(x,y) which can be used to compute the intensity estimate at *any* spatial locations with coordinates x,y. For purposes such as model-fitting it is more accurate to use densityfun.

Value

A function with arguments x,y,drop. The function also belongs to the class "densityfun" which has methods for print and as.im. It also belongs to the class "funxy" which has methods for plot, contour and persp.

Using the result of densityfun

If f <- densityfun(X), where X is a two-dimensional point pattern, the resulting object f is a function in the R language.

By calling this function, the user can evaluate the estimated intensity at any desired spatial locations.

Additionally f belongs to other classes which allow it to be printed and plotted easily.

The function f has arguments x,y,drop.

- The arguments x,y of f specify the query locations. They can be numeric vectors of coordinates. Alternatively x can be a point pattern (or data acceptable to as.ppp) and y is omitted. The result of f(x,y) is a numeric vector giving the values of the intensity.
- The argument drop of f specifies how to handle query locations which are outside the window of the original data. If drop=TRUE (the default), such locations are ignored. If drop=FALSE, a value of NA is returned for each such location.

Note that the smoothing parameters, such as the bandwidth sigma, are assigned when densityfun is executed. Smoothing parameters are fixed inside the function f and cannot be changed by arguments of f.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

density.

To interpolate values observed at the points, use Smoothfun.

Examples

```
f <- densityfun(swedishpines)
f
f(42, 60)
X <- runifpoint(2, Window(swedishpines))
f(X)
plot(f)
```

---

densityHeat *Diffusion Estimate of Point Pattern Intensity*

Description

Computes a diffusion estimate of intensity for a point pattern.

Usage

```
densityHeat(x, sigma, ...)
```

Arguments

x	Point pattern (object of class "ppp" or another class).
sigma	Smoothing bandwidth. Usually a single number giving the equivalent standard deviation of the smoother.
...	Additional arguments depending on the method.

Details

The generic function densityHeat computes an estimate of point process intensity using a diffusion kernel method.

Further details depend on the class of point pattern x. See the help file for the appropriate method.

Value

Depends on the class of x.

Author(s)

Adrian Baddeley and Tilman Davies.

See Also

For two-dimensional point patterns (objects of class "ppp"), the diffusion kernel estimator is densityHeat.ppp. The usual kernel estimator is density.ppp, and the tessellation-based estimator is adaptive.density.

---

densityHeat.ppp *Diffusion Estimate of Point Pattern Intensity*

Description

Computes the diffusion estimate of the intensity of a point pattern.

Usage

```
## S3 method for class 'ppp'
densityHeat(x, sigma, ..., weights=NULL,
          connect=8, symmetric=FALSE,
          sigmaX=NULL, k=1, show=FALSE, se=FALSE,
          at=c("pixels", "points"),
          leaveoneout = TRUE,
          extrapolate = FALSE, coarsen = TRUE,
          verbose=TRUE, internal=NULL)
```

Arguments

x	Point pattern (object of class "ppp").
sigma	Smoothing bandwidth. A single number giving the equivalent standard deviation of the smoother. Alternatively, a pixel image (class "im") or a function(x,y) giving the spatially-varying bandwidth.
...	Arguments passed to pixellate.ppp controlling the pixel resolution.
weights	Optional numeric vector of weights associated with each point of x.
connect	Grid connectivity: either 4 or 8.
symmetric	Logical value indicating whether to *force* the algorithm to use a symmetric random walk.
sigmaX	Numeric vector of bandwidths, one associated with each data point in x. See Details.
k	Integer. Calculations will be performed by repeatedly multiplying the current state by the k-step transition matrix.
show	Logical value indicating whether to plot successive iterations.
se	Logical value indicating whether to compute standard errors.

at	Character string specifying whether to compute values at a grid of pixels (at="pixels", the default) or at the data points of x (at="points").
leaveoneout	Logical value specifying whether to compute a leave-one-out estimate at each data point, when at="points".
extrapolate	Logical value specifying whether to use Richardson extrapolation to improve the accuracy of the computation.
coarsen	Logical value, controlling the calculation performed when extrapolate=TRUE. See Details.
verbose	Logical value specifying whether to print progress reports.
internal	Developer use only.

Details

This command computes a diffusion kernel estimate of point process intensity from the observed point pattern x.

The function densityHeat is generic, with methods for point patterns in two dimensions (class "ppp") and point patterns on a linear network (class "lpp"). The function densityHeat.ppp described here is the method for class "ppp". Given a two-dimensional point pattern x, it computes a diffusion kernel estimate of the intensity of the point process which generated x.

Diffusion kernel estimates were developed by Botev et al (2010), Barry and McIntyre (2011) and Baddeley et al (2022).

Barry and McIntyre (2011) proposed an estimator for point process intensity based on a random walk on the pixel grid inside the observation window. Baddeley et al (2022) showed that the Barry-McIntyre method is a special case of the *diffusion estimator* proposed by Botev et al (2010).

The original Barry-McIntyre algorithm assumes a symmetric random walk (i.e. each possible transition has the same probability p) and requires a square pixel grid (i.e. equal spacing in the x and y directions). Their original algorithm is used if symmetric=TRUE. Use the ... arguments to ensure a square grid: for example, the argument eps specifies a square grid with spacing eps units.

The more general algorithm used here (Baddeley et al, 2022) does not require a square grid of pixels. If the pixel grid is not square, and if symmetric=FALSE (the default), then the random walk is not symmetric, in the sense that the probabilities of different jumps will be different, in order to ensure that the smoothing is isotropic.

This implementation also includes two generalizations to the case of adaptive smoothing (Baddeley et al, 2022).

In the first version of adaptive smoothing, the bandwidth is spatially-varying. The argument sigma should be a pixel image (class "im") or a function(x,y) specifying the bandwidth at each spatial location. The smoothing is performed by solving the heat equation with spatially-varying parameters.

In the second version of adaptive smoothing, each data point in x is smoothed using a separate bandwidth. The argument sigmaX should be a numeric vector specifying the bandwidth for each point of x. The smoothing is performed using the lagged arrival algorithm. The argument sigma can be omitted.

If extrapolate=FALSE (the default), calculations are performed using the Euler scheme for the heat equation. If extrapolate=TRUE, the accuracy of the result will be improved by applying

Richardson extrapolation (Baddeley et al, 2022, Section 4). After computing the intensity estimate using the Euler scheme on the desired pixel grid, another estimate is computed using the same method on another pixel grid, and the two estimates are combined by Richardson extrapolation to obtain a more accurate result. The second grid is coarser than the original grid if `coarsen=TRUE` (the default), and finer than the original grid if `coarsen=FALSE`. Setting `extrapolate=TRUE` increases computation time by 35% if `coarsen=TRUE` and by 400% if `coarsen=FALSE`.

Value

Pixel image (object of class `"im"`) giving the estimated intensity of the point process.

If `se=TRUE`, the result has an attribute `"se"` which is another pixel image giving the estimated standard error.

If `at="points"` then the result is a numeric vector with one entry for each point of `x`.

Author(s)

Adrian Baddeley and Tilman Davies.

References

Baddeley, A., Davies, T., Rakshit, S., Nair, G. and McSwiggan, G. (2022) Diffusion smoothing for spatial point patterns. *Statistical Science* **37** (1) 123–142.

Barry, R.P. and McIntyre, J. (2011) Estimating animal densities and home range in regions with irregular boundaries and holes: a lattice-based alternative to the kernel density estimator. *Ecological Modelling* **222**, 1666–1672.

Botev, Z.I., Grotowski, J.F. and Kroese, D.P. (2010) Kernel density estimation via diffusion. *Annals of Statistics* **38**, 2916–2957.

See Also

`density.ppp` for the usual kernel estimator, and `adaptive.density` for the tessellation-based estimator.

Examples

```
online <- interactive()
if(!online) op <- spatstat.options(npixel=32)

X <- runifpoint(25, letterR)
Z <- densityHeat(X, 0.2)
if(online) {
  plot(Z, main="Diffusion estimator")
  plot(X, add=TRUE, pch=16)
  integral(Z) # should equal 25
}

Z <- densityHeat(X, 0.2, se=TRUE)
Zse <- attr(Z, "se")
if(online) plot(solist(estimate=Z, SE=Zse), main="")
```

```
Zex <- densityHeat(X, 0.2, extrapolate=TRUE)

ZS <- densityHeat(X, 0.2, symmetric=TRUE, eps=0.125)
if(online) {
  plot(ZS, main="fixed bandwidth")
  plot(X, add=TRUE, pch=16)
}

sig <- function(x,y) { (x-1.5)/10 }
ZZ <- densityHeat(X, sig)
if(online) {
  plot(ZZ, main="adaptive (I)")
  plot(X, add=TRUE, pch=16)
}

sigX <- sig(X$x, X$y)
AA <- densityHeat(X, sigmaX=sigX)
if(online) {
  plot(AA, main="adaptive (II)")
  plot(X, add=TRUE, pch=16)
}
if(!online) spatstat.options(op)
```

`densityVoronoi` *Intensity Estimate of Point Pattern Using Voronoi-Dirichlet Tessellation*

Description

Computes an adaptive estimate of the intensity function of a point pattern using the Dirichlet-Voronoi tessellation.

Usage

```
densityVoronoi(X, ...)

## S3 method for class 'ppp'
densityVoronoi(X, f = 1, ...,
                          counting=FALSE,
                          fixed=FALSE,
                          nrep = 1, verbose=TRUE)
```

Arguments

`X`	Point pattern dataset (object of class `"ppp"`).
`f`	Fraction (between 0 and 1 inclusive) of the data points that will be used to build a tessellation for the intensity estimate.
`...`	Arguments passed to `as.im` determining the pixel resolution of the result.

counting	Logical value specifying the choice of estimation method. See Details.
fixed	Logical. If FALSE (the default), the data points are independently randomly thinned, so the number of data points that are retained is random. If TRUE, the number of data points retained is fixed. See Details.
nrep	Number of independent repetitions of the randomised procedure.
verbose	Logical value indicating whether to print progress reports.

Details

This function is an alternative to density.ppp. It computes an estimate of the intensity function of a point pattern dataset. The result is a pixel image giving the estimated intensity.

If f=1 (the default), the Voronoi estimate (Barr and Schoenberg, 2010) is computed: the point pattern X is used to construct a Voronoi/Dirichlet tessellation (see dirichlet); the areas of the Dirichlet tiles are computed; the estimated intensity in each tile is the reciprocal of the tile area. The result is a pixel image of intensity estimates which are constant on each tile of the tessellation.

If f=0, the intensity estimate at every location is equal to the average intensity (number of points divided by window area). The result is a pixel image of intensity estimates which are constant.

If f is strictly between 0 and 1, the estimation method is applied to a random subset of X. This randomised procedure is repeated nrep times, and the results are averaged. The subset is selected as follows:

- if fixed=FALSE, the dataset X is randomly thinned by deleting or retaining each point independently, with probability f of retaining a point.
- if fixed=TRUE, a random sample of fixed size m is taken from the dataset X, where m is the largest integer less than or equal to f*n and n is the number of points in X.

Then the intensity estimate is calculated as follows:

- if counting = FALSE (the default), the thinned pattern is used to construct a Dirichlet tessellation and form the Voronoi estimate (Barr and Schoenberg, 2010) which is then adjusted by a factor 1/f or n/m as appropriate. to obtain an estimate of the intensity of X in the tile.
- if counting = TRUE, the randomly selected subset A is used to construct a Dirichlet tessellation, while the complementary subset B (consisting of points that were not selected in the sample) is used for counting to calculate a quadrat count estimate of intensity. For each tile of the Dirichlet tessellation formed by A, we count the number of points of B falling in the tile, and divide by the area of the same tile, to obtain an estimate of the intensity of the pattern B in the tile. This estimate is adjusted by 1/(1-f) or n/(n-m) as appropriate to obtain an estimate of the intensity of X in the tile.

Ogata et al. (2003) and Ogata (2004) estimated intensity using the Dirichlet-Voronoi tessellation in a modelling context. Baddeley (2007) proposed intensity estimation by subsampling with 0 < f < 1, and used the technique described above with fixed=TRUE and counting=TRUE. Barr and Schoenberg (2010) described and analysed the Voronoi estimator (corresponding to f=1). Moradi et al (2019) developed the subsampling technique with fixed=FALSE and counting=FALSE and called it the *smoothed Voronoi estimator*.

Value

A pixel image (object of class "im") whose values are estimates of the intensity of X.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk> and Mehdi Moradi <m2.moradi@yahoo.com>.

References

Baddeley, A. (2007) Validation of statistical models for spatial point patterns. In J.G. Babu and E.D. Feigelson (eds.) *SCMA IV: Statistical Challenges in Modern Astronomy IV*, volume 317 of Astronomical Society of the Pacific Conference Series, San Francisco, California USA, 2007. Pages 22–38.

Barr, C., and Schoenberg, F.P. (2010). On the Voronoi estimator for the intensity of an inhomogeneous planar Poisson process. *Biometrika* **97** (4), 977–984.

Moradi, M., Cronie, 0., Rubak, E., Lachieze-Rey, R., Mateu, J. and Baddeley, A. (2019) Resample-smoothing of Voronoi intensity estimators. *Statistics and Computing* **29** (5) 995–1010.

Ogata, Y. (2004) Space-time model for regional seismicity and detection of crustal stress changes. *Journal of Geophysical Research*, **109**, 2004.

Ogata, Y., Katsura, K. and Tanemura, M. (2003). Modelling heterogeneous space-time occurrences of earthquakes and its residual analysis. *Applied Statistics* **52** 499–509.

See Also

adaptive.density, density.ppp, dirichlet, im.object.

Examples

```
plot(densityVoronoi(nztrees, 1, f=1), main="Voronoi estimate")
nr <- if(interactive()) 100 else 5
plot(densityVoronoi(nztrees, f=0.5, nrep=nr), main="smoothed Voronoi estimate")
```

---

deriv.fv *Calculate Derivative of Function Values*

Description

Applies numerical differentiation to the values in selected columns of a function value table.

Usage

```
## S3 method for class 'fv'
deriv(expr, which = "*", ...,
          method=c("spline", "numeric"),
          kinks=NULL,
          periodic=FALSE,
          Dperiodic=periodic)
```

Arguments

expr	Function values to be differentiated. A function value table (object of class "fv", see fv.object).
which	Character vector identifying which columns of the table should be differentiated. Either a vector containing names of columns, or one of the wildcard strings "*" or "." explained below.
...	Extra arguments passed to smooth.spline to control the differentiation algorithm, if method="spline".
method	Differentiation method. A character string, partially matched to either "spline" or "numeric".
kinks	Optional vector of x values where the derivative is allowed to be discontinuous.
periodic	Logical value indicating whether the function expr is periodic.
Dperiodic	Logical value indicating whether the resulting derivative should be a periodic function.

Details

This command performs numerical differentiation on the function values in a function value table (object of class "fv"). The differentiation is performed either by smooth.spline or by a naive numerical difference algorithm.

The command deriv is generic. This is the method for objects of class "fv".

Differentiation is applied to every column (or to each of the selected columns) of function values in turn, using the function argument as the x coordinate and the selected column as the y coordinate. The original function values are then replaced by the corresponding derivatives.

The optional argument which specifies which of the columns of function values in expr will be differentiated. The default (indicated by the wildcard which="*") is to differentiate all function values, i.e.\ all columns except the function argument. Alternatively which="." designates the subset of function values that are displayed in the default plot. Alternatively which can be a character vector containing the names of columns of expr.

If the argument kinks is given, it should be a numeric vector giving the discontinuity points of the function: the value or values of the function argument at which the function is not differentiable. Differentiation will be performed separately on intervals between the discontinuity points.

If periodic=TRUE then the function expr is taken to be periodic, with period equal to the range of the function argument in expr. The resulting derivative is periodic.

If periodic=FALSE but Dperiodic=TRUE, then the *derivative* is assumed to be periodic. This would be appropriate if expr is the cumulative distribution function of an angular variable, for example.

Value

Another function value table (object of class "fv") of the same format.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

with.fv, fv.object, smooth.spline

Examples

```
G <- Gest(cells)
plot(deriv(G, which=".", spar=0.5))
A <- pairorient(redwood, 0.05, 0.15)
DA <- deriv(A, spar=0.6, Dperiodic=TRUE)
```

---

dg.envelope *Global Envelopes for Dao-Genton Test*

Description

Computes the global envelopes corresponding to the Dao-Genton test of goodness-of-fit.

Usage

```
dg.envelope(X, ...,
            nsim = 19, nsimsub=nsim-1, nrank = 1,
            alternative=c("two.sided", "less", "greater"),
            leaveout=1, interpolate = FALSE,
            savefuns=FALSE, savepatterns=FALSE,
            verbose = TRUE)
```

Arguments

X	Either a point pattern dataset (object of class "ppp", "lpp" or "pp3") or a fitted point process model (object of class "ppm", "kppm" or "slrm").
...	Arguments passed to mad.test or envelope to control the conduct of the test. Useful arguments include fun to determine the summary function, rinterval to determine the range of r values used in the test, and verbose=FALSE to turn off the messages.
nsim	Number of simulated patterns to be generated in the primary experiment.
nsimsub	Number of simulations in each basic test. There will be nsim repetitions of the basic test, each involving nsimsub simulated realisations, so there will be a total of nsim * (nsimsub + 1) simulations.
nrank	Integer. Rank of the envelope value amongst the nsim simulated values. A rank of 1 means that the minimum and maximum simulated values will be used.
alternative	Character string determining whether the envelope corresponds to a two-sided test (alternative="two.sided", the default) or a one-sided test with a lower critical boundary (alternative="less") or a one-sided test with an upper critical boundary (alternative="greater").

leaveout	Optional integer 0, 1 or 2 indicating how to calculate the deviation between the observed summary function and the nominal reference value, when the reference value must be estimated by simulation. See Details.
interpolate	Logical value indicating whether to interpolate the distribution of the test statistic by kernel smoothing, as described in Dao and Genton (2014, Section 5).
savefuns	Logical flag indicating whether to save the simulated function values (from the first stage).
savepatterns	Logical flag indicating whether to save the simulated point patterns (from the first stage).
verbose	Logical value determining whether to print progress reports.

Details

Computes global simulation envelopes corresponding to the Dao-Genton (2014) adjusted Monte Carlo goodness-of-fit test. The envelopes were developed in Baddeley et al (2015) and described in Baddeley, Rubak and Turner (2015).

If X is a point pattern, the null hypothesis is CSR.

If X is a fitted model, the null hypothesis is that model.

The Dao-Genton test is biased when the significance level is very small (small p-values are not reliable) and we recommend bits.envelope in this case.

Value

An object of class "fv".

Author(s)

Adrian Baddeley, Andrew Hardegen, Tom Lawrence, Robin Milne, Gopalan Nair and Suman Rakshit. Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

Dao, N.A. and Genton, M. (2014) A Monte Carlo adjusted goodness-of-fit test for parametric models describing spatial point patterns. *Journal of Graphical and Computational Statistics* **23**, 497–517.

Baddeley, A., Hardegen, A., Lawrence, L., Milne, R.K., Nair, G.M. and Rakshit, S. (2015) Pushing the envelope: extensions of graphical Monte Carlo tests. Unpublished manuscript.

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. Chapman and Hall/CRC Press.

See Also

dg.test, mad.test, envelope

Examples

```
ns <- if(interactive()) 19 else 4
E <- dg.envelope(swedishpines, Lest, nsim=ns)
E
plot(E)
Eo <- dg.envelope(swedishpines, Lest, alternative="less", nsim=ns)
Ei <- dg.envelope(swedishpines, Lest, interpolate=TRUE, nsim=ns)
```

`dg.progress` *Progress Plot of Dao-Genton Test of Spatial Pattern*

Description

Generates a progress plot (envelope representation) of the Dao-Genton test for a spatial point pattern.

Usage

```
dg.progress(X, fun = Lest, ...,
            exponent = 2, nsim = 19, nsimsub = nsim - 1,
            nrank = 1, alpha, leaveout=1, interpolate = FALSE, rmin=0,
            savefuns = FALSE, savepatterns = FALSE, verbose=TRUE)
```

Arguments

`X`	Either a point pattern (object of class `"ppp"`, `"lpp"` or other class), a fitted point process model (object of class `"ppm"`, `"kppm"` or other class) or an envelope object (class `"envelope"`).
`fun`	Function that computes the desired summary statistic for a point pattern.
`...`	Arguments passed to `envelope`. Useful arguments include `alternative` to specify one-sided or two-sided envelopes.
`exponent`	Positive number. The exponent of the L^p distance. See Details.
`nsim`	Number of repetitions of the basic test.
`nsimsub`	Number of simulations in each basic test. There will be `nsim` repetitions of the basic test, each involving `nsimsub` simulated realisations, so there will be a total of `nsim * (nsimsub + 1)` simulations.
`nrank`	Integer. The rank of the critical value of the Monte Carlo test, amongst the `nsim` simulated values. A rank of 1 means that the minimum and maximum simulated values will become the critical values for the test.
`alpha`	Optional. The significance level of the test. Equivalent to `nrank/(nsim+1)` where `nsim` is the number of simulations.
`leaveout`	Optional integer 0, 1 or 2 indicating how to calculate the deviation between the observed summary function and the nominal reference value, when the reference value must be estimated by simulation. See Details.

interpolate	Logical value indicating how to compute the critical value. If interpolate=FALSE (the default), a standard Monte Carlo test is performed, and the critical value is the largest simulated value of the test statistic (if nrank=1) or the nrank-th largest (if nrank is another number). If interpolate=TRUE, kernel density estimation is applied to the simulated values, and the critical value is the upper alpha quantile of this estimated distribution.
rmin	Optional. Left endpoint for the interval of r values on which the test statistic is calculated.
savefuns	Logical value indicating whether to save the simulated function values (from the first stage).
savepatterns	Logical value indicating whether to save the simulated point patterns (from the first stage).
verbose	Logical value indicating whether to print progress reports.

Details

The Dao and Genton (2014) test for a spatial point pattern is described in dg.test. This test depends on the choice of an interval of distance values (the argument rinterval). A *progress plot* or *envelope representation* of the test (Baddeley et al, 2014, 2015; Baddeley, Rubak and Turner, 2015) is a plot of the test statistic (and the corresponding critical value) against the length of the interval rinterval.

The command dg.progress effectively performs dg.test on X using all possible intervals of the form $[0, R]$, and returns the resulting values of the test statistic, and the corresponding critical values of the test, as a function of R.

The result is an object of class "fv" that can be plotted to obtain the progress plot. The display shows the test statistic (solid black line) and the test acceptance region (grey shading). If X is an envelope object, then some of the data stored in X may be re-used:

- If X is an envelope object containing simulated functions, and fun=NULL, then the code will re-use the simulated functions stored in X.
- If X is an envelope object containing simulated point patterns, then fun will be applied to the stored point patterns to obtain the simulated functions. If fun is not specified, it defaults to Lest.
- Otherwise, new simulations will be performed, and fun defaults to Lest.

If the argument rmin is given, it specifies the left endpoint of the interval defining the test statistic: the tests are performed using intervals $[r_{\min}, R]$ where $R \geq r_{\min}$.

The argument leaveout specifies how to calculate the discrepancy between the summary function for the data and the nominal reference value, when the reference value must be estimated by simulation. The values leaveout=0 and leaveout=1 are both algebraically equivalent (Baddeley et al, 2014, Appendix) to computing the difference observed - reference where the reference is the mean of simulated values. The value leaveout=2 gives the leave-two-out discrepancy proposed by Dao and Genton (2014).

Value

An object of class "fv" that can be plotted to obtain the progress plot.

Author(s)

Adrian Baddeley, Andrew Hardegen, Tom Lawrence, Robin Milne, Gopalan Nair and Suman Rakshit. Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A., Diggle, P., Hardegen, A., Lawrence, T., Milne, R. and Nair, G. (2014) On tests of spatial pattern based on simulation envelopes. *Ecological Monographs* **84** (3) 477–489.

Baddeley, A., Hardegen, A., Lawrence, L., Milne, R.K., Nair, G.M. and Rakshit, S. (2015) Pushing the envelope: extensions of graphical Monte Carlo tests. Unpublished manuscript.

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. Chapman and Hall/CRC Press.

Dao, N.A. and Genton, M. (2014) A Monte Carlo adjusted goodness-of-fit test for parametric models describing spatial point patterns. *Journal of Graphical and Computational Statistics* **23**, 497–517.

See Also

dg.test, dclf.progress

Examples

```
ns <- if(interactive()) 19 else 5
plot(dg.progress(cells, nsim=ns))
```

---

dg.sigtrace *Significance Trace of Dao-Genton Test*

Description

Generates a Significance Trace of the Dao and Genton (2014) test for a spatial point pattern.

Usage

```
dg.sigtrace(X, fun = Lest, ...,
            exponent = 2, nsim = 19, nsimsub = nsim - 1,
            alternative = c("two.sided", "less", "greater"),
            rmin=0, leaveout=1,
            interpolate = FALSE, confint = TRUE, alpha = 0.05,
            savefuns=FALSE, savepatterns=FALSE, verbose=FALSE)
```

Arguments

X	Either a point pattern (object of class "ppp", "lpp" or other class), a fitted point process model (object of class "ppm", "kppm" or other class) or an envelope object (class "envelope").
fun	Function that computes the desired summary statistic for a point pattern.
...	Arguments passed to envelope.
exponent	Positive number. Exponent used in the test statistic. Use exponent=2 for the Diggle-Cressie-Loosmore-Ford test, and exponent=Inf for the Maximum Absolute Deviation test. See Details.
nsim	Number of repetitions of the basic test.
nsimsub	Number of simulations in each basic test. There will be nsim repetitions of the basic test, each involving nsimsub simulated realisations, so there will be a total of nsim * (nsimsub + 1) simulations.
alternative	Character string specifying the alternative hypothesis. The default (alternative="two.sided") is that the true value of the summary function is not equal to the theoretical value postulated under the null hypothesis. If alternative="less" the alternative hypothesis is that the true value of the summary function is lower than the theoretical value.
rmin	Optional. Left endpoint for the interval of r values on which the test statistic is calculated.
leaveout	Optional integer 0, 1 or 2 indicating how to calculate the deviation between the observed summary function and the nominal reference value, when the reference value must be estimated by simulation. See Details.
interpolate	Logical value indicating whether to interpolate the distribution of the test statistic by kernel smoothing, as described in Dao and Genton (2014, Section 5).
confint	Logical value indicating whether to compute a confidence interval for the 'true' p-value.
alpha	Significance level to be plotted (this has no effect on the calculation but is simply plotted as a reference value).
savefuns	Logical flag indicating whether to save the simulated function values (from the first stage).
savepatterns	Logical flag indicating whether to save the simulated point patterns (from the first stage).
verbose	Logical flag indicating whether to print progress reports.

Details

The Dao and Genton (2014) test for a spatial point pattern is described in dg.test. This test depends on the choice of an interval of distance values (the argument rinterval). A *significance trace* (Bowman and Azzalini, 1997; Baddeley et al, 2014, 2015; Baddeley, Rubak and Turner, 2015) of the test is a plot of the p-value obtained from the test against the length of the interval rinterval.

The command dg.sigtrace effectively performs dg.test on X using all possible intervals of the form $[0, R]$, and returns the resulting p-values as a function of R.

The result is an object of class "fv" that can be plotted to obtain the significance trace. The plot shows the Dao-Genton adjusted p-value (solid black line), the critical value 0.05 (dashed red line), and a pointwise 95% confidence band (grey shading) for the 'true' (Neyman-Pearson) p-value. The confidence band is based on the Agresti-Coull (1998) confidence interval for a binomial proportion.

If X is an envelope object and fun=NULL then the code will re-use the simulated functions stored in X.

If the argument rmin is given, it specifies the left endpoint of the interval defining the test statistic: the tests are performed using intervals $[r_{\min}, R]$ where $R \geq r_{\min}$.

The argument leaveout specifies how to calculate the discrepancy between the summary function for the data and the nominal reference value, when the reference value must be estimated by simulation. The values leaveout=0 and leaveout=1 are both algebraically equivalent (Baddeley et al, 2014, Appendix) to computing the difference observed - reference where the reference is the mean of simulated values. The value leaveout=2 gives the leave-two-out discrepancy proposed by Dao and Genton (2014).

Value

An object of class "fv" that can be plotted to obtain the significance trace.

Author(s)

Adrian Baddeley, Andrew Hardegen, Tom Lawrence, Robin Milne, Gopalan Nair and Suman Rakshit. Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

Agresti, A. and Coull, B.A. (1998) Approximate is better than "Exact" for interval estimation of binomial proportions. *American Statistician* **52**, 119–126.

Baddeley, A., Diggle, P., Hardegen, A., Lawrence, T., Milne, R. and Nair, G. (2014) On tests of spatial pattern based on simulation envelopes. *Ecological Monographs* **84**(3) 477–489.

Baddeley, A., Hardegen, A., Lawrence, L., Milne, R.K., Nair, G.M. and Rakshit, S. (2015) Pushing the envelope: extensions of graphical Monte Carlo tests. Unpublished manuscript.

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. Chapman and Hall/CRC Press.

Bowman, A.W. and Azzalini, A. (1997) *Applied smoothing techniques for data analysis: the kernel approach with S-Plus illustrations*. Oxford University Press, Oxford.

Dao, N.A. and Genton, M. (2014) A Monte Carlo adjusted goodness-of-fit test for parametric models describing spatial point patterns. *Journal of Graphical and Computational Statistics* **23**, 497–517.

See Also

dg.test for the Dao-Genton test, dclf.sigtrace for significance traces of other tests.

Examples

```
ns <- if(interactive()) 19 else 5
plot(dg.sigtrace(cells, nsim=ns))
```

`dg.test` *Dao-Genton Adjusted Goodness-Of-Fit Test*

Description

Performs the Dao and Genton (2014) adjusted goodness-of-fit test of spatial pattern.

Usage

```
dg.test(X, ...,
        exponent = 2, nsim=19, nsimsub=nsim-1,
        alternative=c("two.sided", "less", "greater"),
        reuse = TRUE, leaveout=1, interpolate = FALSE,
        savefuns=FALSE, savepatterns=FALSE,
        verbose = TRUE)
```

Arguments

`X`	Either a point pattern dataset (object of class `"ppp"`, `"lpp"` or `"pp3"`) or a fitted point process model (object of class `"ppm"`, `"kppm"`, `"lppm"` or `"slrm"`).
`...`	Arguments passed to `dclf.test` or `mad.test` or `envelope` to control the conduct of the test. Useful arguments include `fun` to determine the summary function, `rinterval` to determine the range of r values used in the test, and `use.theory` described under Details.
`exponent`	Exponent used in the test statistic. Use `exponent=2` for the Diggle-Cressie-Loosmore-Ford test, and `exponent=Inf` for the Maximum Absolute Deviation test.
`nsim`	Number of repetitions of the basic test.
`nsimsub`	Number of simulations in each basic test. There will be `nsim` repetitions of the basic test, each involving `nsimsub` simulated realisations, so there will be a total of `nsim * (nsimsub + 1)` simulations.
`alternative`	Character string specifying the alternative hypothesis. The default (`alternative="two.sided"`) is that the true value of the summary function is not equal to the theoretical value postulated under the null hypothesis. If `alternative="less"` the alternative hypothesis is that the true value of the summary function is lower than the theoretical value.
`reuse`	Logical value indicating whether to re-use the first stage simulations at the second stage, as described by Dao and Genton (2014).
`leaveout`	Optional integer 0, 1 or 2 indicating how to calculate the deviation between the observed summary function and the nominal reference value, when the reference value must be estimated by simulation. See Details.

interpolate	Logical value indicating whether to interpolate the distribution of the test statistic by kernel smoothing, as described in Dao and Genton (2014, Section 5).
savefuns	Logical flag indicating whether to save the simulated function values (from the first stage).
savepatterns	Logical flag indicating whether to save the simulated point patterns (from the first stage).
verbose	Logical value indicating whether to print progress reports.

Details

Performs the Dao-Genton (2014) adjusted Monte Carlo goodness-of-fit test, in the equivalent form described by Baddeley et al (2014).

If X is a point pattern, the null hypothesis is CSR.

If X is a fitted model, the null hypothesis is that model.

The argument use.theory passed to envelope determines whether to compare the summary function for the data to its theoretical value for CSR (use.theory=TRUE) or to the sample mean of simulations from CSR (use.theory=FALSE).

The argument leaveout specifies how to calculate the discrepancy between the summary function for the data and the nominal reference value, when the reference value must be estimated by simulation. The values leaveout=0 and leaveout=1 are both algebraically equivalent (Baddeley et al, 2014, Appendix) to computing the difference observed - reference where the reference is the mean of simulated values. The value leaveout=2 gives the leave-two-out discrepancy proposed by Dao and Genton (2014).

The Dao-Genton test is biased when the significance level is very small (small p-values are not reliable) and we recommend bits.test in this case.

Value

A hypothesis test (object of class "htest" which can be printed to show the outcome of the test.

Author(s)

Adrian Baddeley, Andrew Hardegen, Tom Lawrence, Robin Milne, Gopalan Nair and Suman Rakshit. Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

Dao, N.A. and Genton, M. (2014) A Monte Carlo adjusted goodness-of-fit test for parametric models describing spatial point patterns. *Journal of Graphical and Computational Statistics* **23**, 497–517.

Baddeley, A., Diggle, P.J., Hardegen, A., Lawrence, T., Milne, R.K. and Nair, G. (2014) On tests of spatial pattern based on simulation envelopes. *Ecological Monographs* **84** (3) 477–489.

Baddeley, A., Hardegen, A., Lawrence, L., Milne, R.K., Nair, G.M. and Rakshit, S. (2017) On two-stage Monte Carlo tests of composite hypotheses. *Computational Statistics and Data Analysis* **114**, 75–87.

See Also

bits.test, dclf.test, mad.test

Examples

```
ns <- if(interactive()) 19 else 4
dg.test(cells, nsim=ns)
dg.test(cells, alternative="less", nsim=ns)
dg.test(cells, nsim=ns, interpolate=TRUE)
```

---

dimhat *Estimate Dimension of Central Subspace*

Description

Given the kernel matrix that characterises a central subspace, this function estimates the dimension of the subspace.

Usage

```
dimhat(M)
```

Arguments

M	Kernel of subspace. A symmetric, non-negative definite, numeric matrix, typically obtained from sdr.

Details

This function computes the maximum descent estimate of the dimension of the central subspace with a given kernel matrix M.

The matrix M should be the kernel matrix of a central subspace, which can be obtained from sdr. It must be a symmetric, non-negative-definite, numeric matrix.

The algorithm finds the eigenvalues $\lambda_1 \geq \ldots \geq \lambda_n$ of M, and then determines the index k for which λ_k/λ_{k-1} is greatest.

Value

A single integer giving the estimated dimension.

Author(s)

Matlab original by Yongtao Guan, translated to R by Suman Rakshit.

References

Guan, Y. and Wang, H. (2010) Sufficient dimension reduction for spatial point processes directed by Gaussian random fields. *Journal of the Royal Statistical Society, Series B*, **72**, 367–387.

See Also

sdr, subspaceDistance

---

distcdf *Distribution Function of Interpoint Distance*

Description

Computes the cumulative distribution function of the distance between two independent random points in a given window or windows.

Usage

```
distcdf(W, V=W, ..., dW=1, dV=dW, nr=1024,
        regularise=TRUE, savedenom=FALSE, delta=NULL)
```

Arguments

W	A window (object of class "owin") containing the first random point.
V	Optional. Another window containing the second random point. Defaults to W.
...	Arguments passed to as.mask to determine the pixel resolution for the calculation.
dV, dW	Optional. Probability densities (not necessarily normalised) for the first and second random points respectively. Data in any format acceptable to as.im, for example, a function(x,y) or a pixel image or a numeric value. The default corresponds to a uniform distribution over the window.
nr	Integer. The number of values of interpoint distance r for which the CDF will be computed. Should be a large value. Alternatively if nr=NULL, a good default value will be chosen, depending on the pixel resolution.
regularise	Logical value indicating whether to smooth the results for very small distances, to avoid discretisation artefacts.
savedenom	Logical value indicating whether to save the denominator of the double integral as an attribute of the result.
delta	Optional. A positive number. The maximum permitted spacing between values of the function argument.

Details

This command computes the Cumulative Distribution Function $CDF(r) = Prob(T \leq r)$ of the Euclidean distance $T = \|X_1 - X_2\|$ between two independent random points X_1 and X_2.

In the simplest case, the command distcdf(W), the random points are assumed to be uniformly distributed in the same window W.

Alternatively the two random points may be uniformly distributed in two different windows W and V.

In the most general case the first point X_1 is random in the window W with a probability density proportional to dW, and the second point X_2 is random in a different window V with probability density proportional to dV. The values of dW and dV must be finite and nonnegative.

The calculation is performed by numerical integration of the set covariance function setcov for uniformly distributed points, and by computing the covariance function imcov in the general case. The accuracy of the result depends on the pixel resolution used to represent the windows: this is controlled by the arguments ... which are passed to as.mask. For example use eps=0.1 to specify pixels of size 0.1 units.

The arguments W or V may also be point patterns (objects of class "ppp"). The result is the cumulative distribution function of the distance from a randomly selected point in the point pattern, to a randomly selected point in the other point pattern or window.

If regularise=TRUE (the default), values of the cumulative distribution function for very short distances are smoothed to avoid discretisation artefacts. Smoothing is applied to all distances shorter than the width of 10 pixels.

Numerical accuracy of some calculations requires very fine spacing of the values of the function argument r. If the argument delta is given, then after the cumulative distribution function has been calculated, it will be interpolated onto a finer grid of r values with spacing less than or equal to delta.

Value

An object of class "fv", see fv.object, which can be plotted directly using plot.fv.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

setcov, as.mask.

Examples

```
# The unit disc
B <- disc()
plot(distcdf(B))
```

---

domain.quadrattest *Extract the Domain of any Spatial Object*

Description

Given a spatial object such as a point pattern, in any number of dimensions, this function extracts the spatial domain in which the object is defined.

Usage

```
 ## S3 method for class 'quadrattest'
domain(X, ...)
```

Arguments

X	A spatial object such as a point pattern (in any number of dimensions), line segment pattern or pixel image.
...	Extra arguments. They are ignored by all the methods listed here.

Details

The function domain is generic.

For a spatial object X in any number of dimensions, domain(X) extracts the spatial domain in which X is defined.

For a two-dimensional object X, typically domain(X) is the same as Window(X).

Exceptions occur for methods related to linear networks.

Value

A spatial object representing the domain of X. Typically a window (object of class "owin"), a three-dimensional box ("box3"), a multidimensional box ("boxx") or a linear network ("linnet").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

domain, domain.quadratcount, domain.ppm, domain.rmhmodel, domain.lpp. Window, Frame.

Examples

```
domain(quadrat.test(redwood, 2, 2))
```

edge.Ripley *Ripley's Isotropic Edge Correction*

Description

Computes Ripley's isotropic edge correction weights for a point pattern.

Usage

```
edge.Ripley(X, r, W = Window(X), method = c("C", "interpreted"),
            maxweight = 100, internal=list())

rmax.Ripley(W)
```

Arguments

X	Point pattern (object of class "ppp").
W	Window for which the edge correction is required.
r	Vector or matrix of interpoint distances for which the edge correction should be computed.
method	Choice of algorithm. Either "interpreted" or "C". This is needed only for debugging purposes.
maxweight	Maximum permitted value of the edge correction weight.
internal	For developer use only.

Details

The function edge.Ripley computes Ripley's (1977) isotropic edge correction weight, which is used in estimating the K function and in many other contexts.

The function rmax.Ripley computes the maximum value of distance r for which the isotropic edge correction estimate of $K(r)$ is valid.

For a single point x in a window W, and a distance $r > 0$, the isotropic edge correction weight is

$$e(u, r) = \frac{2\pi r}{\text{length}(c(u, r) \cap W)}$$

where $c(u, r)$ is the circle of radius r centred at the point u. The denominator is the length of the overlap between this circle and the window W.

The function edge.Ripley computes this edge correction weight for each point in the point pattern X and for each corresponding distance value in the vector or matrix r.

If r is a vector, with one entry for each point in X, then the result is a vector containing the edge correction weights e(X[i], r[i]) for each i.

If r is a matrix, with one row for each point in X, then the result is a matrix whose i,j entry gives the edge correction weight e(X[i], r[i,j]). For example edge.Ripley(X, pairdist(X)) computes all the edge corrections required for the K-function.

If any value of the edge correction weight exceeds `maxwt`, it is set to `maxwt`.

The function `rmax.Ripley` computes the smallest distance r such that it is possible to draw a circle of radius r, centred at a point of `W`, such that the circle does not intersect the interior of `W`.

Value

A numeric vector or matrix.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

References

Ripley, B.D. (1977) Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, Series B*, **39**, 172 – 212.

See Also

`edge.Trans`, `rmax.Trans`, `Kest`

Examples

```
v <- edge.Ripley(cells, pairdist(cells))

rmax.Ripley(Window(cells))
```

---

edge.Trans *Translation Edge Correction*

Description

Computes Ohser and Stoyan's translation edge correction weights for a point pattern.

Usage

```
edge.Trans(X, Y = X, W = Window(X),
      exact = FALSE, paired = FALSE,
      ...,
      trim = spatstat.options("maxedgewt"),
      dx=NULL, dy=NULL,
      give.rmax=FALSE, gW=NULL)

rmax.Trans(W, g=setcov(W))
```

Arguments

X, Y	Point patterns (objects of class "ppp").
W	Window for which the edge correction is required.
exact	Logical. If TRUE, a slow algorithm will be used to compute the exact value. If FALSE, a fast algorithm will be used to compute the approximate value.
paired	Logical value indicating whether X and Y are paired. If TRUE, compute the edge correction for corresponding points X[i], Y[i] for all i. If FALSE, compute the edge correction for each possible pair of points X[i], Y[j] for all i and j.
...	Ignored.
trim	Maximum permitted value of the edge correction weight.
dx, dy	Alternative data giving the x and y coordinates of the vector differences between the points. Incompatible with X and Y. See Details.
give.rmax	Logical. If TRUE, also compute the value of rmax.Trans(W) and return it as an attribute of the result.
g, gW	Optional. Set covariance of W, if it has already been computed. Not required if W is a rectangle.

Details

The function edge.Trans computes Ohser and Stoyan's translation edge correction weight, which is used in estimating the K function and in many other contexts.

The function rmax.Trans computes the maximum value of distance r for which the translation edge correction estimate of $K(r)$ is valid.

For a pair of points x and y in a window W, the translation edge correction weight is

$$e(u, r) = \frac{\mathrm{area}(W)}{\mathrm{area}(W \cap (W + y - x))}$$

where $W + y - x$ is the result of shifting the window W by the vector $y - x$. The denominator is the area of the overlap between this shifted window and the original window.

The function edge.Trans computes this edge correction weight. If paired=TRUE, then X and Y should contain the same number of points. The result is a vector containing the edge correction weights e(X[i], Y[i]) for each i.

If paired=FALSE, then the result is a matrix whose i,j entry gives the edge correction weight e(X[i], Y[j]).

Computation is exact if the window is a rectangle. Otherwise,

- if exact=TRUE, the edge correction weights are computed exactly using overlap.owin, which can be quite slow.
- if exact=FALSE (the default), the weights are computed rapidly by evaluating the set covariance function setcov using the Fast Fourier Transform.

If any value of the edge correction weight exceeds trim, it is set to trim.

The arguments dx and dy can be provided as an alternative to X and Y. If paired=TRUE then dx,dy should be vectors of equal length such that the vector difference of the ith pair is c(dx[i], dy[i]).

If paired=FALSE then dx,dy should be matrices of the same dimensions, such that the vector difference between X[i] and Y[j] is c(dx[i,j], dy[i,j]). The argument W is needed.

The value of rmax.Trans is the shortest distance from the origin $(0,0)$ to the boundary of the support of the set covariance function of W. It is computed by pixel approximation using setcov, unless W is a rectangle, when rmax.Trans(W) is the length of the shortest side of the rectangle.

Value

Numeric vector or matrix.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>.

References

Ohser, J. (1983) On estimators for the reduced second moment measure of point processes. *Mathematische Operationsforschung und Statistik, series Statistics*, **14**, 63 – 71.

See Also

rmax.Trans, edge.Ripley, setcov, Kest

Examples

```
v <- edge.Trans(cells)
rmax.Trans(Window(cells))
```

Emark *Diagnostics for random marking*

Description

Estimate the summary functions $E(r)$ and $V(r)$ for a marked point pattern, proposed by Schlather et al (2004) as diagnostics for dependence between the points and the marks.

Usage

```
Emark(X, r=NULL,
         correction=c("isotropic", "Ripley", "translate"),
         method="density", ..., normalise=FALSE)
Vmark(X, r=NULL,
         correction=c("isotropic", "Ripley", "translate"),
         method="density", ..., normalise=FALSE)
```

Arguments

X	The observed point pattern. An object of class "ppp" or something acceptable to as.ppp. The pattern should have numeric marks.
r	Optional. Numeric vector. The values of the argument r at which the function $E(r)$ or $V(r)$ should be evaluated. There is a sensible default.
correction	A character vector containing any selection of the options "isotropic", "Ripley" or "translate". It specifies the edge correction(s) to be applied.
method	A character vector indicating the user's choice of density estimation technique to be used. Options are "density", "loess", "sm" and "smrep".
...	Arguments passed to the density estimation routine (density, loess or sm.density) selected by method.
normalise	IfTRUE, normalise the estimate of $E(r)$ or $V(r)$ so that it would have value equal to 1 if the marks are independent of the points.

Details

For a marked point process, Schlather et al (2004) defined the functions $E(r)$ and $V(r)$ to be the conditional mean and conditional variance of the mark attached to a typical random point, given that there exists another random point at a distance r away from it.

More formally,

$$E(r) = E_{0u}[M(0)]$$

and

$$V(r) = E_{0u}[(M(0) - E(u))^2]$$

where E_{0u} denotes the conditional expectation given that there are points of the process at the locations 0 and u separated by a distance r, and where $M(0)$ denotes the mark attached to the point 0.

These functions may serve as diagnostics for dependence between the points and the marks. If the points and marks are independent, then $E(r)$ and $V(r)$ should be constant (not depending on r). See Schlather et al (2004).

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to as.ppp. It must be a marked point pattern with numeric marks.

The argument r is the vector of values for the distance r at which $k_f(r)$ is estimated.

This algorithm assumes that X can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in X as Window(X)) may have arbitrary shape.

Biases due to edge effects are treated in the same manner as in Kest. The edge corrections implemented here are

isotropic/Ripley Ripley's isotropic correction (see Ripley, 1988; Ohser, 1983). This is implemented only for rectangular and polygonal windows (not for binary masks).

translate Translation correction (Ohser, 1983). Implemented for all window geometries, but slow for complex windows.

Note that the estimator assumes the process is stationary (spatially homogeneous).

The numerator and denominator of the mark correlation function (in the expression above) are estimated using density estimation techniques. The user can choose between

`"density"` which uses the standard kernel density estimation routine `density`, and works only for evenly-spaced `r` values;

`"loess"` which uses the function `loess` in the package **modreg**;

`"sm"` which uses the function `sm.density` in the package **sm** and is extremely slow;

`"smrep"` which uses the function `sm.density` in the package **sm** and is relatively fast, but may require manual control of the smoothing parameter `hmult`.

Value

If `marks(X)` is a numeric vector, the result is an object of class `"fv"` (see `fv.object`). If `marks(X)` is a data frame, the result is a list of objects of class `"fv"`, one for each column of marks.

An object of class `"fv"` is essentially a data frame containing numeric columns

`r`	the values of the argument r at which the function $E(r)$ or $V(r)$ has been estimated
`theo`	the theoretical, constant value of $E(r)$ or $V(r)$ when the marks attached to different points are independent

together with a column or columns named `"iso"` and/or `"trans"`, according to the selected edge corrections. These columns contain estimates of the function $E(r)$ or $V(r)$ obtained by the edge corrections named.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

References

Schlather, M. and Ribeiro, P. and Diggle, P. (2004) Detecting dependence between marks and locations of marked point processes. *Journal of the Royal Statistical Society, series B* **66** (2004) 79-83.

See Also

Mark correlation `markcorr`, mark variogram `markvario` for numeric marks.

Mark connection function `markconnect` and multitype K-functions `Kcross`, `Kdot` for factor-valued marks.

Examples

```
plot(Emark(spruces))
E <- Emark(spruces, method="density", kernel="epanechnikov")
plot(Vmark(spruces))

plot(Emark(finpines))
V <- Vmark(finpines)
```

envelope *Simulation Envelopes of Summary Function*

Description

Computes simulation envelopes of a summary function.

Usage

```
  envelope(Y, fun, ...)

  ## S3 method for class 'ppp'
envelope(Y, fun=Kest, nsim=99, nrank=1, ...,
  funargs=list(), funYargs=funargs,
  simulate=NULL,  fix.n=FALSE, fix.marks=FALSE,
  verbose=TRUE, clipdata=TRUE,
  transform=NULL, global=FALSE, ginterval=NULL, use.theory=NULL,
  alternative=c("two.sided", "less", "greater"),
  scale=NULL, clamp=FALSE,
  savefuns=FALSE, savepatterns=FALSE,
  nsim2=nsim, VARIANCE=FALSE, nSD=2, Yname=NULL,
  maxnerr=nsim, rejectNA=FALSE, silent=FALSE,
  do.pwrong=FALSE, envir.simul=NULL)
```

Arguments

Y	Object containing point pattern data. A point pattern (object of class "ppp") or a fitted point process model (object of class "ppm", "kppm" or "slrm").
fun	Function that computes the desired summary statistic for a point pattern.
nsim	Number of simulated point patterns to be generated when computing the envelopes.
nrank	Integer. Rank of the envelope value amongst the nsim simulated values. A rank of 1 means that the minimum and maximum simulated values will be used.
...	Extra arguments passed to fun.
funargs	A list, containing extra arguments to be passed to fun.
funYargs	Optional. A list, containing extra arguments to be passed to fun when applied to the original data Y only.
simulate	Optional. Specifies how to generate the simulated point patterns. If simulate is an expression in the R language, then this expression will be evaluated nsim times, to obtain nsim point patterns which are taken as the simulated patterns from which the envelopes are computed. If simulate is a function, then this function will be repeatedly applied to the data pattern Y to obtain nsim simulated patterns. If simulate is a list of point patterns, then the entries in this list will be treated as the simulated patterns from which the envelopes are computed.

	Alternatively `simulate` may be an object produced by the `envelope` command: see Details.
`fix.n`	Logical. If `TRUE`, simulated patterns will have the same number of points as the original data pattern. This option is currently not available for `envelope.kppm`.
`fix.marks`	Logical. If `TRUE`, simulated patterns will have the same number of points *and* the same marks as the original data pattern. In a multitype point pattern this means that the simulated patterns will have the same number of points *of each type* as the original data. This option is currently not available for `envelope.kppm`.
`verbose`	Logical flag indicating whether to print progress reports during the simulations.
`clipdata`	Logical flag indicating whether the data point pattern should be clipped to the same window as the simulated patterns, before the summary function for the data is computed. This should usually be `TRUE` to ensure that the data and simulations are properly comparable.
`transform`	Optional. A transformation to be applied to the function values, before the envelopes are computed. An expression object (see Details).
`global`	Logical flag indicating whether envelopes should be pointwise (`global=FALSE`) or simultaneous (`global=TRUE`).
`ginterval`	Optional. A vector of length 2 specifying the interval of r values for the simultaneous critical envelopes. Only relevant if `global=TRUE`.
`use.theory`	Logical value indicating whether to use the theoretical value, computed by `fun`, as the reference value for simultaneous envelopes. Applicable only when `global=TRUE`. Default is `use.theory=TRUE` if `Y` is a point pattern, or a point process model equivalent to Complete Spatial Randomness, and `use.theory=FALSE` otherwise.
`alternative`	Character string determining whether the envelope corresponds to a two-sided test (`side="two.sided"`, the default) or a one-sided test with a lower critical boundary (`side="less"`) or a one-sided test with an upper critical boundary (`side="greater"`).
`scale`	Optional. Scaling function for global envelopes. A function in the R language which determines the relative scale of deviations, as a function of distance r, when computing the global envelopes. Applicable only when `global=TRUE`. Summary function values for distance `r` will be *divided* by `scale(r)` before the maximum deviation is computed. The resulting global envelopes will have width proportional to `scale(r)`.
`clamp`	Logical value indicating how to compute envelopes when `alternative="less"` or `alternative="greater"`. Deviations of the observed summary function from the theoretical summary function are initially evaluated as signed real numbers, with large positive values indicating consistency with the alternative hypothesis. If `clamp=FALSE` (the default), these values are not changed. If `clamp=TRUE`, any negative values are replaced by zero.
`savefuns`	Logical flag indicating whether to save all the simulated function values.
`savepatterns`	Logical flag indicating whether to save all the simulated point patterns.
`nsim2`	Number of extra simulated point patterns to be generated if it is necessary to use simulation to estimate the theoretical mean of the summary function. Only relevant when `global=TRUE` and the simulations are not based on CSR.

VARIANCE	Logical. If TRUE, critical envelopes will be calculated as sample mean plus or minus nSD times sample standard deviation.
nSD	Number of estimated standard deviations used to determine the critical envelopes, if VARIANCE=TRUE.
Yname	Character string that should be used as the name of the data point pattern Y when printing or plotting the results.
maxnerr	Maximum number of rejected patterns. If fun yields a fatal error when applied to a simulated point pattern (for example, because the pattern is empty and fun requires at least one point), the pattern will be rejected and a new random point pattern will be generated. If this happens more than maxnerr times, the algorithm will give up.
rejectNA	Logical value specifying whether to reject a simulated pattern if the resulting values of fun are all equal to NA, NaN or infinite. If FALSE (the default), then simulated patterns are only rejected when fun gives a fatal error.
silent	Logical value specifying whether to print a report each time a simulated pattern is rejected.
do.pwrong	Logical. If TRUE, the algorithm will also estimate the true significance level of the "wrong" test (the test that declares the summary function for the data to be significant if it lies outside the *pointwise* critical boundary at any point). This estimate is printed when the result is printed.
envir.simul	Environment in which to evaluate the expression simulate, if not the current environment.

Details

The envelope command performs simulations and computes envelopes of a summary statistic based on the simulations. The result is an object that can be plotted to display the envelopes. The envelopes can be used to assess the goodness-of-fit of a point process model to point pattern data.

For the most basic use, if you have a point pattern X and you want to test Complete Spatial Randomness (CSR), type plot(envelope(X, Kest,nsim=39)) to see the K function for X plotted together with the envelopes of the K function for 39 simulations of CSR.

The envelope function is generic, with methods for the classes "ppp", "ppm", "kppm" and "slrm" described here. There are also methods for the classes "pp3", "lpp" and "lppm" which are described separately under envelope.pp3 and envelope.lpp. Envelopes can also be computed from other envelopes, using envelope.envelope.

To create simulation envelopes, the command envelope(Y, ...) first generates nsim random point patterns in one of the following ways.

- If Y is a point pattern (an object of class "ppp") and simulate=NULL, then we generate nsim simulations of Complete Spatial Randomness (i.e. nsim simulated point patterns each being a realisation of the uniform Poisson point process) with the same intensity as the pattern Y. (If Y is a multitype point pattern, then the simulated patterns are also given independent random marks; the probability distribution of the random marks is determined by the relative frequencies of marks in Y.)

- If Y is a fitted point process model (an object of class "ppm" or "kppm" or "slrm") and simulate=NULL, then this routine generates nsim simulated realisations of that model.
- If simulate is supplied, then it determines how the simulated point patterns are generated. It may be either
  - an expression in the R language, typically containing a call to a random generator. This expression will be evaluated nsim times to yield nsim point patterns. For example if simulate=expression(runifpoint(100)) then each simulated pattern consists of exactly 100 independent uniform random points.
  - a function in the R language, typically containing a call to a random generator. This function will be applied repeatedly to the original data pattern Y to yield nsim point patterns. For example if simulate=rlabel then each simulated pattern was generated by evaluating rlabel(Y) and consists of a randomly-relabelled version of Y.
  - a list of point patterns. The entries in this list will be taken as the simulated patterns.
  - an object of class "envelope". This should have been produced by calling envelope with the argument savepatterns=TRUE. The simulated point patterns that were saved in this object will be extracted and used as the simulated patterns for the new envelope computation. This makes it possible to plot envelopes for two different summary functions based on exactly the same set of simulated point patterns.

The summary statistic fun is applied to each of these simulated patterns. Typically fun is one of the functions Kest, Gest, Fest, Jest, pcf, Kcross, Kdot, Gcross, Gdot, Jcross, Jdot, Kmulti, Gmulti, Jmulti or Kinhom. It may also be a character string containing the name of one of these functions.

The statistic fun can also be a user-supplied function; if so, then it must have arguments X and r like those in the functions listed above, and it must return an object of class "fv".

Upper and lower critical envelopes are computed in one of the following ways:

pointwise: by default, envelopes are calculated pointwise (i.e. for each value of the distance argument r), by sorting the nsim simulated values, and taking the m-th lowest and m-th highest values, where m = nrank. For example if nrank=1, the upper and lower envelopes are the pointwise maximum and minimum of the simulated values.

The pointwise envelopes are **not** "confidence bands" for the true value of the function! Rather, they specify the critical points for a Monte Carlo test (Ripley, 1981). The test is constructed by choosing a *fixed* value of r, and rejecting the null hypothesis if the observed function value lies outside the envelope *at this value of* r. This test has exact significance level alpha = 2 * nrank/(1 + nsim).

simultaneous: if global=TRUE, then the envelopes are determined as follows. First we calculate the theoretical mean value of the summary statistic (if we are testing CSR, the theoretical value is supplied by fun; otherwise we perform a separate set of nsim2 simulations, compute the average of all these simulated values, and take this average as an estimate of the theoretical mean value). Then, for each simulation, we compare the simulated curve to the theoretical curve, and compute the maximum absolute difference between them (over the interval of r values specified by ginterval). This gives a deviation value d_i for each of the nsim simulations. Finally we take the m-th largest of the deviation values, where m=nrank, and call this dcrit. Then the simultaneous envelopes are of the form lo = expected - dcrit and hi = expected + dcrit where expected is either the theoretical mean value theo (if we are testing CSR) or the estimated theoretical value mmean (if we are testing another model). The simultaneous critical envelopes have constant width 2 * dcrit.

The simultaneous critical envelopes allow us to perform a different Monte Carlo test (Ripley, 1981). The test rejects the null hypothesis if the graph of the observed function lies outside the envelope **at any value of** r. This test has exact significance level `alpha = nrank/(1 + nsim)`. This test can also be performed using `mad.test`.

based on sample moments: if `VARIANCE=TRUE`, the algorithm calculates the (pointwise) sample mean and sample variance of the simulated functions. Then the envelopes are computed as mean plus or minus `nSD` standard deviations. These envelopes do not have an exact significance interpretation. They are a naive approximation to the critical points of the Neyman-Pearson test assuming the summary statistic is approximately Normally distributed.

The return value is an object of class `"fv"` containing the summary function for the data point pattern, the upper and lower simulation envelopes, and the theoretical expected value (exact or estimated) of the summary function for the model being tested. It can be plotted using `plot.envelope`.

If `VARIANCE=TRUE` then the return value also includes the sample mean, sample variance and other quantities.

Arguments can be passed to the function `fun` through `...`. This means that you simply specify these arguments in the call to `envelope`, and they will be passed to `fun`. In particular, the argument `correction` determines the edge correction to be used to calculate the summary statistic. See the section on Edge Corrections, and the Examples.

Arguments can also be passed to the function `fun` through the list `funargs`. This mechanism is typically used if an argument of `fun` has the same name as an argument of `envelope`. The list `funargs` should contain entries of the form `name=value`, where each `name` is the name of an argument of `fun`.

There is also an option, rarely used, in which different function arguments are used when computing the summary function for the data `Y` and for the simulated patterns. If `funYargs` is given, it will be used when the summary function for the data `Y` is computed, while `funargs` will be used when computing the summary function for the simulated patterns. This option is only needed in rare cases: usually the basic principle requires that the data and simulated patterns must be treated equally, so that `funargs` and `funYargs` should be identical.

If `Y` is a fitted cluster point process model (object of class `"kppm"`), and `simulate=NULL`, then the model is simulated directly using `simulate.kppm`.

If `Y` is a fitted Gibbs point process model (object of class `"ppm"`), and `simulate=NULL`, then the model is simulated by running the Metropolis-Hastings algorithm `rmh`. Complete control over this algorithm is provided by the arguments `start` and `control` which are passed to `rmh`.

For simultaneous critical envelopes (`global=TRUE`) the following options are also useful:

`ginterval` determines the interval of r values over which the deviation between curves is calculated. It should be a numeric vector of length 2. There is a sensible default (namely, the recommended plotting interval for `fun(X)`, or the range of `r` values if `r` is explicitly specified).

`transform` specifies a transformation of the summary function `fun` that will be carried out before the deviations are computed. Such transforms are useful if `global=TRUE` or `VARIANCE=TRUE`. The `transform` must be an expression object using the symbol `.` to represent the function value (and possibly other symbols recognised by `with.fv`). For example, the conventional way to normalise the K function (Ripley, 1981) is to transform it to the L function $L(r) = \sqrt{K(r)/\pi}$ and this is implemented by setting `transform=expression(sqrt(./pi))`.

It is also possible to extract the summary functions for each of the individual simulated point patterns, by setting savefuns=TRUE. Then the return value also has an attribute "simfuns" containing all the summary functions for the individual simulated patterns. It is an "fv" object containing functions named sim1, sim2, ... representing the nsim summary functions.

It is also possible to save the simulated point patterns themselves, by setting savepatterns=TRUE. Then the return value also has an attribute "simpatterns" which is a list of length nsim containing all the simulated point patterns.

See plot.envelope and plot.fv for information about how to plot the envelopes.

Different envelopes can be recomputed from the same data using envelope.envelope. Envelopes can be combined using pool.envelope.

Value

An object of class "envelope" and "fv", see fv.object, which can be printed and plotted directly.

Essentially a data frame containing columns

r	the vector of values of the argument r at which the summary function fun has been estimated
obs	values of the summary function for the data point pattern
lo	lower envelope of simulations
hi	upper envelope of simulations

and *either*

theo	theoretical value of the summary function under CSR (Complete Spatial Randomness, a uniform Poisson point process) if the simulations were generated according to CSR
mmean	estimated theoretical value of the summary function, computed by averaging simulated values, if the simulations were not generated according to CSR.

Additionally, if savepatterns=TRUE, the return value has an attribute "simpatterns" which is a list containing the nsim simulated patterns. If savefuns=TRUE, the return value has an attribute "simfuns" which is an object of class "fv" containing the summary functions computed for each of the nsim simulated patterns.

Errors and warnings

An error may be generated if one of the simulations produces a point pattern that is empty, or is otherwise unacceptable to the function fun.

The upper envelope may be NA (plotted as plus or minus infinity) if some of the function values computed for the simulated point patterns are NA. Whether this occurs will depend on the function fun, but it usually happens when the simulated point pattern does not contain enough points to compute a meaningful value.

Confidence intervals

Simulation envelopes do **not** compute confidence intervals; they generate significance bands. If you really need a confidence interval for the true summary function of the point process, use lohboot. See also varblock.

Edge corrections

It is common to apply a correction for edge effects when calculating a summary function such as the K function. Typically the user has a choice between several possible edge corrections. In a call to `envelope`, the user can specify the edge correction to be applied in `fun`, using the argument `correction`. See the Examples below.

Summary functions in spatstat Summary functions that are available in **spatstat**, such as `Kest`, `Gest` and `pcf`, have a standard argument called `correction` which specifies the name of one or more edge corrections.

The list of available edge corrections is different for each summary function, and may also depend on the kind of window in which the point pattern is recorded. In the case of `Kest` (the default and most frequently used value of `fun`) the best edge correction is Ripley's isotropic correction if the window is rectangular or polygonal, and the translation correction if the window is a binary mask. See the help files for the individual functions for more information.

All the summary functions in **spatstat** recognise the option `correction="best"` which gives the "best" (most accurate) available edge correction for that function.

In a call to `envelope`, if `fun` is one of the summary functions provided in **spatstat**, then the default is `correction="best"`. This means that *by default, the envelope will be computed using the "best" available edge correction*.

The user can override this default by specifying the argument `correction`. For example the computation can be accelerated by choosing another edge correction which is less accurate than the "best" one, but faster to compute.

User-written summary functions If `fun` is a function written by the user, then `envelope` has to guess what to do.

If `fun` has an argument called `correction`, or has `...` arguments, then `envelope` assumes that the function can handle a correction argument. To compute the envelope, `fun` will be called with a `correction` argument. The default is `correction="best"`, unless overridden in the call to `envelope`.

Otherwise, if `fun` does not have an argument called `correction` and does not have `...` arguments, then `envelope` assumes that the function *cannot* handle a correction argument. To compute the envelope, `fun` is called without a correction argument.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A., Diggle, P.J., Hardegen, A., Lawrence, T., Milne, R.K. and Nair, G. (2014) On tests of spatial pattern based on simulation envelopes. *Ecological Monographs* **84** (3) 477–489.

Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.

Diggle, P.J. *Statistical analysis of spatial point patterns*. Arnold, 2003.

Ripley, B.D. (1981) *Spatial statistics*. John Wiley and Sons.

Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.

Stoyan, D. and Stoyan, H. (1994) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

See Also

dclf.test, mad.test for envelope-based tests.

fv.object, plot.envelope, plot.fv, envelope.envelope, pool.envelope for handling envelopes. There are also methods for print and summary.

Kest, Gest, Fest, Jest, pcf, ppp, ppm, default.expand

Examples

```
X <- simdat
online <- interactive()
Nsim <- if(online) 19 else 3

# Envelope of K function under CSR
plot(envelope(X, nsim=Nsim))

# Translation edge correction (this is also FASTER):
if(online) {
  plot(envelope(X, correction="translate"))
} else {
 E <- envelope(X, nsim=Nsim, correction="translate")
}

# Global envelopes
if(online) {
  plot(envelope(X, Lest, global=TRUE))
  plot(envelope(X, Kest,  global=TRUE, scale=function(r) { r }))
} else {
 E <- envelope(X, Lest, nsim=Nsim, global=TRUE)
 E <- envelope(X, Kest, nsim=Nsim, global=TRUE, scale=function(r) { r })
 E
 summary(E)
}




# Envelope of G function under CSR
if(online) {
  plot(envelope(X, Gest))
} else {
 E <- envelope(X, Gest, correction="rs", nsim=Nsim)
}

# Envelope of L function under CSR
#  L(r) = sqrt(K(r)/pi)
if(online) {
 E <- envelope(X, Kest)
} else {
 E <- envelope(X, Kest, correction="border", nsim=Nsim)
}
plot(E, sqrt(./pi) ~ r)
```

```
# Simultaneous critical envelope for L function
# (alternatively, use Lest)
if(online) {
 plot(envelope(X, Kest, transform=expression(sqrt(./pi)), global=TRUE))
} else {
 E <- envelope(X, Kest, nsim=Nsim, correction="border",
              transform=expression(sqrt(./pi)), global=TRUE)
}

## One-sided envelope
if(online) {
 plot(envelope(X, Lest, alternative="less"))
} else {
 E <- envelope(X, Lest, nsim=Nsim, alternative="less")
}

# How to pass arguments needed to compute the summary functions:
# We want envelopes for Jcross(X, "A", "B")
# where "A" and "B" are types of points in the dataset 'demopat'

if(online) {
plot(envelope(demopat, Jcross, i="A", j="B"))
} else {
plot(envelope(demopat, Jcross, correction="rs", i="A", j="B", nsim=Nsim))
}

# Use of `simulate' expression
if(online) {
plot(envelope(cells, Gest, simulate=expression(runifpoint(42))))
plot(envelope(cells, Gest, simulate=expression(rMaternI(100,0.02))))
} else {
plot(envelope(cells, Gest, correction="rs", simulate=expression(runifpoint(42)), nsim=Nsim))
  plot(envelope(cells, Gest, correction="rs", simulate=expression(rMaternI(100, 0.02)),
nsim=Nsim, global=TRUE))
}

# Use of `simulate' function
if(online) {
  plot(envelope(amacrine, Kcross, simulate=rlabel))
} else {
  plot(envelope(amacrine, Kcross, simulate=rlabel, nsim=Nsim))
}

# Envelope under random toroidal shifts
if(online) {
plot(envelope(amacrine, Kcross, i="on", j="off",
              simulate=expression(rshift(amacrine, radius=0.25))))
}

# Envelope under random shifts with erosion
if(online) {
plot(envelope(amacrine, Kcross, i="on", j="off",
```

```
             simulate=expression(rshift(amacrine, radius=0.1, edge="erode"))))
}


# Note that the principle of symmetry, essential to the validity of
# simulation envelopes, requires that both the observed and
# simulated patterns be subjected to the same method of intensity
# estimation. In the following example it would be incorrect to set the
# argument 'lambda=red.dens' in the envelope command, because this
# would mean that the inhomogeneous K functions of the simulated
# patterns would be computed using the intensity function estimated
# from the original redwood data, violating the symmetry.  There is
# still a concern about the fact that the simulations are generated
# from a model that was fitted to the data; this is only a problem in
# small datasets.

if(online) {
red.dens <- density(redwood, sigma=bw.diggle, positive=TRUE)
plot(envelope(redwood, Kinhom, sigma=bw.diggle,
        simulate=expression(rpoispp(red.dens))))
}

# Precomputed list of point patterns
if(online) {
 nX <- npoints(X)
 PatList <- list()
 for(i in 1:Nsim) PatList[[i]] <- runifpoint(nX)
 E <- envelope(X, Kest, nsim=19, simulate=PatList)
} else {
 PatList <- list()
 for(i in 1:Nsim) PatList[[i]] <- runifpoint(10)
}
E <- envelope(X, Kest, nsim=Nsim, simulate=PatList)

# re-using the same point patterns
EK <- envelope(X, Kest, nsim=Nsim, savepatterns=TRUE)
EG <- envelope(X, Gest, nsim=Nsim, simulate=EK)
```

`envelope.envelope` *Recompute Envelopes*

Description

Given a simulation envelope (object of class `"envelope"`), compute another envelope from the same simulation data using different parameters.

Usage

```
## S3 method for class 'envelope'
envelope(Y, fun = NULL, ...,
                            transform=NULL, global=FALSE, VARIANCE=FALSE)
```

Arguments

Y	A simulation envelope (object of class "envelope").
fun	Optional. Summary function to be applied to the simulated point patterns.
..., transform, global, VARIANCE	Parameters controlling the type of envelope that is re-computed. See envelope.

Details

This function can be used to re-compute a simulation envelope from previously simulated data, using different parameter settings for the envelope: for example, a different significance level, or a global envelope instead of a pointwise envelope.

The function envelope is generic. This is the method for the class "envelope".

The argument Y should be a simulation envelope (object of class "envelope") produced by any of the methods for envelope. Additionally, Y must contain either

- the simulated point patterns that were used to create the original envelope (so Y should have been created by calling envelope with savepatterns=TRUE);
- the summary functions of the simulated point patterns that were used to create the original envelope (so Y should have been created by calling envelope with savefuns=TRUE).

If the argument fun is given, it should be a summary function that can be applied to the simulated point patterns that were used to create Y. The envelope of the summary function fun for these point patterns will be computed using the parameters specified in ....

If fun is not given, then:

- If Y contains the summary functions that were used to compute the original envelope, then the new envelope will be computed from these original summary functions.
- Otherwise, if Y contains the simulated point patterns. then the K function Kest will be applied to each of these simulated point patterns, and the new envelope will be based on the K functions.

The new envelope will be computed using the parameters specified in ....

See envelope for a full list of envelope parameters. Frequently-used parameters include nrank and nsim (to change the number of simulations used and the significance level of the envelope), global (to change from pointwise to global envelopes) and VARIANCE (to compute the envelopes from the sample moments instead of the ranks).

Value

An envelope (object of class "envelope".

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

envelope

Examples

```
E <- envelope(cells, Kest, nsim=19, savefuns=TRUE, savepatterns=TRUE)
E2 <- envelope(E, nrank=2)
Eg <- envelope(E, global=TRUE)
EG <- envelope(E, Gest)
EL <- envelope(E, transform=expression(sqrt(./pi)))
```

---

envelope.pp3 *Simulation Envelopes of Summary Function for 3D Point Pattern*

Description

Computes simulation envelopes of a summary function for a three-dimensional point pattern.

Usage

```
  ## S3 method for class 'pp3'
envelope(Y, fun=K3est, nsim=99, nrank=1, ...,
  funargs=list(), funYargs=funargs, simulate=NULL, verbose=TRUE,
  transform=NULL,global=FALSE,ginterval=NULL,use.theory=NULL,
  alternative=c("two.sided", "less", "greater"),
  scale=NULL, clamp=FALSE,
  savefuns=FALSE, savepatterns=FALSE,
  nsim2=nsim, VARIANCE=FALSE, nSD=2, Yname=NULL,
  maxnerr=nsim, rejectNA=FALSE, silent=FALSE,
  do.pwrong=FALSE, envir.simul=NULL)
```

Arguments

Y	A three-dimensional point pattern (object of class "pp3").
fun	Function that computes the desired summary statistic for a 3D point pattern.
nsim	Number of simulated point patterns to be generated when computing the envelopes.
nrank	Integer. Rank of the envelope value amongst the nsim simulated values. A rank of 1 means that the minimum and maximum simulated values will be used.
...	Extra arguments passed to fun.
funargs	A list, containing extra arguments to be passed to fun.
funYargs	Optional. A list, containing extra arguments to be passed to fun when applied to the original data Y only.

simulate	Optional. Specifies how to generate the simulated point patterns. If simulate is an expression in the R language, then this expression will be evaluated nsim times, to obtain nsim point patterns which are taken as the simulated patterns from which the envelopes are computed. If simulate is a function, then this function will be repeatedly applied to the data pattern Y to obtain nsim simulated patterns. If simulate is a list of point patterns, then the entries in this list will be treated as the simulated patterns from which the envelopes are computed. Alternatively simulate may be an object produced by the envelope command: see Details.
verbose	Logical flag indicating whether to print progress reports during the simulations.
transform	Optional. A transformation to be applied to the function values, before the envelopes are computed. An expression object (see Details).
global	Logical flag indicating whether envelopes should be pointwise (global=FALSE) or simultaneous (global=TRUE).
ginterval	Optional. A vector of length 2 specifying the interval of r values for the simultaneous critical envelopes. Only relevant if global=TRUE.
use.theory	Logical value indicating whether to use the theoretical value, computed by fun, as the reference value for simultaneous envelopes. Applicable only when global=TRUE.
alternative	Character string determining whether the envelope corresponds to a two-sided test (side="two.sided", the default) or a one-sided test with a lower critical boundary (side="less") or a one-sided test with an upper critical boundary (side="greater").
scale	Optional. Scaling function for global envelopes. A function in the R language which determines the relative scale of deviations, as a function of distance r, when computing the global envelopes. Applicable only when global=TRUE. Summary function values for distance r will be *divided* by scale(r) before the maximum deviation is computed. The resulting global envelopes will have width proportional to scale(r).
clamp	Logical value indicating how to compute envelopes when alternative="less" or alternative="greater". Deviations of the observed summary function from the theoretical summary function are initially evaluated as signed real numbers, with large positive values indicating consistency with the alternative hypothesis. If clamp=FALSE (the default), these values are not changed. If clamp=TRUE, any negative values are replaced by zero.
savefuns	Logical flag indicating whether to save all the simulated function values.
savepatterns	Logical flag indicating whether to save all the simulated point patterns.
nsim2	Number of extra simulated point patterns to be generated if it is necessary to use simulation to estimate the theoretical mean of the summary function. Only relevant when global=TRUE and the simulations are not based on CSR.
VARIANCE	Logical. If TRUE, critical envelopes will be calculated as sample mean plus or minus nSD times sample standard deviation.
nSD	Number of estimated standard deviations used to determine the critical envelopes, if VARIANCE=TRUE.
Yname	Character string that should be used as the name of the data point pattern Y when printing or plotting the results.

maxnerr	Maximum number of rejected patterns. If fun yields a fatal error when applied to a simulated point pattern (for example, because the pattern is empty and fun requires at least one point), the pattern will be rejected and a new random point pattern will be generated. If this happens more than maxnerr times, the algorithm will give up.
rejectNA	Logical value specifying whether to reject a simulated pattern if the resulting values of fun are all equal to NA, NaN or infinite. If FALSE (the default), then simulated patterns are only rejected when fun gives a fatal error.
silent	Logical value specifying whether to print a report each time a simulated pattern is rejected.
do.pwrong	Logical. If TRUE, the algorithm will also estimate the true significance level of the "wrong" test (the test that declares the summary function for the data to be significant if it lies outside the *pointwise* critical boundary at any point). This estimate is printed when the result is printed.
envir.simul	Environment in which to evaluate the expression simulate, if not the current environment.

Details

The envelope command performs simulations and computes envelopes of a summary statistic based on the simulations. The result is an object that can be plotted to display the envelopes. The envelopes can be used to assess the goodness-of-fit of a point process model to point pattern data.

The envelope function is generic, with methods for the classes "ppp", "ppm" and "kppm" described in the help file for envelope. This function envelope.pp3 is the method for three-dimensional point patterns (objects of class "pp3").

For the most basic use, if you have a 3D point pattern X and you want to test Complete Spatial Randomness (CSR), type plot(envelope(X, K3est,nsim=39)) to see the three-dimensional K function for X plotted together with the envelopes of the three-dimensional K function for 39 simulations of CSR.

To create simulation envelopes, the command envelope(Y, ...) first generates nsim random point patterns in one of the following ways.

- If simulate=NULL, then we generate nsim simulations of Complete Spatial Randomness (i.e. nsim simulated point patterns each being a realisation of the uniform Poisson point process) with the same intensity as the pattern Y.
- If simulate is supplied, then it determines how the simulated point patterns are generated. See envelope for details.

The summary statistic fun is applied to each of these simulated patterns. Typically fun is one of the functions K3est, G3est, F3est or pcf3est. It may also be a character string containing the name of one of these functions.

For further information, see the documentation for envelope.

Value

A function value table (object of class "fv") which can be plotted directly. See envelope for further details.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

References

Baddeley, A.J, Moyeed, R.A., Howard, C.V. and Boyde, A. (1993) Analysis of a three-dimensional point pattern with replication. *Applied Statistics* **42**, 641–668.

See Also

pp3, rpoispp3, K3est, G3est, F3est, pcf3est.

Examples

```
  X <- rpoispp3(20, box3())
if(interactive()) {
plot(envelope(X, nsim=39))
}
```

---

envelopeArray *Array of Simulation Envelopes of Summary Function*

Description

Compute an array of simulation envelopes using a summary function that returns an array of curves.

Usage

```
envelopeArray(X, fun, ..., dataname = NULL, verb = FALSE, reuse = TRUE)
```

Arguments

X	Object containing point pattern data. A point pattern (object of class "ppp", "lpp", "pp3" or "ppx") or a fitted point process model (object of class "ppm", "kppm" or "lppm").
fun	Function that computes the desired summary statistic for a point pattern. The result of fun should be a function array (object of class "fasp").
...	Arguments passed to envelope to control the simulations, or passed to fun when evaluating the function.
dataname	Optional character string name for the data.
verb	Logical value indicating whether to print progress reports.
reuse	Logical value indicating whether the envelopes in each panel should be based on the same set of simulated patterns (reuse=TRUE, the default) or on different, independent sets of simulated patterns (reuse=FALSE).

Details

This command is the counterpart of envelope when the function fun that is evaluated on each simulated point pattern will return an object of class "fasp" representing an array of summary functions.

Simulated point patterns are generated according to the rules described for envelope. In brief, if X is a point pattern, the algorithm generates simulated point patterns of the same kind, according to complete spatial randomness. If X is a fitted model, the algorithm generates simulated point patterns according to this model.

For each simulated point pattern Y, the function fun is invoked. The result Z <- fun(Y, ...) should be an object of class "fasp" representing an array of summary functions. The dimensions of the array Z should be the same for each simulated pattern Y.

This algorithm finds the simulation envelope of the summary functions in each cell of the array.

Value

An object of class "fasp" representing an array of envelopes.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

envelope, alltypes.

Examples

```
if(interactive()) {
  Nsim <- 19
  X <- finpines
  co <- "best"
} else {
  ## smaller task to reduce check time
  Nsim <- 3
  X <- finpines[c(FALSE, TRUE)]
  co <- "none"
}
A <- envelopeArray(X, markcrosscorr, nsim=Nsim, correction=co)
plot(A)
```

`eval.fasp` *Evaluate Expression Involving Function Arrays*

Description

Evaluates any expression involving one or more function arrays (`fasp` objects) and returns another function array.

Usage

```
eval.fasp(expr, envir, dotonly=TRUE)
```

Arguments

`expr`	An expression involving the names of objects of class `"fasp"`.
`envir`	Optional. The environment in which to evaluate the expression, or a named list containing `"fasp"` objects to be used in the expression.
`dotonly`	Logical. Passed to `eval.fv`.

Details

This is a wrapper to make it easier to perform pointwise calculations with the arrays of summary functions used in spatial statistics.

A function array (object of class `"fasp"`) can be regarded as a matrix whose entries are functions. Objects of this kind are returned by the command `alltypes`.

Suppose `X` is an object of class `"fasp"`. Then `eval.fasp(X+3)` effectively adds 3 to the value of every function in the array `X`, and returns the resulting object.

Suppose `X` and `Y` are two objects of class `"fasp"` which are compatible (for example the arrays must have the same dimensions). Then `eval.fasp(X + Y)` will add the corresponding functions in each cell of the arrays `X` and `Y`, and return the resulting array of functions.

Suppose `X` is an object of class `"fasp"` and `f` is an object of class `"fv"`. Then `eval.fasp(X + f)` will add the function `f` to the functions in each cell of the array `X`, and return the resulting array of functions.

In general, `expr` can be any expression involving (a) the *names* of objects of class `"fasp"` or `"fv"`, (b) scalar constants, and (c) functions which are vectorised. See the Examples.

First `eval.fasp` determines which of the *variable names* in the expression `expr` refer to objects of class `"fasp"`. The expression is then evaluated for each cell of the array using `eval.fv`.

The expression `expr` must be vectorised. There must be at least one object of class `"fasp"` in the expression. All such objects must be compatible.

Value

Another object of class `"fasp"`.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

fasp.object, Kest

Examples

```
K <- alltypes(amacrine, "K")

# expressions involving a fasp object
eval.fasp(K + 3)
L <- eval.fasp(sqrt(K/pi))

# expression involving two fasp objects
D <- eval.fasp(K - L)

# subtracting the unmarked K function from the cross-type K functions
K0 <- Kest(unmark(amacrine))
DK <- eval.fasp(K - K0)

## Use of 'envir'
S <- eval.fasp(1-G, list(G=alltypes(amacrine, 'G')))
```

---

eval.fv *Evaluate Expression Involving Functions*

Description

Evaluates any expression involving one or more function value (fv) objects, and returns another object of the same kind.

Usage

```
eval.fv(expr, envir, dotonly=TRUE, equiv=NULL, relabel=TRUE)
```

Arguments

expr	An expression.
envir	Optional. The environment in which to evaluate the expression, or a named list containing "fv" objects to be used in the expression.
dotonly	Logical. See Details.
equiv	Mapping between column names of different objects that are deemed to be equivalent. See Details.
relabel	Logical value indicating whether to compute appropriate labels for the resulting function. This should normally be TRUE (the default). See Details.

Details

This is a wrapper to make it easier to perform pointwise calculations with the summary functions used in spatial statistics.

An object of class "fv" is essentially a data frame containing several different statistical estimates of the same function. Such objects are returned by Kest and its relatives.

For example, suppose X is an object of class "fv" containing several different estimates of the Ripley's K function $K(r)$, evaluated at a sequence of values of r. Then eval.fv(X+3) effectively adds 3 to each function estimate in X, and returns the resulting object.

Suppose X and Y are two objects of class "fv" which are compatible (in particular they have the same vector of r values). Then eval.im(X + Y) will add the corresponding function values in X and Y, and return the resulting function.

In general, expr can be any expression involving (a) the *names* of objects of class "fv", (b) scalar constants, and (c) functions which are vectorised. See the Examples.

First eval.fv determines which of the *variable names* in the expression expr refer to objects of class "fv". Each such name is replaced by a vector containing the function values. The expression is then evaluated. The result should be a vector; it is taken as the new vector of function values.

The expression expr must be vectorised. There must be at least one object of class "fv" in the expression. If the objects are not compatible, they will be made compatible by harmonise.fv.

If dotonly=TRUE (the default), the expression will be evaluated only for those columns of an "fv" object that contain values of the function itself (rather than values of the derivative of the function, the hazard rate, etc). If dotonly=FALSE, the expression will be evaluated for all columns.

For example the result of Fest includes several columns containing estimates of the empty space function $F(r)$, but also includes an estimate of the *hazard* $h(r)$ of $F(r)$. Transformations that are valid for F may not be valid for h. Accordingly, h would normally be omitted from the calculation.

The columns of an object x that represent the function itself are identified by its "dot" names, fvnames(x, "."). They are the columns normally plotted by plot.fv and identified by the symbol "." in plot formulas in plot.fv.

The argument equiv can be used to specify that two different column names in different function objects are mathematically equivalent or cognate. It should be a list of name=value pairs, or a named vector of character strings, indicating the pairing of equivalent names. (Without this argument, these columns would be discarded.) See the Examples.

The argument relabel should normally be TRUE (the default). It determines whether to compute appropriate mathematical labels and descriptions for the resulting function object (used when the object is printed or plotted). If relabel=FALSE then this does not occur, and the mathematical labels and descriptions in the result are taken from the function object that appears first in the expression. This reduces computation time slightly (for advanced use only).

Value

Another object of class "fv".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

fv.object, Kest

Examples

```
# manipulating the K function
X <- runifrect(42)
Ks <- Kest(X)

eval.fv(Ks + 3)
Ls <- eval.fv(sqrt(Ks/pi))

# manipulating two K functions
Y <- runifrect(20)
Kr <- Kest(Y)

Kdif <- eval.fv(Ks - Kr)
Z <- eval.fv(sqrt(Ks/pi) - sqrt(Kr/pi))

## Use of 'envir'
U <- eval.fv(sqrt(K), list(K=Ks))

## Use of 'equiv'
Fc <- Fest(cells)
Gc <- Gest(cells)
# Hanisch and Chiu-Stoyan estimators are cognate
Dc <- eval.fv(Fc - Gc, equiv=list(cs="han"))
```

Extract.fasp *Extract Subset of Function Array*

Description

Extract a subset of a function array (an object of class "fasp").

Usage

```
  ## S3 method for class 'fasp'
x[I, J, drop=TRUE,...]
```

Arguments

x	A function array. An object of class "fasp".
I	any valid expression for a subset of the row indices of the array.
J	any valid expression for a subset of the column indices of the array.
drop	Logical. When the selected subset consists of only one cell of the array, if drop=FALSE the result is still returned as a 1×1 array of functions (class "fasp") while if drop=TRUE it is returned as a function (class "fv").
...	Ignored.

Details

A function array can be regarded as a matrix whose entries are functions. See fasp.object for an explanation of function arrays.

This routine extracts a sub-array according to the usual conventions for matrix indexing.

Value

A function array (of class "fasp"). Exceptionally, if the array has only one cell, and if drop=TRUE, then the result is a function value table (class "fv").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>

See Also

fasp.object

Examples

```
online <- interactive()
# Lansing woods data - multitype points with 6 types
X <- lansing

if(!online) {
  # subsample data (from 2251 to 450 points) to shorten check time
  X <- X[c(FALSE,FALSE,FALSE,FALSE,TRUE)]
}

a <- alltypes(X, 'K')

# extract first three marks only
b <- a[1:3,1:3]
if(online) {plot(b)}
# subset of array pertaining to hickories
h <- a["hickory", ]
if(online) {plot(h)}
```

Extract.fv *Extract or Replace Subset of Function Values*

Description

Extract or replace a subset of an object of class "fv".

Usage

```
  ## S3 method for class 'fv'
x[i, j, ..., drop=FALSE]

  ## S3 replacement method for class 'fv'
x[i, j] <- value

  ## S3 replacement method for class 'fv'
x$name <- value
```

Arguments

x	a function value object, of class "fv" (see fv.object). Essentially a data frame.
i	any appropriate subset index. Selects a subset of the rows of the data frame, i.e. a subset of the domain of the function(s) represented by x.
j	any appropriate subset index for the columns of the data frame. Selects some of the functions present in x.
name	the name of a column of the data frame.
...	Ignored.
drop	Logical. If TRUE, the result is a data frame or vector containing the selected rows and columns of data. If FALSE (the default), the result is another object of class "fv".
value	Replacement value for the column or columns selected by name or j.

Details

These functions extract a designated subset of an object of class "fv", or replace the designated subset with other data, or delete the designated subset.

The subset is specified by the row index i and column index j, or by the column name name. Either i or j may be missing, or both may be missing.

The function [.fv is a method for the generic operator [for the class "fv". It extracts the designated subset of x, and returns it as another object of class "fv" (if drop=FALSE) or as a data frame or vector (if drop=TRUE).

The function [<-.fv is a method for the generic operator [<- for the class "fv". If value is NULL, the designated subset of x will be deleted from x. Otherwise, the designated subset of x will be replaced by the data contained in value. The return value is the modified object x.

The function $<-.fv is a method for the generic operator $<- for the class "fv". If value is NULL, the designated column of x will be deleted from x. Otherwise, the designated column of x will be replaced by the data contained in value. The return value is the modified object x.

Value

The result of [.fv with drop=TRUE is a data frame or vector.

Otherwise, the result is another object of class "fv".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

fv.object

Examples

```
K <- Kest(cells)

# discard the estimates of K(r) for r  > 0.1
Ksub <- K[K$r <= 0.1, ]

# extract the border method estimates
bor <- K[ , "border", drop=TRUE]
# or equivalently
bor <- K$border

# remove the border-method estimates
K$border <- NULL
K
```

F3est *Empty Space Function of a Three-Dimensional Point Pattern*

Description

Estimates the empty space function $F_3(r)$ from a three-dimensional point pattern.

Usage

```
F3est(X, ..., rmax = NULL, nrval = 128, vside = NULL,
              correction = c("rs", "km", "cs"),
              sphere = c("fudge", "ideal", "digital"))
```

Arguments

X	Three-dimensional point pattern (object of class "pp3").
...	Ignored.
rmax	Optional. Maximum value of argument r for which $F_3(r)$ will be estimated.
nrval	Optional. Number of values of r for which $F_3(r)$ will be estimated. A large value of nrval is required to avoid discretisation effects.
vside	Optional. Side length of the voxels in the discrete approximation.
correction	Optional. Character vector specifying the edge correction(s) to be applied. See Details.
sphere	Optional. Character string specifying how to calculate the theoretical value of $F_3(r)$ for a Poisson process. See Details.

Details

For a stationary point process Φ in three-dimensional space, the empty space function is

$$F_3(r) = P(d(0, \Phi) \le r)$$

where $d(0, \Phi)$ denotes the distance from a fixed origin 0 to the nearest point of Φ.

The three-dimensional point pattern `X` is assumed to be a partial realisation of a stationary point process Φ. The empty space function of Φ can then be estimated using techniques described in the References.

The box containing the point pattern is discretised into cubic voxels of side length `vside`. The distance function $d(u, \Phi)$ is computed for every voxel centre point u using a three-dimensional version of the distance transform algorithm (Borgefors, 1986). The empirical cumulative distribution function of these values, with appropriate edge corrections, is the estimate of $F_3(r)$.

The available edge corrections are:

`"rs"`: the reduced sample (aka minus sampling, border correction) estimator (Baddeley et al, 1993)

`"km"`: the three-dimensional version of the Kaplan-Meier estimator (Baddeley and Gill, 1997)

`"cs"`: the three-dimensional generalisation of the Chiu-Stoyan or Hanisch estimator (Chiu and Stoyan, 1998).

Alternatively `correction="all"` selects all options.

The result includes a column `theo` giving the theoretical value of $F_3(r)$ for a uniform Poisson process (Complete Spatial Randomness). This value depends on the volume of the sphere of radius `r` measured in the discretised distance metric. The argument `sphere` determines how this will be calculated.

- If `sphere="ideal"` the calculation will use the volume of an ideal sphere of radius r namely $(4/3)\pi r^3$. This is not recommended because the theoretical values of $F_3(r)$ are inaccurate.
- If `sphere="fudge"` then the volume of the ideal sphere will be multiplied by 0.78, which gives the approximate volume of the sphere in the discretised distance metric.
- If `sphere="digital"` then the volume of the sphere in the discretised distance metric is computed exactly using another distance transform. This takes longer to compute, but is exact.

Value

A function value table (object of class `"fv"`) that can be plotted, printed or coerced to a data frame containing the function values.

Warnings

A small value of `vside` and a large value of `nrval` are required for reasonable accuracy.

The default value of `vside` ensures that the total number of voxels is `2^22` or about 4 million. To change the default number of voxels, see `spatstat.options("nvoxel")`.

Author(s)

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

and Rana Moyeed.

References

Baddeley, A.J, Moyeed, R.A., Howard, C.V. and Boyde, A. Analysis of a three-dimensional point pattern with replication. *Applied Statistics* **42** (1993) 641–668.

Baddeley, A.J. and Gill, R.D. (1997) Kaplan-Meier estimators of interpoint distance distributions for spatial point processes. *Annals of Statistics* **25**, 263–292.

Borgefors, G. (1986) Distance transformations in digital images. *Computer Vision, Graphics and Image Processing* **34**, 344–371.

Chiu, S.N. and Stoyan, D. (1998) Estimators of distance distributions for spatial patterns. *Statistica Neerlandica* **52**, 239–246.

See Also

`pp3` to create a three-dimensional point pattern (object of class `"pp3"`).

`G3est`, `K3est`, `pcf3est` for other summary functions of a three-dimensional point pattern.

`Fest` to estimate the empty space function of point patterns in two dimensions.

Examples

```
X <- rpoispp3(42)
Z <- F3est(X)
if(interactive()) plot(Z)
```

`fasp.object` *Function Arrays for Spatial Patterns*

Description

A class `"fasp"` to represent a "matrix" of functions, amenable to plotting as a matrix of plot panels.

Details

An object of this class is a convenient way of storing (and later plotting, editing, etc) a set of functions $f_{i,j}(r)$ of a real argument r, defined for each possible pair (i, j) of indices $1 \leq i, j \leq n$. We may think of this as a matrix or array of functions $f_{i,j}$.

Function arrays are particularly useful in the analysis of a multitype point pattern (a point pattern in which the points are identified as belonging to separate types). We may want to compute a summary function for the points of type i only, for each of the possible types i. This produces a $1 \times m$ array of functions. Alternatively we may compute a summary function for each possible pair of types (i, j). This produces an $m \times m$ array of functions.

For multitype point patterns the command `alltypes` will compute arrays of summary functions for each possible type or for each possible pair of types. The function `alltypes` returns an object of class `"fasp"`.

An object of class `"fasp"` is a list containing at least the following components:

fns A list of data frames, each representing one of the functions.

which A matrix representing the spatial arrangement of the functions. If `which[i,j] = k` then the function represented by `fns[[k]]` should be plotted in the panel at position (i, j). If `which[i,j] = NA` then nothing is plotted in that position.

titles A list of character strings, providing suitable plotting titles for the functions.

default.formulae A list of default formulae for plotting each of the functions.

title A character string, giving a default title for the array when it is plotted.

Functions available

There are methods for `plot`, `print` and `"["` for this class.

The plot method displays the entire array of functions. The method `[.fasp` selects a sub-array using the natural indices `i,j`.

The command `eval.fasp` can be used to apply a transformation to each function in the array, and to combine two arrays.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

`alltypes`, `plot.fasp`, `[.fasp`, `eval.fasp`

Examples

```
GG <- alltypes(amacrine, 'G')

plot(GG)

# select the row corresponding to cells of type "on"
Gon <- GG["on", ]
plot(Gon)

# extract the G function for i = "on", j = "off"
Gonoff <- GG["on", "off", drop=TRUE]

# Fisher variance stabilising transformation
GGfish <- eval.fasp(asin(sqrt(GG)))
plot(GGfish)
```

`Fest` *Estimate the Empty Space Function or its Hazard Rate*

Description

Estimates the empty space function $F(r)$ or its hazard rate $h(r)$ from a point pattern in a window of arbitrary shape.

Usage

```
Fest(X, ..., eps, r=NULL, breaks=NULL,
     correction=c("rs", "km", "cs"),
     domain=NULL)

Fhazard(X, ...)
```

Arguments

`X`	The observed point pattern, from which an estimate of $F(r)$ will be computed. An object of class `ppp`, or data in any format acceptable to `as.ppp()`.
`...`	Extra arguments, passed from `Fhazard` to `Fest`. Extra arguments to `Fest` are ignored.
`eps`	Optional. A positive number. The resolution of the discrete approximation to Euclidean distance (see below). There is a sensible default.
`r`	Optional. Numeric vector. The values of the argument r at which $F(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
`breaks`	This argument is for internal use only.
`correction`	Optional. The edge correction(s) to be used to estimate $F(r)$. A vector of character strings selected from `"none"`, `"rs"`, `"km"`, `"cs"` and `"best"`. Alternatively `correction="all"` selects all options.
`domain`	Optional. Calculations will be restricted to this subset of the window. See Details.

Details

`Fest` computes an estimate of the empty space function $F(r)$, and `Fhazard` computes an estimate of its hazard rate $h(r)$.

The empty space function (also called the "*spherical contact distribution*" or the "*point-to-nearest-event*" distribution) of a stationary point process X is the cumulative distribution function F of the distance from a fixed point in space to the nearest point of X.

An estimate of F derived from a spatial point pattern dataset can be used in exploratory data analysis and formal inference about the pattern (Cressie, 1991; Diggle, 1983; Ripley, 1988). In exploratory analyses, the estimate of F is a useful statistic summarising the sizes of gaps in the pattern. For

inferential purposes, the estimate of F is usually compared to the true value of F for a completely random (Poisson) point process, which is

$$F(r) = 1 - e^{-\lambda \pi r^2}$$

where λ is the intensity (expected number of points per unit area). Deviations between the empirical and theoretical F curves may suggest spatial clustering or spatial regularity.

This algorithm estimates the empty space function F from the point pattern `X`. It assumes that `X` can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in `X`) may have arbitrary shape.

The argument `X` is interpreted as a point pattern object (of class `"ppp"`, see `ppp.object`) and can be supplied in any of the formats recognised by `as.ppp`.

The algorithm uses two discrete approximations which are controlled by the parameter `eps` and by the spacing of values of `r` respectively. (See below for details.) First-time users are strongly advised not to specify these arguments.

The estimation of F is hampered by edge effects arising from the unobservability of points of the random pattern outside the window. An edge correction is needed to reduce bias (Baddeley, 1998; Ripley, 1988). The edge corrections implemented here are the border method or "*reduced sample*" estimator, the spatial Kaplan-Meier estimator (Baddeley and Gill, 1997) and the Chiu-Stoyan estimator (Chiu and Stoyan, 1998).

Our implementation makes essential use of the distance transform algorithm of image processing (Borgefors, 1986). A fine grid of pixels is created in the observation window. The Euclidean distance between two pixels is approximated by the length of the shortest path joining them in the grid, where a path is a sequence of steps between adjacent pixels, and horizontal, vertical and diagonal steps have length 1, 1 and $\sqrt{2}$ respectively in pixel units. If the pixel grid is sufficiently fine then this is an accurate approximation.

The parameter `eps` is the pixel width of the rectangular raster used to compute the distance transform (see below). It must not be too large: the absolute error in distance values due to discretisation is bounded by `eps`.

If `eps` is not specified, the function checks whether the window `Window(X)` contains pixel raster information. If so, then `eps` is set equal to the pixel width of the raster; otherwise, `eps` defaults to 1/100 of the width of the observation window.

The argument `r` is the vector of values for the distance r at which $F(r)$ should be evaluated. It is also used to determine the breakpoints (in the sense of `hist`) for the computation of histograms of distances. The estimators are computed from histogram counts. This introduces a discretisation error which is controlled by the fineness of the breakpoints.

First-time users would be strongly advised not to specify `r`. However, if it is specified, `r` must satisfy `r[1] = 0`, and `max(r)` must be larger than the radius of the largest disc contained in the window. Furthermore, the spacing of successive `r` values must be very fine (ideally not greater than `eps/4`).

The algorithm also returns an estimate of the hazard rate function, $h(r)$ of $F(r)$. The hazard rate is defined by

$$h(r) = -\frac{d}{dr} \log(1 - F(r))$$

The hazard rate of F has been proposed as a useful exploratory statistic (Baddeley and Gill, 1994). The estimate of $h(r)$ given here is a discrete approximation to the hazard rate of the Kaplan-Meier

estimator of F. Note that F is absolutely continuous (for any stationary point process X), so the hazard function always exists (Baddeley and Gill, 1997).

If the argument `domain` is given, the estimate of $F(r)$ will be based only on the empty space distances measured from locations inside `domain` (although their nearest data points may lie outside `domain`). This is useful in bootstrap techniques. The argument `domain` should be a window (object of class `"owin"`) or something acceptable to `as.owin`. It must be a subset of the window of the point pattern `X`.

The naive empirical distribution of distances from each location in the window to the nearest point of the data pattern, is a biased estimate of F. However this is also returned by the algorithm (if `correction="none"`), as it is sometimes useful in other contexts. Care should be taken not to use the uncorrected empirical F as if it were an unbiased estimator of F.

Value

An object of class `"fv"`, see `fv.object`, which can be plotted directly using `plot.fv`.

The result of `Fest` is essentially a data frame containing up to seven columns:

`r`	the values of the argument r at which the function $F(r)$ has been estimated
`rs`	the "reduced sample" or "border correction" estimator of $F(r)$
`km`	the spatial Kaplan-Meier estimator of $F(r)$
`hazard`	the hazard rate $\lambda(r)$ of $F(r)$ by the spatial Kaplan-Meier method
`cs`	the Chiu-Stoyan estimator of $F(r)$
`raw`	the uncorrected estimate of $F(r)$, i.e. the empirical distribution of the distance from a random point in the window to the nearest point of the data pattern `X`
`theo`	the theoretical value of $F(r)$ for a stationary Poisson process of the same estimated intensity.

The result of `Fhazard` contains only three columns

`r`	the values of the argument r at which the hazard rate $h(r)$ has been estimated
`hazard`	the spatial Kaplan-Meier estimate of the hazard rate $h(r)$
`theo`	the theoretical value of $h(r)$ for a stationary Poisson process of the same estimated intensity.

Warnings

The reduced sample (border method) estimator of F is pointwise approximately unbiased, but need not be a valid distribution function; it may not be a nondecreasing function of r. Its range is always within $[0, 1]$.

The spatial Kaplan-Meier estimator of F is always nondecreasing but its maximum value may be less than 1.

The estimate of hazard rate $h(r)$ returned by the algorithm is an approximately unbiased estimate for the integral of $h()$ over the corresponding histogram cell. It may exhibit oscillations due to discretisation effects. We recommend modest smoothing, such as kernel smoothing with kernel width equal to the width of a histogram cell, using `Smooth.fv`.

Note

Sizeable amounts of memory may be needed during the calculation.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

References

Baddeley, A.J. Spatial sampling and censoring. In O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*. Chapman and Hall, 1998. Chapter 2, pages 37-78.

Baddeley, A.J. and Gill, R.D. The empty space hazard of a spatial pattern. Research Report 1994/3, Department of Mathematics, University of Western Australia, May 1994.

Baddeley, A.J. and Gill, R.D. Kaplan-Meier estimators of interpoint distance distributions for spatial point processes. *Annals of Statistics* **25** (1997) 263-292.

Borgefors, G. Distance transformations in digital images. *Computer Vision, Graphics and Image Processing* **34** (1986) 344-371.

Chiu, S.N. and Stoyan, D. (1998) Estimators of distance distributions for spatial patterns. *Statistica Neerlandica* **52**, 239–246.

Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.

Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.

Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.

Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.

See Also

Gest, Jest, Kest, km.rs, reduced.sample, kaplan.meier

Examples

```
Fc <- Fest(cells, 0.01)

# Tip: don't use F for the left hand side!
# That's an abbreviation for FALSE

plot(Fc)

# P-P style plot
plot(Fc, cbind(km, theo) ~ theo)

# The empirical F is above the Poisson F
# indicating an inhibited pattern

if(interactive()) {
```

```
plot(Fc, . ~ theo)
plot(Fc, asin(sqrt(.)) ~ asin(sqrt(theo)))
}
```

Finhom *Inhomogeneous Empty Space Function*

Description

Estimates the inhomogeneous empty space function of a non-stationary point pattern.

Usage

```
Finhom(X, lambda = NULL, lmin = NULL, ...,
       sigma = NULL, varcov = NULL,
       r = NULL, breaks = NULL, ratio = FALSE,
       update = TRUE, warn.bias=TRUE, savelambda=FALSE)
```

Arguments

X	The observed data point pattern, from which an estimate of the inhomogeneous F function will be computed. An object of class "ppp" or in a format recognised by as.ppp()
lambda	Optional. Values of the estimated intensity function. Either a vector giving the intensity values at the points of the pattern X, a pixel image (object of class "im") giving the intensity values at all locations, a fitted point process model (object of class "ppm") or a function(x,y) which can be evaluated to give the intensity value at any location.
lmin	Optional. The minimum possible value of the intensity over the spatial domain. A positive numerical value.
sigma, varcov	Optional arguments passed to density.ppp to control the smoothing bandwidth, when lambda is estimated by kernel smoothing.
...	Extra arguments passed to as.mask to control the pixel resolution, or passed to density.ppp to control the smoothing bandwidth.
r	vector of values for the argument r at which the inhomogeneous K function should be evaluated. Not normally given by the user; there is a sensible default.
breaks	This argument is for internal use only.
ratio	Logical. If TRUE, the numerator and denominator of the estimate will also be saved, for use in analysing replicated point patterns.
update	Logical. If lambda is a fitted model (class "ppm" or "kppm") and update=TRUE (the default), the model will first be refitted to the data X (using update.ppm or update.kppm) before the fitted intensity is computed. If update=FALSE, the fitted intensity of the model will be computed without fitting it to X.

warn.bias — Logical value specifying whether to issue a warning when the inhomogeneity correction factor takes extreme values, which can often lead to biased results. This usually occurs when insufficient smoothing is used to estimate the intensity.

savelambda — Logical value specifying whether to save the values of lmin and lambda as attributes of the result.

Details

This command computes estimates of the inhomogeneous F-function (van Lieshout, 2010) of a point pattern. It is the counterpart, for inhomogeneous spatial point patterns, of the empty space function F for homogeneous point patterns computed by Fest.

The argument X should be a point pattern (object of class "ppp").

The inhomogeneous F function is computed using the border correction, equation (6) in Van Lieshout (2010).

The argument lambda should supply the (estimated) values of the intensity function λ of the point process. It may be either

a numeric vector containing the values of the intensity function at the points of the pattern X.

a pixel image (object of class "im") assumed to contain the values of the intensity function at all locations in the window.

a fitted point process model (object of class "ppm" or "kppm") whose fitted *trend* can be used as the fitted intensity. (If update=TRUE the model will first be refitted to the data X before the trend is computed.)

a function which can be evaluated to give values of the intensity at any locations.

omitted: if lambda is omitted, then it will be estimated using a 'leave-one-out' kernel smoother.

If lambda is a numeric vector, then its length should be equal to the number of points in the pattern X. The value lambda[i] is assumed to be the the (estimated) value of the intensity $\lambda(x_i)$ for the point x_i of the pattern X. Each value must be a positive number; NA's are not allowed.

If lambda is a pixel image, the domain of the image should cover the entire window of the point pattern. If it does not (which may occur near the boundary because of discretisation error), then the missing pixel values will be obtained by applying a Gaussian blur to lambda using blur, then looking up the values of this blurred image for the missing locations. (A warning will be issued in this case.)

If lambda is a function, then it will be evaluated in the form lambda(x,y) where x and y are vectors of coordinates of the points of X. It should return a numeric vector with length equal to the number of points in X.

If lambda is omitted, then it will be estimated using a 'leave-one-out' kernel smoother. The estimate lambda[i] for the point X[i] is computed by removing X[i] from the point pattern, applying kernel smoothing to the remaining points using density.ppp, and evaluating the smoothed intensity at the point X[i]. The smoothing kernel bandwidth is controlled by the arguments sigma and varcov, which are passed to density.ppp along with any extra arguments.

Value

An object of class "fv", see fv.object, which can be plotted directly using plot.fv.

Author(s)

Original code by Marie-Colette van Lieshout. C implementation and R adaptation by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Ege Rubak <rubak@math.aau.dk>.

References

Van Lieshout, M.N.M. and Baddeley, A.J. (1996) A nonparametric measure of spatial interaction in point patterns. *Statistica Neerlandica* **50**, 344–361.

Van Lieshout, M.N.M. (2010) A J-function for inhomogeneous point processes. *Statistica Neerlandica* **65**, 183–201.

See Also

Ginhom, Jinhom, Fest

Examples

```
online <- interactive()
if(online) {
  plot(Finhom(swedishpines, sigma=10))
  plot(Finhom(swedishpines, sigma=bw.diggle, adjust=2))
} else {
  ## use a coarse grid for faster computation and package testing
  plot(Finhom(swedishpines, sigma=10, dimyx=32))
}
```

FmultiInhom *Inhomogeneous Marked F-Function*

Description

For a marked point pattern, estimate the inhomogeneous version of the multitype F function, effectively the cumulative distribution function of the distance from a fixed point to the nearest point in subset J, adjusted for spatially varying intensity.

Usage

```
Fmulti.inhom(X, J,
             lambda = NULL, lambdaJ = NULL, lambdamin = NULL,
             ...,
             r = NULL)

FmultiInhom(X, J,
             lambda = NULL, lambdaJ = NULL, lambdamin = NULL,
             ...,
             r = NULL)
```

Arguments

X	A spatial point pattern (object of class "ppp".
J	A subset index specifying the subset of points to which distances are measured. Any kind of subset index acceptable to [.ppp.
lambda	Intensity estimates for each point of X. A numeric vector of length equal to npoints(X). Incompatible with lambdaJ.
lambdaJ	Intensity estimates for each point of X[J]. A numeric vector of length equal to npoints(X[J]). Incompatible with lambda.
lambdamin	A lower bound for the intensity, or at least a lower bound for the values in lambdaJ or lambda[J].
...	Extra arguments passed to as.mask to control the pixel resolution for the computation.
r	Vector of distance values at which the inhomogeneous G function should be estimated. There is a sensible default.

Details

See Cronie and Van Lieshout (2015).

The functions FmultiInhom and Fmulti.inhom are identical.

Value

Object of class "fv" containing the estimate of the inhomogeneous multitype F function.

Author(s)

Ottmar Cronie and Marie-Colette van Lieshout. Rewritten for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu

References

Cronie, O. and Van Lieshout, M.N.M. (2015) Summary statistics for inhomogeneous marked point processes. *Annals of the Institute of Statistical Mathematics* DOI: 10.1007/s10463-015-0515-z

See Also

Finhom

Examples

```
X <- amacrine
J <- (marks(X) == "off")
online <- interactive()
eps <- if(online) NULL else 0.025
if(online && require(spatstat.model)) {
  mod <- ppm(X ~ marks * x, eps=eps)
  lambdaX <- fitted(mod, dataonly=TRUE)
  lambdaOff <- predict(mod, eps=eps)[["off"]]
  lmin <- min(lambdaOff) * 0.9
```

```
} else {
  ## faster computation for package checker only
  lambdaX <- intensity(X)[as.integer(marks(X))]
  lmin <- intensity(X)[2] * 0.9
}

plot(FmultiInhom(X, J, lambda=lambdaX, lambdamin=lmin, eps=eps))
```

---

formula.fv *Extract or Change the Plot Formula for a Function Value Table*

Description

Extract or change the default plotting formula for an object of class "fv" (function value table).

Usage

```
## S3 method for class 'fv'
formula(x, ...)

formula(x, ...) <- value

## S3 replacement method for class 'fv'
formula(x, ...) <- value
```

Arguments

x	An object of class "fv", containing the values of several estimates of a function.
...	Arguments passed to other methods.
value	New value of the formula. Either a formula or a character string.

Details

A function value table (object of class "fv", see fv.object) is a convenient way of storing and plotting several different estimates of the same function.

The default behaviour of plot(x) for a function value table x is determined by a formula associated with x called its *plot formula*. See plot.fv for explanation about these formulae.

The function formula.fv is a method for the generic command formula. It extracts the plot formula associated with the object.

The function formula<- is generic. It changes the formula associated with an object.

The function formula<-.fv is the method for formula<- for the class "fv". It changes the plot formula associated with the object.

Value

The result of formula.fv is a character string containing the plot formula. The result of formula<-.fv is a new object of class "fv".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

fv, plot.fv, formula.

Examples

```
K <- Kest(cells)
formula(K)
formula(K) <- (iso ~ r)
```

---

fryplot *Fry Plot of Point Pattern*

Description

Displays the Fry plot (Patterson plot) of a spatial point pattern.

Usage

```
fryplot(X, ..., width=NULL, from=NULL, to=NULL, axes=FALSE)
frypoints(X, from=NULL, to=NULL, dmax=Inf)
```

Arguments

X	A point pattern (object of class "ppp") or something acceptable to as.ppp.
...	Optional arguments to control the appearance of the plot.
width	Optional parameter indicating the width of a box for a zoomed-in view of the Fry plot near the origin.
from, to	Optional. Subset indices specifying which points of X will be considered when forming the vectors (drawn from each point of from, to each point of to.)
axes	Logical value indicating whether to draw axes, crossing at the origin.
dmax	Maximum distance between points. Pairs at greater distances do not contribute to the result. The default means there is no maximum distance.

Details

The function fryplot generates a Fry plot (or Patterson plot); frypoints returns the points of the Fry plot as a point pattern dataset.

Fry (1979) and Hanna and Fry (1979) introduced a manual graphical method for investigating features of a spatial point pattern of mineral deposits. A transparent sheet, marked with an origin or centre point, is placed over the point pattern. The transparent sheet is shifted so that the origin lies over one of the data points, and the positions of all the *other* data points are copied onto the transparent sheet. This procedure is repeated for each data point in turn. The resulting plot (the Fry

plot) is a pattern of $n(n-1)$ points, where n is the original number of data points. This procedure was previously proposed by Patterson (1934, 1935) for studying inter-atomic distances in crystals, and is also known as a Patterson plot.

The function `fryplot` generates the Fry/Patterson plot. Standard graphical parameters such as `main`, `pch`, `lwd`, `col`, `bg`, `cex` can be used to control the appearance of the plot. To zoom in (to view only a subset of the Fry plot at higher magnification), use the argument `width` to specify the width of a rectangular field of view centred at the origin, or the standard graphical arguments `xlim` and `ylim` to specify another rectangular field of view. (The actual field of view may be slightly larger, depending on the graphics device.)

The function `frypoints` returns the points of the Fry plot as a point pattern object. There may be a large number of points in this pattern, so this function should be used only if further analysis of the Fry plot is required.

Fry plots are particularly useful for recognising anisotropy in regular point patterns. A void around the origin in the Fry plot suggests regularity (inhibition between points) and the shape of the void gives a clue to anisotropy in the pattern. Fry plots are also useful for detecting periodicity or rounding of the spatial coordinates.

In mathematical terms, the Fry plot of a point pattern `X` is simply a plot of the vectors `X[i] - X[j]` connecting all pairs of distinct points in `X`.

The Fry plot is related to the K function (see `Kest`) and the reduced second moment measure (see `Kmeasure`). For example, the number of points in the Fry plot lying within a circle of given radius is an unnormalised and uncorrected version of the K function. The Fry plot has a similar appearance to the plot of the reduced second moment measure `Kmeasure` when the smoothing parameter `sigma` is very small.

The Fry plot does not adjust for the effect of the size and shape of the sampling window. The density of points in the Fry plot tapers off near the edges of the plot. This is an edge effect, a consequence of the bounded sampling window. In geological applications this is usually not important, because interest is focused on the behaviour near the origin where edge effects can be ignored. To correct for the edge effect, use `Kmeasure` or `Kest` or its relatives.

Value

`fryplot` returns `NULL`. `frypoints` returns a point pattern (object of class `"ppp"`).

Author(s)

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

and Rolf Turner `<rolfturner@posteo.net>`

References

Fry, N. (1979) Random point distributions and strain measurement in rocks. *Tectonophysics* **60**, 89–105.

Hanna, S.S. and Fry, N. (1979) A comparison of methods of strain determination in rocks from southwest Dyfed (Pembrokeshire) and adjacent areas. *Journal of Structural Geology* **1**, 155–162.

Patterson, A.L. (1934) A Fourier series method for the determination of the component of inter-atomic distances in crystals. *Physics Reviews* **46**, 372–376.

Patterson, A.L. (1935) A direct method for the determination of the components of inter-atomic distances in crystals. *Zeitschrift fuer Krystallographie* **90**, 517–554.

See Also

Kmeasure, Kest

Examples

```
## unmarked data
fryplot(cells)
Y <- frypoints(cells)

## numerical marks
fryplot(longleaf, width=4, axes=TRUE)

## multitype points
fryplot(amacrine, width=0.2,
                  from=(marks(amacrine) == "on"),
                  chars=c(3,16), cols=2:3,
                  main="Fry plot centred at an On-cell")
points(0,0)
```

---

fv *Create a Function Value Table*

Description

Advanced Use Only. This low-level function creates an object of class "fv" from raw numerical data.

Usage

```
fv(x, argu = "r", ylab = NULL, valu, fmla = NULL, alim = NULL,
   labl = names(x), desc = NULL, unitname = NULL, fname = NULL, yexp = ylab)
```

Arguments

x	A data frame with at least 2 columns containing the values of the function argument and the corresponding values of (one or more versions of) the function.
argu	String. The name of the column of x that contains the values of the function argument.
ylab	Either NULL, or an R language expression representing the mathematical name of the function. See Details.
valu	String. The name of the column of x that should be taken as containing the function values, in cases where a single column is required.
fmla	Either NULL, or a formula specifying the default plotting behaviour. See Details.

alim	Optional. The default range of values of the function argument for which the function will be plotted. Numeric vector of length 2.
labl	Optional. Plot labels for the columns of x. A vector of strings, with one entry for each column of x.
desc	Optional. Descriptions of the columns of x. A vector of strings, with one entry for each column of x.
unitname	Optional. Name of the unit (usually a unit of length) in which the function argument is expressed. Either a single character string, or a vector of two character strings giving the singular and plural forms, respectively.
fname	Optional. The name of the function itself. A character string.
yexp	Optional. Alternative form of ylab more suitable for annotating an axis of the plot. See Details.

Details

This documentation is provided for experienced programmers who want to modify the internal behaviour of **spatstat**. Other users please see fv.object.

The low-level function fv is used to create an object of class "fv" from raw numerical data.

The data frame x contains the numerical data. It should have one column (typically but not necessarily named "r") giving the values of the function argument for which the function has been evaluated; and at least one other column, containing the corresponding values of the function.

Typically there is more than one column of function values. These columns typically give the values of different versions or estimates of the same function, for example, different estimates of the K function obtained using different edge corrections. However they may also contain the values of related functions such as the derivative or hazard rate.

argu specifies the name of the column of x that contains the values of the function argument (typically argu="r" but this is not compulsory).

valu specifies the name of another column that contains the 'recommended' estimate of the function. It will be used to provide function values in those situations where a single column of data is required. For example, envelope computes its simulation envelopes using the recommended value of the summary function.

fmla specifies the default plotting behaviour. It should be a formula, or a string that can be converted to a formula. Variables in the formula are names of columns of x. See plot.fv for the interpretation of this formula.

alim specifies the recommended range of the function argument. This is used in situations where statistical theory or statistical practice indicates that the computed estimates of the function are not trustworthy outside a certain range of values of the function argument. By default, plot.fv will restrict the plot to this range.

fname is a string giving the name of the function itself. For example, the K function would have fname="K".

ylab is a mathematical expression for the function value, used when labelling an axis of the plot, or when printing a description of the function. It should be an R language object. For example the K function's mathematical name $K(r)$ is rendered by ylab=quote(K(r)).

If yexp is present, then ylab will be used only for printing, and yexp will be used for annotating axes in a plot. (Otherwise yexp defaults to ylab). For example the cross-type K function $K_{1,2}(r)$ is rendered by something like ylab=quote(Kcross[1,2](r)) and yexp=quote(Kcross[list(1,2)](r)) to get the most satisfactory behaviour.

(A useful tip: use substitute instead of quote to insert values of variables into an expression, e.g. substitute(Kcross[i,j](r), list(i=42,j=97)) yields the same as quote(Kcross[42, 97](r)).)

labl is a character vector specifying plot labels for each column of x. These labels will appear on the plot axes (in non-default plots), legends and printed output. Entries in labl may contain the string "%s" which will be replaced by fname. For example the border-corrected estimate of the K function has label "%s[bord](r)" which becomes "K[bord](r)".

desc is a character vector containing intelligible explanations of each column of x. Entries in desc may contain the string "%s" which will be replaced by ylab. For example the border correction estimate of the K function has description "border correction estimate of %s".

Value

An object of class "fv", see fv.object.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>.

See Also

See plot.fv for plotting an "fv" object.

See as.function.fv to convert an "fv" object to an R function.

Use cbind.fv to combine several "fv" objects. Use bind.fv to glue additional columns onto an existing "fv" object.

Simple calculations such as arithmetic and mathematical operations can be computed directly. The range of y values of a function f can be computed by typing range(f). These operations are dispatched to Summary.fv, Math.fv and Ops.fv.

Use eval.fv or with.fv for more complicated calculations.

The functions fvnames, fvnames<- allow the user to use standard abbreviations to refer to columns of an "fv" object.

Undocumented functions for modifying an "fv" object include tweak.fv.entry and rebadge.fv.

Examples

```
df <- data.frame(r=seq(0,5,by=0.1))
df <- transform(df, a=pi*r^2, b=3*r^2)
X <- fv(df, "r", quote(A(r)),
            "a", cbind(a, b) ~ r,
            alim=c(0,4),
            labl=c("r", "%s[true](r)", "%s[approx](r)"),
            desc=c("radius of circle",
                   "true area %s",
```

```
                "rough area %s"),
        fname="A")
X
```

`fv.object` *Function Value Table*

Description

A class `"fv"` to support the convenient plotting of several estimates of the same function.

Details

An object of this class is a convenient way of storing and plotting several different estimates of the same function.

It is a data frame with extra attributes indicating the recommended way of plotting the function, and other information.

There are methods for `print` and `plot` for this class.

Objects of class `"fv"` are returned by `Fest`, `Gest`,`Jest`, and `Kest` along with many other functions.

Author(s)

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

and Rolf Turner `<rolfturner@posteo.net>`

See Also

Objects of class `"fv"` are returned by `Fest`, `Gest`,`Jest`, and `Kest` along with many other functions.

See `plot.fv` for plotting an `"fv"` object.

See `as.function.fv` to convert an `"fv"` object to an R function.

Use `cbind.fv` to combine several `"fv"` objects. Use `bind.fv` to glue additional columns onto an existing `"fv"` object.

Undocumented functions for modifying an `"fv"` object include `fvnames`, `fvnames<-`, `tweak.fv.entry` and `rebadge.fv`.

Examples

```
K <- Kest(cells)
class(K)

K  # prints a sensible summary

plot(K)
```

`fvnames` *Abbreviations for Groups of Columns in Function Value Table*

Description

Groups of columns in a function value table (object of class `"fv"`) identified by standard abbreviations.

Usage

```
fvnames(X, a = ".")

fvnames(X, a = ".") <- value
```

Arguments

`X`	Function value table (object of class `"fv"`). See `fv.object`.
`a`	One of the standard abbreviations listed below.
`value`	Character vector containing names of columns of `X`.

Details

An object of class `"fv"` represents a table of values of a function, usually a summary function for spatial data such as the K-function, for which several different statistical estimators may be available. The different estimates are stored as columns of the table.

Auxiliary information carried in the object `X` specifies some columns or groups of columns of this table that should be used for particular purposes. For convenience these groups can be referred to by standard abbreviations which are recognised by various functions in the **spatstat** package, such as `plot.fv`.

These abbreviations are:

`".x"`	the function argument
`".y"`	the recommended value of the function
`"."`	all function values to be plotted by default (in order of plotting)
`".s"`	the upper and lower limits of shading (for envelopes and confidence intervals)
`".a"`	all function values (in column order)

The command `fvnames(X, a)` expands the abbreviation `a` and returns a character vector containing the names of the columns.

The assignment `fvnames(X, a) <- value` changes the definition of the abbreviation `a` to the character string `value` (which should be the name of another column of `X`). The column names of `X` are not changed.

Note that `fvnames(x, ".")` lists the columns of values that will be plotted by default, in the order that they would be plotted, not in order of the column position. The order in which curves are plotted affects the colours and line styles associated with the curves.

Value

For `fvnames`, a character vector.

For `fvnames<-`, the updated object.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

`fv.object`, `plot.fv`

Examples

```
K <- Kest(cells)
fvnames(K, ".y")
fvnames(K, ".y") <- "trans"
```

`G3est` *Nearest Neighbour Distance Distribution Function of a Three-Dimensional Point Pattern*

Description

Estimates the nearest-neighbour distance distribution function $G_3(r)$ from a three-dimensional point pattern.

Usage

```
G3est(X, ..., rmax = NULL, nrval = 128, correction = c("rs", "km", "Hanisch"))
```

Arguments

`X`	Three-dimensional point pattern (object of class `"pp3"`).
`...`	Ignored.
`rmax`	Optional. Maximum value of argument r for which $G_3(r)$ will be estimated.
`nrval`	Optional. Number of values of r for which $G_3(r)$ will be estimated. A large value of `nrval` is required to avoid discretisation effects.
`correction`	Optional. Character vector specifying the edge correction(s) to be applied. See Details.

Details

For a stationary point process Φ in three-dimensional space, the nearest-neighbour function is

$$G_3(r) = P(d^*(x, \Phi) \leq r \mid x \in \Phi)$$

the cumulative distribution function of the distance $d^*(x, \Phi)$ from a typical point x in Φ to its nearest neighbour, i.e. to the nearest *other* point of Φ.

The three-dimensional point pattern `X` is assumed to be a partial realisation of a stationary point process Φ. The nearest neighbour function of Φ can then be estimated using techniques described in the References. For each data point, the distance to the nearest neighbour is computed. The empirical cumulative distribution function of these values, with appropriate edge corrections, is the estimate of $G_3(r)$.

The available edge corrections are:

`"rs"`**:** the reduced sample (aka minus sampling, border correction) estimator (Baddeley et al, 1993)

`"km"`**:** the three-dimensional version of the Kaplan-Meier estimator (Baddeley and Gill, 1997)

`"Hanisch"`**:** the three-dimensional generalisation of the Hanisch estimator (Hanisch, 1984).

Alternatively `correction="all"` selects all options.

Value

A function value table (object of class `"fv"`) that can be plotted, printed or coerced to a data frame containing the function values.

Warnings

A large value of `nrval` is required in order to avoid discretisation effects (due to the use of histograms in the calculation).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rana Moyeed.

References

Baddeley, A.J, Moyeed, R.A., Howard, C.V. and Boyde, A. (1993) Analysis of a three-dimensional point pattern with replication. *Applied Statistics* **42**, 641–668.

Baddeley, A.J. and Gill, R.D. (1997) Kaplan-Meier estimators of interpoint distance distributions for spatial point processes. *Annals of Statistics* **25**, 263–292.

Hanisch, K.-H. (1984) Some remarks on estimators of the distribution function of nearest neighbour distance in stationary spatial point patterns. *Mathematische Operationsforschung und Statistik, series Statistics* **15**, 409–412.

See Also

`pp3` to create a three-dimensional point pattern (object of class `"pp3"`).

`F3est`, `K3est`, `pcf3est` for other summary functions of a three-dimensional point pattern.

`Gest` to estimate the empty space function of point patterns in two dimensions.

Examples

```
X <- rpoispp3(42)
Z <- G3est(X)
if(interactive()) plot(Z)
```

Gcross *Multitype Nearest Neighbour Distance Function (i-to-j)*

Description

For a multitype point pattern, estimate the distribution of the distance from a point of type i to the nearest point of type j.

Usage

```
Gcross(X, i, j, r=NULL, breaks=NULL, ..., correction=c("rs", "km", "han"))
```

Arguments

X	The observed point pattern, from which an estimate of the cross type distance distribution function $G_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of marks(X).
j	The type (mark value) of the points in X to which distances are measured. A character string (or something that will be converted to a character string). Defaults to the second level of marks(X).
r	Optional. Numeric vector. The values of the argument r at which the distribution function $G_{ij}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
breaks	This argument is for internal use only.
...	Ignored.
correction	Optional. Character string specifying the edge correction(s) to be used. Options are "none", "rs", "km", "hanisch" and "best". Alternatively correction="all" selects all options.

Details

This function Gcross and its companions Gdot and Gmulti are generalisations of the function Gest to multitype point patterns.

A multitype point pattern is a spatial pattern of points classified into a finite number of possible "colours" or "types". In the **spatstat** package, a multitype pattern is represented as a single point

pattern object in which the points carry marks, and the mark value attached to each point determines the type of that point.

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to as.ppp. It must be a marked point pattern, and the mark vector X$marks must be a factor. The arguments i and j will be interpreted as levels of the factor X$marks. (Warning: this means that an integer value i=3 will be interpreted as the number 3, **not** the 3rd smallest level).

The "cross-type" (type i to type j) nearest neighbour distance distribution function of a multitype point process is the cumulative distribution function $G_{ij}(r)$ of the distance from a typical random point of the process with type i the nearest point of type j.

An estimate of $G_{ij}(r)$ is a useful summary statistic in exploratory data analysis of a multitype point pattern. If the process of type i points were independent of the process of type j points, then $G_{ij}(r)$ would equal $F_j(r)$, the empty space function of the type j points. For a multitype Poisson point process where the type i points have intensity λ_i, we have

$$G_{ij}(r) = 1 - e^{-\lambda_j \pi r^2}$$

Deviations between the empirical and theoretical G_{ij} curves may suggest dependence between the points of types i and j.

This algorithm estimates the distribution function $G_{ij}(r)$ from the point pattern X. It assumes that X can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in X as Window(X)) may have arbitrary shape. Biases due to edge effects are treated in the same manner as in Gest.

The argument r is the vector of values for the distance r at which $G_{ij}(r)$ should be evaluated. It is also used to determine the breakpoints (in the sense of hist) for the computation of histograms of distances. The reduced-sample and Kaplan-Meier estimators are computed from histogram counts. In the case of the Kaplan-Meier estimator this introduces a discretisation error which is controlled by the fineness of the breakpoints.

First-time users would be strongly advised not to specify r. However, if it is specified, r must satisfy r[1] = 0, and max(r) must be larger than the radius of the largest disc contained in the window. Furthermore, the successive entries of r must be finely spaced.

The algorithm also returns an estimate of the hazard rate function, $\lambda(r)$, of $G_{ij}(r)$. This estimate should be used with caution as $G_{ij}(r)$ is not necessarily differentiable.

The naive empirical distribution of distances from each point of the pattern X to the nearest other point of the pattern, is a biased estimate of G_{ij}. However this is also returned by the algorithm, as it is sometimes useful in other contexts. Care should be taken not to use the uncorrected empirical G_{ij} as if it were an unbiased estimator of G_{ij}.

Value

An object of class "fv" (see fv.object).

Essentially a data frame containing six numeric columns

r	the values of the argument r at which the function $G_{ij}(r)$ has been estimated
rs	the "reduced sample" or "border correction" estimator of $G_{ij}(r)$
han	the Hanisch-style estimator of $G_{ij}(r)$

km	the spatial Kaplan-Meier estimator of $G_{ij}(r)$
hazard	the hazard rate $\lambda(r)$ of $G_{ij}(r)$ by the spatial Kaplan-Meier method
raw	the uncorrected estimate of $G_{ij}(r)$, i.e. the empirical distribution of the distances from each point of type i to the nearest point of type j
theo	the theoretical value of $G_{ij}(r)$ for a marked Poisson process with the same estimated intensity (see below).

Warnings

The arguments `i` and `j` are always interpreted as levels of the factor `X$marks`. They are converted to character strings if they are not already character strings. The value `i=1` does **not** refer to the first level of the factor.

The function G_{ij} does not necessarily have a density.

The reduced sample estimator of G_{ij} is pointwise approximately unbiased, but need not be a valid distribution function; it may not be a nondecreasing function of r. Its range is always within $[0, 1]$.

The spatial Kaplan-Meier estimator of G_{ij} is always nondecreasing but its maximum value may be less than 1.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>.

References

Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.

Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.

Diggle, P. J. (1986). Displaced amacrine cells in the retina of a rabbit : analysis of a bivariate spatial point pattern. *J. Neurosci. Meth.* **18**, 115–125.

Harkness, R.D and Isham, V. (1983) A bivariate spatial point pattern of ants' nests. *Applied Statistics* **32**, 293–303

Lotwick, H. W. and Silverman, B. W. (1982). Methods for analysing spatial processes of several types of points. *J. Royal Statist. Soc. Ser. B* **44**, 406–413.

Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.

Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.

Van Lieshout, M.N.M. and Baddeley, A.J. (1999) Indices of dependence between types in multivariate point patterns. *Scandinavian Journal of Statistics* **26**, 511–532.

See Also

`Gdot`, `Gest`, `Gmulti`

Examples

```
# amacrine cells data
G01 <- Gcross(amacrine)

# equivalent to:

  G01 <- Gcross(amacrine, "off", "on")


plot(G01)

# empty space function of `on' points
if(interactive()) {
   F1 <- Fest(split(amacrine)$on, r = G01$r)
   lines(F1$r, F1$km, lty=3)
}

# synthetic example
pp <- runifpoispp(30)
pp <- pp %mark% factor(sample(0:1, npoints(pp), replace=TRUE))
G <- Gcross(pp, "0", "1")   # note: "0" not 0
```

`Gcross.inhom` *Inhomogeneous Multitype G Cross Function*

Description

For a multitype point pattern, estimate the inhomogeneous version of the cross G function, which is the distribution of the distance from a point of type i to the nearest point of type j, adjusted for spatially varying intensity.

Usage

```
Gcross.inhom(X, i, j,
             lambda = NULL, lambdaI = NULL, lambdaJ = NULL,
             lambdamin = NULL,
             ...,
             r = NULL,
             ReferenceMeasureMarkSetI = NULL,
             ratio = FALSE)
```

Arguments

`X`	The observed point pattern, from which an estimate of the inhomogeneous cross type G function $G_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
`i`	The type (mark value) of the points in `X` from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of `marks(X)`.

j	The type (mark value) of the points in X to which distances are measured. A character string (or something that will be converted to a character string). Defaults to the second level of marks(X).
lambda	Optional. Values of the estimated intensity of the point process. Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the points in X, a fitted point process model (object of class "ppm" or "kppm" or "dppm"), or a function(x,y) which can be evaluated to give the intensity value at any location.
lambdaI	Optional. Values of the estimated intensity of the sub-process of points of type i. Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the type i points in X, a fitted point process model (object of class "ppm" or "kppm" or "dppm"), or a function(x,y) which can be evaluated to give the intensity value at any location.
lambdaJ	Optional. Values of the the estimated intensity of the sub-process of points of type j. Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the type j points in X, a fitted point process model (object of class "ppm" or "kppm" or "dppm"), or a function(x,y) which can be evaluated to give the intensity value at any location.
lambdamin	Optional. The minimum possible value of the intensity over the spatial domain. A positive numerical value.
...	Extra arguments passed to as.mask to control the pixel resolution for the computation.
r	vector of values for the argument r at which the inhomogeneous G function should be evaluated. Not normally given by the user; there is a sensible default.
ReferenceMeasureMarkSetI	Optional. The total measure of the mark set. A positive number.
ratio	Logical value indicating whether to save ratio information.

Details

This is a generalisation of the function Gcross to include an adjustment for spatially inhomogeneous intensity, in a manner similar to the function Ginhom.

The argument lambdaI supplies the values of the intensity of the sub-process of points of type i. It may be either

a pixel image (object of class "im") which gives the values of the type i intensity at all locations in the window containing X;

a numeric vector containing the values of the type i intensity evaluated only at the data points of type i. The length of this vector must equal the number of type i points in X.

a function of the form function(x,y) which can be evaluated to give values of the intensity at any locations.

a fitted point process model (object of class "ppm", "kppm" or "dppm") whose fitted *trend* can be used as the fitted intensity. (If update=TRUE the model will first be refitted to the data X before the trend is computed.)

omitted: if lambdaI is omitted then it will be estimated using a leave-one-out kernel smoother.

If `lambdaI` is omitted, then it will be estimated using a 'leave-one-out' kernel smoother.

Similarly the argument `lambdaJ` should contain estimated values of the intensity of the points of type j. It may be either a pixel image, a numeric vector of length equal to the number of points in `X`, a function, or omitted.

The argument `r` is the vector of values for the distance r at which $G_{ij}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

Value

An object of class `"fv"` (see `fv.object`) containing estimates of the inhomogeneous cross type G function.

Warnings

The argument `i` is interpreted as a level of the factor `X$marks`. It is converted to a character string if it is not already a character string. The value `i=1` does **not** refer to the first level of the factor.

Author(s)

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

References

Cronie, O. and Van Lieshout, M.N.M. (2015) Summary statistics for inhomogeneous marked point processes. *Annals of the Institute of Statistical Mathematics* DOI: 10.1007/s10463-015-0515-z

See Also

`Gcross`, `Ginhom`, `Gcross.inhom`, `Gmulti.inhom`.

Examples

```
X <- rescale(amacrine)
if(interactive() && require(spatstat.model)) {
  ## how to do it normally
  mod <- ppm(X ~ marks * x)
  lam <- fitted(mod, dataonly=TRUE)
  lmin <- min(predict(mod)[["off"]]) * 0.9
} else {
  ## for package testing
  lam <- intensity(X)[as.integer(marks(X))]
  lmin <- intensity(X)[2] * 0.9
}
GC <- Gcross.inhom(X, "on", "off", lambda=lam, lambdamin=lmin)
```

Gdot *Multitype Nearest Neighbour Distance Function (i-to-any)*

Description

For a multitype point pattern, estimate the distribution of the distance from a point of type i to the nearest other point of any type.

Usage

```
Gdot(X, i, r=NULL, breaks=NULL, ..., correction=c("km", "rs", "han"))
```

Arguments

X	The observed point pattern, from which an estimate of the distance distribution function $G_{i\bullet}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of marks(X).
r	Optional. Numeric vector. The values of the argument r at which the distribution function $G_{i\bullet}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
breaks	This argument is for internal use only.
...	Ignored.
correction	Optional. Character string specifying the edge correction(s) to be used. Options are "none", "rs", "km", "hanisch" and "best". Alternatively correction="all" selects all options.

Details

This function Gdot and its companions Gcross and Gmulti are generalisations of the function Gest to multitype point patterns.

A multitype point pattern is a spatial pattern of points classified into a finite number of possible "colours" or "types". In the **spatstat** package, a multitype pattern is represented as a single point pattern object in which the points carry marks, and the mark value attached to each point determines the type of that point.

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to as.ppp. It must be a marked point pattern, and the mark vector X$marks must be a factor. The argument will be interpreted as a level of the factor X$marks. (Warning: this means that an integer value i=3 will be interpreted as the number 3, **not** the 3rd smallest level.)

The "dot-type" (type i to any type) nearest neighbour distance distribution function of a multitype point process is the cumulative distribution function $G_{i\bullet}(r)$ of the distance from a typical random point of the process with type i the nearest other point of the process, regardless of type.

An estimate of $G_{i\bullet}(r)$ is a useful summary statistic in exploratory data analysis of a multitype point pattern. If the type i points were independent of all other points, then $G_{i\bullet}(r)$ would equal $G_{ii}(r)$, the nearest neighbour distance distribution function of the type i points alone. For a multitype Poisson point process with total intensity λ, we have

$$G_{i\bullet}(r) = 1 - e^{-\lambda \pi r^2}$$

Deviations between the empirical and theoretical $G_{i\bullet}$ curves may suggest dependence of the type i points on the other points.

This algorithm estimates the distribution function $G_{i\bullet}(r)$ from the point pattern `X`. It assumes that `X` can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in `X` as `Window(X)`) may have arbitrary shape. Biases due to edge effects are treated in the same manner as in `Gest`.

The argument `r` is the vector of values for the distance r at which $G_{i\bullet}(r)$ should be evaluated. It is also used to determine the breakpoints (in the sense of `hist`) for the computation of histograms of distances. The reduced-sample and Kaplan-Meier estimators are computed from histogram counts. In the case of the Kaplan-Meier estimator this introduces a discretisation error which is controlled by the fineness of the breakpoints.

First-time users would be strongly advised not to specify `r`. However, if it is specified, `r` must satisfy `r[1] = 0`, and `max(r)` must be larger than the radius of the largest disc contained in the window. Furthermore, the successive entries of `r` must be finely spaced.

The algorithm also returns an estimate of the hazard rate function, $\lambda(r)$, of $G_{i\bullet}(r)$. This estimate should be used with caution as $G_{i\bullet}(r)$ is not necessarily differentiable.

The naive empirical distribution of distances from each point of the pattern `X` to the nearest other point of the pattern, is a biased estimate of $G_{i\bullet}$. However this is also returned by the algorithm, as it is sometimes useful in other contexts. Care should be taken not to use the uncorrected empirical $G_{i\bullet}$ as if it were an unbiased estimator of $G_{i\bullet}$.

Value

An object of class `"fv"` (see `fv.object`).

Essentially a data frame containing six numeric columns

`r`	the values of the argument r at which the function $G_{i\bullet}(r)$ has been estimated
`rs`	the "reduced sample" or "border correction" estimator of $G_{i\bullet}(r)$
`han`	the Hanisch-style estimator of $G_{i\bullet}(r)$
`km`	the spatial Kaplan-Meier estimator of $G_{i\bullet}(r)$
`hazard`	the hazard rate $\lambda(r)$ of $G_{i\bullet}(r)$ by the spatial Kaplan-Meier method
`raw`	the uncorrected estimate of $G_{i\bullet}(r)$, i.e. the empirical distribution of the distances from each point of type i to the nearest other point of any type.
`theo`	the theoretical value of $G_{i\bullet}(r)$ for a marked Poisson process with the same estimated intensity (see below).

Warnings

The argument `i` is interpreted as a level of the factor `X$marks`. It is converted to a character string if it is not already a character string. The value `i=1` does **not** refer to the first level of the factor.

The function $G_{i\bullet}$ does not necessarily have a density.

The reduced sample estimator of $G_{i\bullet}$ is pointwise approximately unbiased, but need not be a valid distribution function; it may not be a nondecreasing function of r. Its range is always within $[0, 1]$.

The spatial Kaplan-Meier estimator of $G_{i\bullet}$ is always nondecreasing but its maximum value may be less than 1.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

References

Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.

Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.

Diggle, P. J. (1986). Displaced amacrine cells in the retina of a rabbit : analysis of a bivariate spatial point pattern. *J. Neurosci. Meth.* **18**, 115–125.

Harkness, R.D and Isham, V. (1983) A bivariate spatial point pattern of ants' nests. *Applied Statistics* **32**, 293–303

Lotwick, H. W. and Silverman, B. W. (1982). Methods for analysing spatial processes of several types of points. *J. Royal Statist. Soc. Ser. B* **44**, 406–413.

Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.

Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.

Van Lieshout, M.N.M. and Baddeley, A.J. (1999) Indices of dependence between types in multivariate point patterns. *Scandinavian Journal of Statistics* **26**, 511–532.

See Also

`Gcross`, `Gest`, `Gmulti`

Examples

```
# amacrine cells data
G0. <- Gdot(amacrine, "off")
plot(G0.)

# synthetic example
pp <- runifpoispp(30)
pp <- pp %mark% factor(sample(0:1, npoints(pp), replace=TRUE))
G <- Gdot(pp, "0")
G <- Gdot(pp, 0) # equivalent
```

Gdot.inhom *Inhomogeneous Multitype G Dot Function*

Description

For a multitype point pattern, estimate the inhomogeneous version of the dot G function, which is the distribution of the distance from a point of type i to the nearest other point of any type, adjusted for spatially varying intensity.

Usage

```
Gdot.inhom(X, i,
           lambdaI = NULL, lambdadot = NULL, lambdamin = NULL,
           ...,
           r = NULL, ReferenceMeasureMarkSetI = NULL, ratio = FALSE)
```

Arguments

X	The observed point pattern, from which an estimate of the inhomogeneous dot type G function $G_{i\bullet}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of marks(X).
lambdaI	Optional. Values of the estimated intensity of the sub-process of points of type i. Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the type i points in X, a fitted point process model (object of class "ppm" or "kppm" or "dppm"), or a function(x,y) which can be evaluated to give the intensity value at any location.
lambdadot	Optional. Values of the estimated intensity of the entire point process, Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the points in X, a fitted point process model (object of class "ppm" or "kppm" or "dppm"), or a function(x,y) which can be evaluated to give the intensity value at any location.
lambdamin	Optional. The minimum possible value of the intensity over the spatial domain. A positive numerical value.
...	Ignored.
r	vector of values for the argument r at which the inhomogeneous dot type G function $G_{i\bullet}(r)$ should be evaluated. Not normally given by the user; there is a sensible default.
ReferenceMeasureMarkSetI	Optional. The total measure of the mark set. A positive number.
ratio	Logical value indicating whether to save ratio information.

Details

This is a generalisation of the function Gdot to include an adjustment for spatially inhomogeneous intensity, in a manner similar to the function Ginhom.

The argument lambdaI supplies the values of the intensity of the sub-process of points of type i. It may be either

a pixel image (object of class "im") which gives the values of the type i intensity at all locations in the window containing X;

a numeric vector containing the values of the type i intensity evaluated only at the data points of type i. The length of this vector must equal the number of type i points in X.

a function of the form function(x,y) which can be evaluated to give values of the intensity at any locations.

a fitted point process model (object of class "ppm", "kppm" or "dppm") whose fitted *trend* can be used as the fitted intensity. (If update=TRUE the model will first be refitted to the data X before the trend is computed.)

omitted: if lambdaI is omitted then it will be estimated using a leave-one-out kernel smoother.

If lambdaI is omitted, then it will be estimated using a 'leave-one-out' kernel smoother.

Similarly the argument lambdadot should contain estimated values of the intensity of the entire point process. It may be either a pixel image, a numeric vector of length equal to the number of points in X, a function, or omitted.

The argument r is the vector of values for the distance r at which $G_{i\bullet}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

Value

An object of class "fv" (see fv.object) containing estimates of the inhomogeneous dot type G function.

Warnings

The argument i is interpreted as a level of the factor X$marks. It is converted to a character string if it is not already a character string. The value i=1 does **not** refer to the first level of the factor.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>

References

Cronie, O. and Van Lieshout, M.N.M. (2015) Summary statistics for inhomogeneous marked point processes. *Annals of the Institute of Statistical Mathematics* DOI: 10.1007/s10463-015-0515-z

See Also

Gdot, Ginhom, Gcross.inhom, Gmulti.inhom.

Examples

```
X <- rescale(amacrine)
if(interactive() && require(spatstat.model)) {
  ## how to do it normally
  mod <- ppm(X ~ marks * x)
  lam <- fitted(mod, dataonly=TRUE)
  lmin <- min(predict(mod)[["off"]]) * 0.9
} else {
  ## for package testing
  lam <- intensity(X)[as.integer(marks(X))]
  lmin <- intensity(X)[2] * 0.9
}
lamI <- lam[marks(X) == "on"]
GD <- Gdot.inhom(X, "on", lambdaI=lamI, lambdadot=lam, lambdamin=lmin)
```

Gest *Nearest Neighbour Distance Function G*

Description

Estimates the nearest neighbour distance distribution function $G(r)$ from a point pattern in a window of arbitrary shape.

Usage

```
Gest(X, r=NULL, breaks=NULL, ...,
     correction=c("rs", "km", "han"),
     domain=NULL)
```

Arguments

X	The observed point pattern, from which an estimate of $G(r)$ will be computed. An object of class ppp, or data in any format acceptable to as.ppp().
r	Optional. Numeric vector. The values of the argument r at which $G(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
breaks	This argument is for internal use only.
...	Ignored.
correction	Optional. The edge correction(s) to be used to estimate $G(r)$. A vector of character strings selected from "none", "rs", "km", "Hanisch" and "best". Alternatively correction="all" selects all options.
domain	Optional. Calculations will be restricted to this subset of the window. See Details.

Details

The nearest neighbour distance distribution function (also called the "*event-to-event*" or "*inter-event*" distribution) of a point process X is the cumulative distribution function G of the distance from a typical random point of X to the nearest other point of X.

An estimate of G derived from a spatial point pattern dataset can be used in exploratory data analysis and formal inference about the pattern (Cressie, 1991; Diggle, 1983; Ripley, 1988). In exploratory analyses, the estimate of G is a useful statistic summarising one aspect of the "clustering" of points. For inferential purposes, the estimate of G is usually compared to the true value of G for a completely random (Poisson) point process, which is

$$G(r) = 1 - e^{-\lambda \pi r^2}$$

where λ is the intensity (expected number of points per unit area). Deviations between the empirical and theoretical G curves may suggest spatial clustering or spatial regularity.

This algorithm estimates the nearest neighbour distance distribution function G from the point pattern `X`. It assumes that `X` can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in `X` as `Window(X)`) may have arbitrary shape.

The argument `X` is interpreted as a point pattern object (of class `"ppp"`, see `ppp.object`) and can be supplied in any of the formats recognised by `as.ppp()`.

The estimation of G is hampered by edge effects arising from the unobservability of points of the random pattern outside the window. An edge correction is needed to reduce bias (Baddeley, 1998; Ripley, 1988). The edge corrections implemented here are the border method or "*reduced sample*" estimator, the spatial Kaplan-Meier estimator (Baddeley and Gill, 1997) and the Hanisch estimator (Hanisch, 1984).

The argument `r` is the vector of values for the distance r at which $G(r)$ should be evaluated. It is also used to determine the breakpoints (in the sense of `hist`) for the computation of histograms of distances. The estimators are computed from histogram counts. This introduces a discretisation error which is controlled by the fineness of the breakpoints.

First-time users would be strongly advised not to specify `r`. However, if it is specified, `r` must satisfy `r[1] = 0`, and `max(r)` must be larger than the radius of the largest disc contained in the window. Furthermore, the successive entries of `r` must be finely spaced.

The algorithm also returns an estimate of the hazard rate function, $\lambda(r)$, of $G(r)$. The hazard rate is defined as the derivative

$$\lambda(r) = -\frac{d}{dr} \log(1 - G(r))$$

This estimate should be used with caution as G is not necessarily differentiable.

If the argument `domain` is given, the estimate of $G(r)$ will be based only on the nearest neighbour distances measured from points falling inside `domain` (although their nearest neighbours may lie outside `domain`). This is useful in bootstrap techniques. The argument `domain` should be a window (object of class `"owin"`) or something acceptable to `as.owin`. It must be a subset of the window of the point pattern `X`.

The naive empirical distribution of distances from each point of the pattern `X` to the nearest other point of the pattern, is a biased estimate of G. However it is sometimes useful. It can be returned by the algorithm, by selecting `correction="none"`. Care should be taken not to use the uncorrected empirical G as if it were an unbiased estimator of G.

To simply compute the nearest neighbour distance for each point in the pattern, use nndist. To determine which point is the nearest neighbour of a given point, use nnwhich.

Value

An object of class "fv", see fv.object, which can be plotted directly using plot.fv.

Essentially a data frame containing some or all of the following columns:

r	the values of the argument r at which the function $G(r)$ has been estimated
rs	the "reduced sample" or "border correction" estimator of $G(r)$
km	the spatial Kaplan-Meier estimator of $G(r)$
hazard	the hazard rate $\lambda(r)$ of $G(r)$ by the spatial Kaplan-Meier method
raw	the uncorrected estimate of $G(r)$, i.e. the empirical distribution of the distances from each point in the pattern X to the nearest other point of the pattern
han	the Hanisch correction estimator of $G(r)$
theo	the theoretical value of $G(r)$ for a stationary Poisson process of the same estimated intensity.

Warnings

The function G does not necessarily have a density. Any valid c.d.f. may appear as the nearest neighbour distance distribution function of a stationary point process.

The reduced sample estimator of G is pointwise approximately unbiased, but need not be a valid distribution function; it may not be a nondecreasing function of r. Its range is always within $[0, 1]$.

The spatial Kaplan-Meier estimator of G is always nondecreasing but its maximum value may be less than 1.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

References

Baddeley, A.J. Spatial sampling and censoring. In O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*. Chapman and Hall, 1998. Chapter 2, pages 37-78.

Baddeley, A.J. and Gill, R.D. Kaplan-Meier estimators of interpoint distance distributions for spatial point processes. *Annals of Statistics* **25** (1997) 263-292.

Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.

Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.

Hanisch, K.-H. (1984) Some remarks on estimators of the distribution function of nearest-neighbour distance in stationary spatial point patterns. *Mathematische Operationsforschung und Statistik, series Statistics* **15**, 409–412.

Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.

Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.

See Also

nndist, nnwhich, Fest, Jest, Kest, km.rs, reduced.sample, kaplan.meier

Examples

```
G <- Gest(cells)
plot(G)

# P-P style plot
plot(G, cbind(km,theo) ~ theo)

# the empirical G is below the Poisson G,
# indicating an inhibited pattern

if(interactive()) {
   plot(G, . ~ r)
   plot(G, . ~ theo)
   plot(G, asin(sqrt(.)) ~ asin(sqrt(theo)))
}
```

Gfox *Foxall's Distance Functions*

Description

Given a point pattern X and a spatial object Y, compute estimates of Foxall's G and J functions.

Usage

```
Gfox(X, Y, r=NULL, breaks=NULL, correction=c("km", "rs", "han"), W, ...)
Jfox(X, Y, r=NULL, breaks=NULL, correction=c("km", "rs", "han"), W, ...,
     warn.trim=TRUE)
```

Arguments

X	A point pattern (object of class "ppp") from which distances will be measured.
Y	An object of class "ppp", "psp" or "owin" to which distances will be measured. Alternatively a pixel image (class "im") with logical values.
r	Optional. Numeric vector. The values of the argument r at which $Gfox(r)$ or $Jfox(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
breaks	This argument is for internal use only.
correction	Optional. The edge correction(s) to be used to estimate $Gfox(r)$ or $Jfox(r)$. A vector of character strings selected from "none", "rs", "km", "cs" and "best". Alternatively correction="all" selects all options.

W	Optional. A window (object of class "owin") to be taken as the window of observation. The distribution function will be estimated from data inside W. The default is W=Frame(Y) when Y is a window, and W=Window(Y) otherwise.
...	Extra arguments affecting the discretisation of distances. These arguments are ignored by Gfox, but Jfox passes them to Hest to determine the discretisation of the spatial domain.
warn.trim	Logical value indicating whether a warning should be issued by Jfox when the window of X had to be trimmed in order to be a subset of the frame of Y.

Details

Given a point pattern X and another spatial object Y, these functions compute two nonparametric measures of association between X and Y, introduced by Foxall (Foxall and Baddeley, 2002).

Let the random variable R be the distance from a typical point of X to the object Y. Foxall's G-function is the cumulative distribution function of R:

$$G(r) = P(R \leq r)$$

Let the random variable S be the distance from a *fixed* point in space to the object Y. The cumulative distribution function of S is the (unconditional) spherical contact distribution function

$$H(r) = P(S \leq r)$$

which is computed by Hest.

Foxall's J-function is the ratio

$$J(r) = \frac{1 - G(r)}{1 - H(r)}$$

For further interpretation, see Foxall and Baddeley (2002).

Accuracy of Jfox depends on the pixel resolution, which is controlled by the arguments eps, dimyx and xy passed to as.mask. For example, use eps=0.1 to specify square pixels of side 0.1 units, and dimyx=256 to specify a 256 by 256 grid of pixels.

Value

A function value table (object of class "fv") which can be printed, plotted, or converted to a data frame of values.

Author(s)

Rob Foxall and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

References

Foxall, R. and Baddeley, A. (2002) Nonparametric measures of association between a spatial point process and a random set, with geological applications. *Applied Statistics* **51**, 165–182.

See Also

Gest, Hest, Jest, Fest

Examples

```
X <- copper$SouthPoints
Y <- copper$SouthLines
G <- Gfox(X,Y)
J <- Jfox(X,Y, correction="km")
```

Ginhom *Inhomogeneous Nearest Neighbour Function*

Description

Estimates the inhomogeneous nearest neighbour function G of a non-stationary point pattern.

Usage

```
Ginhom(X, lambda = NULL, lmin = NULL, ...,
       sigma = NULL, varcov = NULL,
       r = NULL, breaks = NULL, ratio = FALSE,
       update = TRUE, warn.bias=TRUE, savelambda=FALSE)
```

Arguments

X	The observed data point pattern, from which an estimate of the inhomogeneous G function will be computed. An object of class "ppp" or in a format recognised by as.ppp()
lambda	Optional. Values of the estimated intensity function. Either a vector giving the intensity values at the points of the pattern X, a pixel image (object of class "im") giving the intensity values at all locations, a fitted point process model (object of class "ppm") or a function(x,y) which can be evaluated to give the intensity value at any location.
lmin	Optional. The minimum possible value of the intensity over the spatial domain. A positive numerical value.
sigma, varcov	Optional arguments passed to density.ppp to control the smoothing bandwidth, when lambda is estimated by kernel smoothing.
...	Extra arguments passed to as.mask to control the pixel resolution, or passed to density.ppp to control the smoothing bandwidth.
r	vector of values for the argument r at which the inhomogeneous K function should be evaluated. Not normally given by the user; there is a sensible default.
breaks	This argument is for internal use only.
ratio	Logical. If TRUE, the numerator and denominator of the estimate will also be saved, for use in analysing replicated point patterns.
update	Logical. If lambda is a fitted model (class "ppm" or "kppm") and update=TRUE (the default), the model will first be refitted to the data X (using update.ppm or update.kppm) before the fitted intensity is computed. If update=FALSE, the fitted intensity of the model will be computed without fitting it to X.

warn.bias — Logical value specifying whether to issue a warning when the inhomogeneity correction factor takes extreme values, which can often lead to biased results. This usually occurs when insufficient smoothing is used to estimate the intensity.

savelambda — Logical value specifying whether to save the values of lmin and lambda as attributes of the result.

Details

This command computes estimates of the inhomogeneous G-function (van Lieshout, 2010) of a point pattern. It is the counterpart, for inhomogeneous spatial point patterns, of the nearest-neighbour distance distribution function G for homogeneous point patterns computed by Gest.

The argument X should be a point pattern (object of class "ppp").

The inhomogeneous G function is computed using the border correction, equation (7) in Van Lieshout (2010).

The argument lambda should supply the (estimated) values of the intensity function λ of the point process. It may be either

a numeric vector containing the values of the intensity function at the points of the pattern X.

a pixel image (object of class "im") assumed to contain the values of the intensity function at all locations in the window.

a fitted point process model (object of class "ppm" or "kppm") whose fitted *trend* can be used as the fitted intensity. (If update=TRUE the model will first be refitted to the data X before the trend is computed.)

a function which can be evaluated to give values of the intensity at any locations.

omitted: if lambda is omitted, then it will be estimated using a 'leave-one-out' kernel smoother.

If lambda is a numeric vector, then its length should be equal to the number of points in the pattern X. The value lambda[i] is assumed to be the the (estimated) value of the intensity $\lambda(x_i)$ for the point x_i of the pattern X. Each value must be a positive number; NA's are not allowed.

If lambda is a pixel image, the domain of the image should cover the entire window of the point pattern. If it does not (which may occur near the boundary because of discretisation error), then the missing pixel values will be obtained by applying a Gaussian blur to lambda using blur, then looking up the values of this blurred image for the missing locations. (A warning will be issued in this case.)

If lambda is a function, then it will be evaluated in the form lambda(x,y) where x and y are vectors of coordinates of the points of X. It should return a numeric vector with length equal to the number of points in X.

If lambda is omitted, then it will be estimated using a 'leave-one-out' kernel smoother. The estimate lambda[i] for the point X[i] is computed by removing X[i] from the point pattern, applying kernel smoothing to the remaining points using density.ppp, and evaluating the smoothed intensity at the point X[i]. The smoothing kernel bandwidth is controlled by the arguments sigma and varcov, which are passed to density.ppp along with any extra arguments.

Value

An object of class "fv", see fv.object, which can be plotted directly using plot.fv.

Author(s)

Original code by Marie-Colette van Lieshout. C implementation and R adaptation by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Ege Rubak <rubak@math.aau.dk>.

References

Van Lieshout, M.N.M. and Baddeley, A.J. (1996) A nonparametric measure of spatial interaction in point patterns. *Statistica Neerlandica* **50**, 344–361.

Van Lieshout, M.N.M. (2010) A J-function for inhomogeneous point processes. *Statistica Neerlandica* **65**, 183–201.

See Also

Finhom, Jinhom, Gest

Examples

```
plot(Ginhom(swedishpines, sigma=10))


  plot(Ginhom(swedishpines, sigma=bw.diggle, adjust=2))
```

Gmulti *Marked Nearest Neighbour Distance Function*

Description

For a marked point pattern, estimate the distribution of the distance from a typical point in subset I to the nearest point of subset J.

Usage

```
Gmulti(X, I, J, r=NULL, breaks=NULL, ...,
       disjoint=NULL, correction=c("rs", "km", "han"))
```

Arguments

X	The observed point pattern, from which an estimate of the multitype distance distribution function $G_{IJ}(r)$ will be computed. It must be a marked point pattern. See under Details.
I	Subset of points of X from which distances are measured.
J	Subset of points in X to which distances are measured.
r	Optional. Numeric vector. The values of the argument r at which the distribution function $G_{IJ}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.

breaks	This argument is for internal use only.
...	Ignored.
disjoint	Optional flag indicating whether the subsets I and J are disjoint. If missing, this value will be computed by inspecting the vectors I and J.
correction	Optional. Character string specifying the edge correction(s) to be used. Options are "none", "rs", "km", "hanisch" and "best". Alternatively correction="all" selects all options.

Details

The function Gmulti generalises Gest (for unmarked point patterns) and Gdot and Gcross (for multitype point patterns) to arbitrary marked point patterns.

Suppose X_I, X_J are subsets, possibly overlapping, of a marked point process. This function computes an estimate of the cumulative distribution function $G_{IJ}(r)$ of the distance from a typical point of X_I to the nearest distinct point of X_J.

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to as.ppp.

The arguments I and J specify two subsets of the point pattern. They may be any type of subset indices, for example, logical vectors of length equal to npoints(X), or integer vectors with entries in the range 1 to npoints(X), or negative integer vectors.

Alternatively, I and J may be **functions** that will be applied to the point pattern X to obtain index vectors. If I is a function, then evaluating I(X) should yield a valid subset index. This option is useful when generating simulation envelopes using envelope.

This algorithm estimates the distribution function $G_{IJ}(r)$ from the point pattern X. It assumes that X can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in X as Window(X)) may have arbitrary shape. Biases due to edge effects are treated in the same manner as in Gest.

The argument r is the vector of values for the distance r at which $G_{IJ}(r)$ should be evaluated. It is also used to determine the breakpoints (in the sense of hist) for the computation of histograms of distances. The reduced-sample and Kaplan-Meier estimators are computed from histogram counts. In the case of the Kaplan-Meier estimator this introduces a discretisation error which is controlled by the fineness of the breakpoints.

First-time users would be strongly advised not to specify r. However, if it is specified, r must satisfy r[1] = 0, and max(r) must be larger than the radius of the largest disc contained in the window. Furthermore, the successive entries of r must be finely spaced.

The algorithm also returns an estimate of the hazard rate function, $\lambda(r)$, of $G_{IJ}(r)$. This estimate should be used with caution as $G_{IJ}(r)$ is not necessarily differentiable.

The naive empirical distribution of distances from each point of the pattern X to the nearest other point of the pattern, is a biased estimate of G_{IJ}. However this is also returned by the algorithm, as it is sometimes useful in other contexts. Care should be taken not to use the uncorrected empirical G_{IJ} as if it were an unbiased estimator of G_{IJ}.

Value

An object of class "fv" (see fv.object).

Essentially a data frame containing six numeric columns

r	the values of the argument r at which the function $G_{IJ}(r)$ has been estimated
rs	the "reduced sample" or "border correction" estimator of $G_{IJ}(r)$
han	the Hanisch-style estimator of $G_{IJ}(r)$
km	the spatial Kaplan-Meier estimator of $G_{IJ}(r)$
hazard	the hazard rate $\lambda(r)$ of $G_{IJ}(r)$ by the spatial Kaplan-Meier method
raw	the uncorrected estimate of $G_{IJ}(r)$, i.e. the empirical distribution of the distances from each point of type i to the nearest point of type j
theo	the theoretical value of $G_{IJ}(r)$ for a marked Poisson process with the same estimated intensity

Warnings

The function G_{IJ} does not necessarily have a density.

The reduced sample estimator of G_{IJ} is pointwise approximately unbiased, but need not be a valid distribution function; it may not be a nondecreasing function of r. Its range is always within $[0, 1]$.

The spatial Kaplan-Meier estimator of G_{IJ} is always nondecreasing but its maximum value may be less than 1.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.

Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.

Diggle, P. J. (1986). Displaced amacrine cells in the retina of a rabbit : analysis of a bivariate spatial point pattern. *J. Neurosci. Meth.* **18**, 115–125.

Harkness, R.D and Isham, V. (1983) A bivariate spatial point pattern of ants' nests. *Applied Statistics* **32**, 293–303

Lotwick, H. W. and Silverman, B. W. (1982). Methods for analysing spatial processes of several types of points. *J. Royal Statist. Soc. Ser. B* **44**, 406–413.

Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.

Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.

Van Lieshout, M.N.M. and Baddeley, A.J. (1999) Indices of dependence between types in multivariate point patterns. *Scandinavian Journal of Statistics* **26**, 511–532.

See Also

Gcross, Gdot, Gest

Examples

```
trees <- longleaf
 # Longleaf Pine data: marks represent diameter

Gm <- Gmulti(trees, marks(trees) <= 15, marks(trees) >= 25)
plot(Gm)
```

GmultiInhom *Inhomogeneous Marked G-Function*

Description

For a marked point pattern, estimate the inhomogeneous version of the multitype G function, effectively the cumulative distribution function of the distance from a point in subset I to the nearest point in subset J, adjusted for spatially varying intensity.

Usage

```
Gmulti.inhom(X, I, J,
           lambda = NULL, lambdaI = NULL, lambdaJ = NULL,
           lambdamin = NULL, ...,
           r = NULL,
           ReferenceMeasureMarkSetI = NULL,
           ratio = FALSE)

GmultiInhom(X, I, J,
           lambda = NULL, lambdaI = NULL, lambdaJ = NULL,
           lambdamin = NULL, ...,
           r = NULL,
           ReferenceMeasureMarkSetI = NULL,
           ratio = FALSE)
```

Arguments

X	A spatial point pattern (object of class "ppp".
I	A subset index specifying the subset of points *from* which distances are measured. Any kind of subset index acceptable to [.ppp.
J	A subset index specifying the subset of points *to* which distances are measured. Any kind of subset index acceptable to [.ppp.
lambda	Intensity estimates for each point of X. A numeric vector of length equal to npoints(X). Incompatible with lambdaI,lambdaJ.

lambdaI	Intensity estimates for each point of X[I]. A numeric vector of length equal to npoints(X[I]). Incompatible with lambda.
lambdaJ	Intensity estimates for each point of X[J]. A numeric vector of length equal to npoints(X[J]). Incompatible with lambda.
lambdamin	A lower bound for the intensity, or at least a lower bound for the values in lambdaJ or lambda[J].
...	Ignored.
r	Vector of distance values at which the inhomogeneous G function should be estimated. There is a sensible default.
ReferenceMeasureMarkSetI	Optional. The total measure of the mark set. A positive number.
ratio	Logical value indicating whether to save ratio information.

Details

See Cronie and Van Lieshout (2015).

The functions GmultiInhom and Gmulti.inhom are identical.

Value

Object of class "fv" containing the estimate of the inhomogeneous multitype G function.

Author(s)

Ottmar Cronie and Marie-Colette van Lieshout. Rewritten for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu

References

Cronie, O. and Van Lieshout, M.N.M. (2015) Summary statistics for inhomogeneous marked point processes. *Annals of the Institute of Statistical Mathematics* DOI: 10.1007/s10463-015-0515-z

See Also

Ginhom, Gmulti

Examples

```
X <- rescale(amacrine)
I <- (marks(X) == "on")
J <- (marks(X) == "off")
if(interactive() && require(spatstat.model)) {
  ## how to do it normally
  mod <- ppm(X ~ marks * x)
  lam <- fitted(mod, dataonly=TRUE)
  lmin <- min(predict(mod)[["off"]]) * 0.9
} else {
  ## for package testing
  lam <- intensity(X)[as.integer(marks(X))]
  lmin <- intensity(X)[2] * 0.9
```

```
}
plot(GmultiInhom(X, I, J, lambda=lam, lambdamin=lmin))
# equivalent
plot(GmultiInhom(X, I, J, lambdaI=lam[I], lambdaJ=lam[J], lambdamin=lmin),
     main="")
```

`harmonise.fv` *Make Function Tables Compatible*

Description

Convert several objects of class `"fv"` to the same values of the function argument.

Usage

```
## S3 method for class 'fv'
harmonise(..., strict=FALSE)

## S3 method for class 'fv'
harmonize(..., strict=FALSE)
```

Arguments

`...`	Any number of function tables (objects of class `"fv"`).
`strict`	Logical. If `TRUE`, a column of data will be deleted if columns of the same name do not appear in every object.

Details

A function value table (object of class `"fv"`) is essentially a data frame giving the values of a function $f(x)$ (or several alternative estimates of this value) at equally-spaced values of the function argument x.

The command `harmonise` is generic. This is the method for objects of class `"fv"`.

This command makes any number of `"fv"` objects compatible, in the loose sense that they have the same sequence of values of x. They can then be combined by `cbind.fv`, but not necessarily by `eval.fv`.

All arguments `...` must be function value tables (objects of class `"fv"`). The result will be a list, of length equal to the number of arguments `...`, containing new versions of each of these functions, converted to a common sequence of x values. If the arguments were named (`name=value`) then the return value also carries these names.

The range of x values in the resulting functions will be the intersection of the ranges of x values in the original functions. The spacing of x values in the resulting functions will be the finest (narrowest) of the spacings of the x values in the original functions. Function values are interpolated using `approxfun`.

If strict=TRUE, each column of data will be retained only if a column of the same name appears in all of the arguments .... This ensures that the resulting objects are strictly compatible in the sense of compatible.fv, and can be combined using eval.fv or collapse.fv.

If strict=FALSE (the default), this does not occur, and then the resulting objects are **not** guaranteed to be compatible in the sense of compatible.fv.

Value

A list, of length equal to the number of arguments ..., whose entries are objects of class "fv". If the arguments were named (name=value) then the return value also carries these names.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

fv.object, cbind.fv, eval.fv, compatible.fv

Examples

```
H <- harmonise(K=Kest(cells), G=Gest(cells))
H
```

Hest *Spherical Contact Distribution Function*

Description

Estimates the spherical contact distribution function of a random set.

Usage

```
Hest(X, r=NULL, breaks=NULL, ...,
     W,
     correction=c("km", "rs", "han"),
     conditional=TRUE)
```

Arguments

X	The observed random set. An object of class "ppp", "psp" or "owin". Alternatively a pixel image (class "im") with logical values.
r	Optional. Vector of values for the argument r at which $H(r)$ should be evaluated. Users are advised *not* to specify this argument; there is a sensible default.
breaks	This argument is for internal use only.
...	Arguments passed to as.mask to control the discretisation.

`W`	Optional. A window (object of class `"owin"`) to be taken as the window of observation. The contact distribution function will be estimated from values of the contact distance inside `W`. The default is `W=Frame(X)` when `X` is a window, and `W=Window(X)` otherwise.
`correction`	Optional. The edge correction(s) to be used to estimate $H(r)$. A vector of character strings selected from `"none"`, `"rs"`, `"km"`, `"han"` and `"best"`. Alternatively `correction="all"` selects all options.
`conditional`	Logical value indicating whether to compute the conditional or unconditional distribution. See Details.

Details

The spherical contact distribution function of a stationary random set X is the cumulative distribution function H of the distance from a fixed point in space to the nearest point of X, given that the point lies outside X. That is, $H(r)$ equals the probability that `X` lies closer than r units away from the fixed point x, given that `X` does not cover x.

Let $D = d(x, X)$ be the shortest distance from an arbitrary point x to the set `X`. Then the spherical contact distribution function is

$$H(r) = P(D \leq r \mid D > 0)$$

For a point process, the spherical contact distribution function is the same as the empty space function F discussed in `Fest`.

The argument `X` may be a point pattern (object of class `"ppp"`), a line segment pattern (object of class `"psp"`) or a window (object of class `"owin"`). It is assumed to be a realisation of a stationary random set.

The algorithm first calls `distmap` to compute the distance transform of `X`, then computes the Kaplan-Meier and reduced-sample estimates of the cumulative distribution following Hansen et al (1999). If `conditional=TRUE` (the default) the algorithm returns an estimate of the spherical contact function $H(r)$ as defined above. If `conditional=FALSE`, it instead returns an estimate of the cumulative distribution function $H^*(r) = P(D \leq r)$ which includes a jump at $r = 0$ if `X` has nonzero area.

Accuracy depends on the pixel resolution, which is controlled by the arguments `eps`, `dimyx` and `xy` passed to `as.mask`. For example, use `eps=0.1` to specify square pixels of side 0.1 units, and `dimyx=256` to specify a 256 by 256 grid of pixels.

Value

An object of class `"fv"`, see `fv.object`, which can be plotted directly using `plot.fv`.

Essentially a data frame containing up to six columns:

`r`	the values of the argument r at which the function $H(r)$ has been estimated
`rs`	the "reduced sample" or "border correction" estimator of $H(r)$
`km`	the spatial Kaplan-Meier estimator of $H(r)$
`hazard`	the hazard rate $\lambda(r)$ of $H(r)$ by the spatial Kaplan-Meier method
`han`	the spatial Hanisch-Chiu-Stoyan estimator of $H(r)$
`raw`	the uncorrected estimate of $H(r)$, i.e. the empirical distribution of the distance from a fixed point in the window to the nearest point of `X`

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk> with contributions from Kassel Hingee.

References

Baddeley, A.J. Spatial sampling and censoring. In O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*. Chapman and Hall, 1998. Chapter 2, pages 37-78.

Baddeley, A.J. and Gill, R.D. The empty space hazard of a spatial pattern. Research Report 1994/3, Department of Mathematics, University of Western Australia, May 1994.

Hansen, M.B., Baddeley, A.J. and Gill, R.D. First contact distributions for spatial patterns: regularity and estimation. *Advances in Applied Probability* **31** (1999) 15-33.

Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.

Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.

See Also

Fest

Examples

```
X <- runifpoint(42)
H <- Hest(X)
Y <- rpoisline(10)
H <- Hest(Y)
H <- Hest(Y, dimyx=256)
X <- heather$coarse
plot(Hest(X))
H <- Hest(X, conditional=FALSE)

P <- owin(poly=list(x=c(5.3, 8.5, 8.3, 3.7, 1.3, 3.7),
                    y=c(9.7, 10.0, 13.6, 14.4, 10.7, 7.2)))
plot(X)
plot(P, add=TRUE, col="red")
H <- Hest(X, W=P)
Z <- as.im(FALSE, Frame(X))
Z[X] <- TRUE
Z <- Z[P, drop=FALSE]
plot(Z)
H <- Hest(Z)
```

hopskel *Hopkins-Skellam Test*

Description

Perform the Hopkins-Skellam test of Complete Spatial Randomness, or simply calculate the test statistic.

Usage

```
hopskel(X)

hopskel.test(X, ...,
             alternative=c("two.sided", "less", "greater",
                           "clustered", "regular"),
             method=c("asymptotic", "MonteCarlo"),
             nsim=999)
```

Arguments

X	Point pattern (object of class "ppp").
alternative	String indicating the type of alternative for the hypothesis test. Partially matched.
method	Method of performing the test. Partially matched.
nsim	Number of Monte Carlo simulations to perform, if a Monte Carlo p-value is required.
...	Ignored.

Details

Hopkins and Skellam (1954) proposed a test of Complete Spatial Randomness based on comparing nearest-neighbour distances with point-event distances.

If the point pattern X contains n points, we first compute the nearest-neighbour distances $P_1, \ldots, P_n$ so that P_i is the distance from the ith data point to the nearest other data point. Then we generate another completely random pattern U with the same number n of points, and compute for each point of U the distance to the nearest point of X, giving distances $I_1, \ldots, I_n$. The test statistic is

$$A = \frac{\sum_i P_i^2}{\sum_i I_i^2}$$

The null distribution of A is roughly an F distribution with shape parameters $(2n, 2n)$. (This is equivalent to using the test statistic $H = A/(1 + A)$ and referring H to the Beta distribution with parameters (n, n)).

The function hopskel calculates the Hopkins-Skellam test statistic A, and returns its numeric value. This can be used as a simple summary of spatial pattern: the value $H = 1$ is consistent with Complete Spatial Randomness, while values $H < 1$ are consistent with spatial clustering, and values $H > 1$ are consistent with spatial regularity.

The function `hopskel.test` performs the test. If `method="asymptotic"` (the default), the test statistic H is referred to the F distribution. If `method="MonteCarlo"`, a Monte Carlo test is performed using `nsim` simulated point patterns.

Value

The value of `hopskel` is a single number.

The value of `hopskel.test` is an object of class `"htest"` representing the outcome of the test. It can be printed.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

Hopkins, B. and Skellam, J.G. (1954) A new method of determining the type of distribution of plant individuals. *Annals of Botany* **18**, 213–227.

See Also

`clarkevans`, `clarkevans.test`, `nndist`, `nncross`

Examples

```
hopskel(redwood)
hopskel.test(redwood, alternative="clustered")
```

---

hotbox *Heat Kernel for a Two-Dimensional Rectangle*

Description

Calculate values of the heat kernel in a rectangle with insulated edges.

Usage

```
hotbox(Xsource, Xquery, sigma,
       ..., W=NULL, squared=FALSE, nmax=20)
```

Arguments

Xsource	Point pattern of sources of heat. Object of class "ppp" or convertible to a point pattern using as.ppp(Xsource, W).
Xquery	Locations where the heat kernel value is required. An object of class "ppp" specifying query location points, or an object of class "im" or "owin" specifying a grid of query points.
sigma	Bandwidth for kernel. A single number.
...	Extra arguments (passed to as.mask) controlling the pixel resolution of the result, when Xquery is a window or an image.
W	Window (object of class "owin") used to define the spatial domain when Xsource is not of class "ppp".
squared	Logical value indicating whether to take the square of each heat kernel value, before summing over the source points.
nmax	Number of terms to be used from the infinite-sum expression for the heat kernel. A single integer.

Details

This function computes the sum of heat kernels associated with each of the source points, evaluating them at each query location.

The window for evaluation of the heat kernel must be a rectangle.

The heat kernel in any region can be expressed as an infinite sum of terms associated with the eigenfunctions of the Laplacian. The heat kernel in a rectangle is the product of heat kernels for one-dimensional intervals on the horizontal and vertical axes. This function uses hotrod to compute the one-dimensional heat kernels, truncating the infinite sum to the first nmax terms, and then calculates the two-dimensional heat kernel from each source point to each query location. If squared=TRUE these values are squared. Finally the values are summed over all source points to obtain a single value for each query location.

Value

If Xquery is a point pattern, the result is a numeric vector with one entry for each query point.

If Xquery is an image or window, the result is a pixel image.

Author(s)

Adrian Baddeley and Greg McSwiggan.

References

Baddeley, A., Davies, T., Rakshit, S., Nair, G. and McSwiggan, G. (2021) Diffusion smoothing for spatial point patterns. *Statistical Science*, in press.

See Also

densityHeat.ppp

Examples

```
X <- runifpoint(10)

Y <- runifpoint(5)
hotbox(X, Y, 0.1)

plot(hotbox(X, Window(X), 0.1))
points(X, pch=16)
```

---

idw — *Inverse-distance weighted smoothing of observations at irregular points*

Description

Performs spatial smoothing of numeric values observed at a set of irregular locations using inverse-distance weighting.

Usage

```
idw(X, power=2, at=c("pixels", "points"), ..., se=FALSE)
```

Arguments

X	A marked point pattern (object of class "ppp").
power	Numeric. Power of distance used in the weighting.
at	Character string specifying whether to compute the intensity values at a grid of pixel locations (at="pixels") or only at the points of X (at="points"). String is partially matched.
...	Arguments passed to as.mask to control the pixel resolution of the result.
se	Logical value specifying whether to calculate a standard error.

Details

This function performs spatial smoothing of numeric values observed at a set of irregular locations.

Smoothing is performed by inverse distance weighting. If the observed values are $v_1, \ldots, v_n$ at locations $x_1, \ldots, x_n$ respectively, then the smoothed value at a location u is

$$g(u) = \frac{\sum_i w_i v_i}{\sum_i w_i}$$

where the weights are the inverse p-th powers of distance,

$$w_i = \frac{1}{d(u, x_i)^p}$$

where $d(u, x_i) = ||u - x_i||$ is the Euclidean distance from u to x_i.

The argument X must be a marked point pattern (object of class "ppp", see ppp.object). The points of the pattern are taken to be the observation locations x_i, and the marks of the pattern are taken to be the numeric values v_i observed at these locations.

The marks are allowed to be a data frame. Then the smoothing procedure is applied to each column of marks.

If at="pixels" (the default), the smoothed mark value is calculated at a grid of pixels, and the result is a pixel image. The arguments ... control the pixel resolution. See as.mask.

If at="points", the smoothed mark values are calculated at the data points only, using a leave-one-out rule (the mark value at a data point is excluded when calculating the smoothed value for that point).

An estimate of standard error is also calculated, if se=TRUE. The calculation assumes that the data point locations are fixed, that is, the standard error only takes into account the variability in the mark values, and not the variability due to randomness of the data point locations.

An alternative to inverse-distance weighting is kernel smoothing, which is performed by Smooth.ppp.

Value

If X *has a single column of marks:*

- If at="pixels" (the default), the result is a pixel image (object of class "im"). Pixel values are values of the interpolated function.
- If at="points", the result is a numeric vector of length equal to the number of points in X. Entries are values of the interpolated function at the points of X.

If X *has a data frame of marks:*

- If at="pixels" (the default), the result is a named list of pixel images (object of class "im"). There is one image for each column of marks. This list also belongs to the class "solist", for which there is a plot method.
- If at="points", the result is a data frame with one row for each point of X, and one column for each column of marks. Entries are values of the interpolated function at the points of X.

If se=TRUE, then the result is a list with two entries named estimate and SE, which each have the format described above.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>. Variance calculation by Andrew P Wheeler with modifications by Adrian Baddeley.

References

Shepard, D. (1968) A two-dimensional interpolation function for irregularly-spaced data. *Proceedings of the 1968 ACM National Conference*, 1968, pages 517–524. DOI: 10.1145/800186.810616

See Also

density.ppp, ppp.object, im.object.

See Smooth.ppp for kernel smoothing, SpatialMedian.ppp for median smoothing and nnmark for nearest-neighbour interpolation.

To perform other kinds of interpolation, see also the akima package.

Examples

```
# data frame of marks: trees marked by diameter and height
plot(idw(finpines))
idw(finpines, at="points")[1:5,]
plot(idw(finpines, se=TRUE)$SE)
idw(finpines, at="points", se=TRUE)$SE[1:5, ]
```

Iest *Estimate the I-function*

Description

Estimates the summary function $I(r)$ for a multitype point pattern.

Usage

```
Iest(X, ..., eps=NULL, r=NULL, breaks=NULL, correction=NULL)
```

Arguments

X	The observed point pattern, from which an estimate of $I(r)$ will be computed. An object of class "ppp", or data in any format acceptable to as.ppp().
...	Ignored.
eps	the resolution of the discrete approximation to Euclidean distance (see below). There is a sensible default.
r	Optional. Numeric vector of values for the argument r at which $I(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
breaks	This argument is for internal use only.
correction	Optional. Vector of character strings specifying the edge correction(s) to be used by Jest.

Details

The I function summarises the dependence between types in a multitype point process (Van Lieshout and Baddeley, 1999) It is based on the concept of the J function for an unmarked point process (Van Lieshout and Baddeley, 1996). See `Jest` for information about the J function.

The I function is defined as

$$I(r) = \sum_{i=1}^{m} p_i J_{ii}(r) - J_{\bullet\bullet}(r)$$

where $J_{\bullet\bullet}$ is the J function for the entire point process ignoring the marks, while J_{ii} is the J function for the process consisting of points of type i only, and p_i is the proportion of points which are of type i.

The I function is designed to measure dependence between points of different types, even if the points are not Poisson. Let X be a stationary multitype point process, and write X_i for the process of points of type i. If the processes X_i are independent of each other, then the I-function is identically equal to 0. Deviations $I(r) < 1$ or $I(r) > 1$ typically indicate negative and positive association, respectively, between types. See Van Lieshout and Baddeley (1999) for further information.

An estimate of I derived from a multitype spatial point pattern dataset can be used in exploratory data analysis and formal inference about the pattern. The estimate of $I(r)$ is compared against the constant function 0. Deviations $I(r) < 1$ or $I(r) > 1$ may suggest negative and positive association, respectively.

This algorithm estimates the I-function from the multitype point pattern `X`. It assumes that `X` can be treated as a realisation of a stationary (spatially homogeneous) random spatial marked point process in the plane, observed through a bounded window.

The argument `X` is interpreted as a point pattern object (of class `"ppp"`, see `ppp.object`) and can be supplied in any of the formats recognised by `as.ppp()`. It must be a multitype point pattern (it must have a `marks` vector which is a `factor`).

The function `Jest` is called to compute estimates of the J functions in the formula above. In fact three different estimates are computed using different edge corrections. See `Jest` for information.

Value

An object of class `"fv"`, see `fv.object`, which can be plotted directly using `plot.fv`.

Essentially a data frame containing

`r`	the vector of values of the argument r at which the function I has been estimated
`rs`	the "reduced sample" or "border correction" estimator of $I(r)$ computed from the border-corrected estimates of J functions
`km`	the spatial Kaplan-Meier estimator of $I(r)$ computed from the Kaplan-Meier estimates of J functions
`han`	the Hanisch-style estimator of $I(r)$ computed from the Hanisch-style estimates of J functions
`un`	the uncorrected estimate of $I(r)$ computed from the uncorrected estimates of J
`theo`	the theoretical value of $I(r)$ for a stationary Poisson process: identically equal to 0

Note

Sizeable amounts of memory may be needed during the calculation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

References

Van Lieshout, M.N.M. and Baddeley, A.J. (1996) A nonparametric measure of spatial interaction in point patterns. *Statistica Neerlandica* **50**, 344–361.

Van Lieshout, M.N.M. and Baddeley, A.J. (1999) Indices of dependence between types in multivariate point patterns. *Scandinavian Journal of Statistics* **26**, 511–532.

See Also

Jest

Examples

```
Ic <- Iest(amacrine)
plot(Ic, main="Amacrine Cells data")
# values are below I= 0, suggesting negative association
# between 'on' and 'off' cells.
```

---

increment.fv *Increments of a Function*

Description

Compute the change in the value of a function f when the function argument increases by delta.

Usage

```
increment.fv(f, delta)
```

Arguments

f	Object of class "fv" representing a function.
delta	Numeric. The increase in the value of the function argument.

Details

This command computes the new function

$$g(x) = f(x + h) - f(x - h)$$

where h = delta/2. The value of $g(x)$ is the change in the value of f over an interval of length delta centred at x.

Value

Another object of class "fv" compatible with X.

Author(s)

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

Rolf Turner <rolfturner@posteo.net>

and Ege Rubak <rubak@math.aau.dk>

See Also

fv.object, deriv.fv

Examples

```
plot(increment.fv(Kest(cells), 0.05))
```

---

integral.fv *Compute Integral of Function Object*

Description

Compute the integral of a function over a specified range.

Usage

```
## S3 method for class 'fv'
integral(f, domain = NULL, ...)
```

Arguments

f	A function value table (object of class "fv").
domain	Optional. Range of values of the argument x over which the density $f(x)$ should be integrated. A numeric vector of length 2 giving the minimum and maximum values of x. Infinite limits are permitted.
...	Ignored.

Details

This is a method for the generic function integral. It computes the numerical integral

$$I = \int f(x)dx$$

of the function object f. If domain is specified, the integral is restricted to the interval of x values given by the domain.

The result is a numeric value or numeric vector containing one entry for each column of function values in f.

Integrals are calculated numerically using the trapezoidal rule restricted to the domain given.

Value

A single numerical value, or a numeric vector.

Author(s)

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

See Also

fv.object, integral

stieltjes

Examples

```
g <- pcf(redwood, divisor="d")
integral(g, domain=c(0, 0.1))
```

ISE.envelope *Integrated Squared Error on an Envelope Object*

Description

Compute integrated squared error of each of the simulated function estimates in an envelope object.

Usage

```
ISE.envelope(object, theo, domain, dimension=2)
```

Arguments

object	Object of class "envelope" generated by the function envelope
theo	Function in the R language that evaluates the true (theoretically expected) value of the spatial summary function.
domain	Numeric vector of length 2 specifying the limits of the domain of integration for the integrated squared error.
dimension	Integer (either 1 or 2) specifying whether to calculate the one-dimensional or two-dimensional integral of squared error.

Details

The first argument should be an object of class "envelope" and should contain the simulated function estimates (i.e. it should have been computed using envelope with savefuns=TRUE).

The simulated function estimates are extracted from object, and their squared error from the true value theo is computed pointwise. The squared errors are integrated over the interval specified by domain, giving one value of integrated squared error for each simulated function estimate. These values are returned as a numerical vector.

Value

A numeric vector of length equal to the number of simulated functions.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Martin Hazelton <Martin.Hazelton@otago.ac.nz> and Tilman Davies <Tilman.Davies@otago.ac.nz>.

See Also

bias.envelope, MISE.envelope.

Examples

```
E <- envelope(cells, Kest, correction="translate", nsim=20, savefuns=TRUE)
theoK <- function(r) { pi * r^2 }
dom <- c(0, 0.1)
ISE.envelope(E, theoK, dom)
```

Jcross *Multitype J Function (i-to-j)*

Description

For a multitype point pattern, estimate the multitype J function summarising the interpoint dependence between points of type i and of type j.

Usage

```
Jcross(X, i, j, eps=NULL, r=NULL, breaks=NULL, ..., correction=NULL)
```

Arguments

X	The observed point pattern, from which an estimate of the multitype J function $J_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of marks(X).
j	The type (mark value) of the points in X to which distances are measured. A character string (or something that will be converted to a character string). Defaults to the second level of marks(X).
eps	A positive number. The resolution of the discrete approximation to Euclidean distance (see below). There is a sensible default.

r	Optional. Numeric vector. The values of the argument r at which the function $J_{ij}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
breaks	This argument is for internal use only.
...	Ignored.
correction	Optional. Character string specifying the edge correction(s) to be used. Options are "none", "rs", "km", "Hanisch" and "best". Alternatively correction="all" selects all options.

Details

This function Jcross and its companions Jdot and Jmulti are generalisations of the function Jest to multitype point patterns.

A multitype point pattern is a spatial pattern of points classified into a finite number of possible "colours" or "types". In the **spatstat** package, a multitype pattern is represented as a single point pattern object in which the points carry marks, and the mark value attached to each point determines the type of that point.

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to as.ppp. It must be a marked point pattern, and the mark vector X$marks must be a factor. The argument i will be interpreted as a level of the factor X$marks. (Warning: this means that an integer value i=3 will be interpreted as the number 3, **not** the 3rd smallest level).

The "type i to type j" multitype J function of a stationary multitype point process X was introduced by Van lieshout and Baddeley (1999). It is defined by

$$J_{ij}(r) = \frac{1 - G_{ij}(r)}{1 - F_j(r)}$$

where $G_{ij}(r)$ is the distribution function of the distance from a type i point to the nearest point of type j, and $F_j(r)$ is the distribution function of the distance from a fixed point in space to the nearest point of type j in the pattern.

An estimate of $J_{ij}(r)$ is a useful summary statistic in exploratory data analysis of a multitype point pattern. If the subprocess of type i points is independent of the subprocess of points of type j, then $J_{ij}(r) \equiv 1$. Hence deviations of the empirical estimate of J_{ij} from the value 1 may suggest dependence between types.

This algorithm estimates $J_{ij}(r)$ from the point pattern X. It assumes that X can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in X as Window(X)) may have arbitrary shape. Biases due to edge effects are treated in the same manner as in Jest, using the Kaplan-Meier and border corrections. The main work is done by Gmulti and Fest.

The argument r is the vector of values for the distance r at which $J_{ij}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

Value

An object of class "fv" (see fv.object).

Essentially a data frame containing six numeric columns

J	the recommended estimator of $J_{ij}(r)$, currently the Kaplan-Meier estimator.
r	the values of the argument r at which the function $J_{ij}(r)$ has been estimated
km	the Kaplan-Meier estimator of $J_{ij}(r)$
rs	the "reduced sample" or "border correction" estimator of $J_{ij}(r)$
han	the Hanisch-style estimator of $J_{ij}(r)$
un	the "uncorrected" estimator of $J_{ij}(r)$ formed by taking the ratio of uncorrected empirical estimators of $1 - G_{ij}(r)$ and $1 - F_j(r)$, see Gdot and Fest.
theo	the theoretical value of $J_{ij}(r)$ for a marked Poisson process, namely 1.

The result also has two attributes "G" and "F" which are respectively the outputs of Gcross and Fest for the point pattern.

Warnings

The arguments i and j are always interpreted as levels of the factor X$marks. They are converted to character strings if they are not already character strings. The value i=1 does **not** refer to the first level of the factor.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

Van Lieshout, M.N.M. and Baddeley, A.J. (1996) A nonparametric measure of spatial interaction in point patterns. *Statistica Neerlandica* **50**, 344–361.

Van Lieshout, M.N.M. and Baddeley, A.J. (1999) Indices of dependence between types in multivariate point patterns. *Scandinavian Journal of Statistics* **26**, 511–532.

See Also

Jdot, Jest, Jmulti

Examples

```
 # Lansing woods data: 6 types of trees
woods <- lansing

Jhm <- Jcross(woods, "hickory", "maple")
# diagnostic plot for independence between hickories and maples
plot(Jhm)

# synthetic example with two types "a" and "b"
```

```
pp <- runifpoint(30) %mark% factor(sample(c("a","b"), 30, replace=TRUE))
J <- Jcross(pp)
```

`Jcross.inhom` *Inhomogeneous Multitype J function (i-to-j)*

Description

For a multitype point pattern, estimate the inhomogeneous multitype J function summarising the interpoint dependence between points of type i and of type j.

Usage

```
Jcross.inhom(X, i, j,
             lambda = NULL, lambdaI = NULL, lambdaJ = NULL,
             lambdamin = NULL,
             ...,
             r = NULL, ReferenceMeasureMarkSetI = NULL, ratio = FALSE)
```

Arguments

`X`	The observed point pattern, from which an estimate of the multitype J function $J_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
`i`	The type (mark value) of the points in `X` from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of `marks(X)`.
`j`	The type (mark value) of the points in `X` to which distances are measured. A character string (or something that will be converted to a character string). Defaults to the second level of `marks(X)`.
`lambda`	Optional. Values of the estimated intensity of the point process. Either a pixel image (object of class `"im"`), a numeric vector containing the intensity values at each of the points in `X`, a fitted point process model (object of class `"ppm"` or `"kppm"` or `"dppm"`), or a `function(x,y)` which can be evaluated to give the intensity value at any location.
`lambdaI`	Optional. Values of the estimated intensity of the sub-process of points of type `i`. Either a pixel image (object of class `"im"`), a numeric vector containing the intensity values at each of the type `i` points in `X`, a fitted point process model (object of class `"ppm"` or `"kppm"` or `"dppm"`), or a `function(x,y)` which can be evaluated to give the intensity value at any location.
`lambdaJ`	Optional. Values of the the estimated intensity of the sub-process of points of type `j`. Either a pixel image (object of class `"im"`), a numeric vector containing the intensity values at each of the type `j` points in `X`, a fitted point process model (object of class `"ppm"` or `"kppm"` or `"dppm"`), or a `function(x,y)` which can be evaluated to give the intensity value at any location.

lambdamin	Optional. The minimum possible value of the intensity over the spatial domain. A positive numerical value.
...	Extra arguments passed to as.mask to control the pixel resolution for the computation.
r	vector of values for the argument r at which the inhomogeneous J function should be evaluated. Not normally given by the user; there is a sensible default.
ReferenceMeasureMarkSetI	Optional. The total measure of the mark set. A positive number.
ratio	Logical value indicating whether to save ratio information.

Details

This function is the counterpart of Jcross for inhomogeneous patterns. It is computed as a special case of Jmulti.inhom.

Value

Object of class "fv" containing the estimate of the inhomogeneous multitype J function.

Author(s)

Jonatan González and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

References

Cronie, O. and Van Lieshout, M.N.M. (2015) Summary statistics for inhomogeneous marked point processes. *Annals of the Institute of Statistical Mathematics* DOI: 10.1007/s10463-015-0515-z

See Also

Jdot.inhom, Jmulti.inhom, Jcross.

Examples

```
X <- rescale(amacrine)
if(interactive() && require(spatstat.model)) {
  ## how to do it normally
  mod <- ppm(X ~ marks * x)
  lam <- fitted(mod, dataonly=TRUE)
  lmin <- min(predict(mod)[["off"]]) * 0.9
  dd <- NULL
} else {
  ## for package testing
  lam <- intensity(X)[as.integer(marks(X))]
  lmin <- intensity(X)[2] * 0.9
  dd <- 32
}
JC <- Jcross.inhom(X, "on", "off", lambda=lam, lambdamin=lmin, dimyx=dd)
```

Jdot *Multitype J Function (i-to-any)*

Description

For a multitype point pattern, estimate the multitype J function summarising the interpoint dependence between the type i points and the points of any type.

Usage

```
Jdot(X, i, eps=NULL, r=NULL, breaks=NULL, ..., correction=NULL)
```

Arguments

X	The observed point pattern, from which an estimate of the multitype J function $J_{i\bullet}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of marks(X).
eps	A positive number. The resolution of the discrete approximation to Euclidean distance (see below). There is a sensible default.
r	numeric vector. The values of the argument r at which the function $J_{i\bullet}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
breaks	This argument is for internal use only.
...	Ignored.
correction	Optional. Character string specifying the edge correction(s) to be used. Options are "none", "rs", "km", "Hanisch" and "best". Alternatively correction="all" selects all options.

Details

This function Jdot and its companions Jcross and Jmulti are generalisations of the function Jest to multitype point patterns.

A multitype point pattern is a spatial pattern of points classified into a finite number of possible "colours" or "types". In the **spatstat** package, a multitype pattern is represented as a single point pattern object in which the points carry marks, and the mark value attached to each point determines the type of that point.

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to as.ppp. It must be a marked point pattern, and the mark vector X$marks must be a factor. The argument i will be interpreted as a level of the factor X$marks. (Warning: this means that an integer value i=3 will be interpreted as the number 3, **not** the 3rd smallest level.)

The "type i to any type" multitype J function of a stationary multitype point process X was introduced by Van lieshout and Baddeley (1999). It is defined by

$$J_{i\bullet}(r) = \frac{1 - G_{i\bullet}(r)}{1 - F_{\bullet}(r)}$$

where $G_{i\bullet}(r)$ is the distribution function of the distance from a type i point to the nearest other point of the pattern, and $F_{\bullet}(r)$ is the distribution function of the distance from a fixed point in space to the nearest point of the pattern.

An estimate of $J_{i\bullet}(r)$ is a useful summary statistic in exploratory data analysis of a multitype point pattern. If the pattern is a marked Poisson point process, then $J_{i\bullet}(r) \equiv 1$. If the subprocess of type i points is independent of the subprocess of points of all types not equal to i, then $J_{i\bullet}(r)$ equals $J_{ii}(r)$, the ordinary J function (see `Jest` and Van Lieshout and Baddeley (1996)) of the points of type i. Hence deviations from zero of the empirical estimate of $J_{i\bullet} - J_{ii}$ may suggest dependence between types.

This algorithm estimates $J_{i\bullet}(r)$ from the point pattern `X`. It assumes that `X` can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in `X` as `Window(X)`) may have arbitrary shape. Biases due to edge effects are treated in the same manner as in `Jest`, using the Kaplan-Meier and border corrections. The main work is done by `Gmulti` and `Fest`.

The argument `r` is the vector of values for the distance r at which $J_{i\bullet}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

Value

An object of class `"fv"` (see `fv.object`).

Essentially a data frame containing six numeric columns

`J`	the recommended estimator of $J_{i\bullet}(r)$, currently the Kaplan-Meier estimator.
`r`	the values of the argument r at which the function $J_{i\bullet}(r)$ has been estimated
`km`	the Kaplan-Meier estimator of $J_{i\bullet}(r)$
`rs`	the "reduced sample" or "border correction" estimator of $J_{i\bullet}(r)$
`han`	the Hanisch-style estimator of $J_{i\bullet}(r)$
`un`	the "uncorrected" estimator of $J_{i\bullet}(r)$ formed by taking the ratio of uncorrected empirical estimators of $1 - G_{i\bullet}(r)$ and $1 - F_{\bullet}(r)$, see `Gdot` and `Fest`.
`theo`	the theoretical value of $J_{i\bullet}(r)$ for a marked Poisson process, namely 1.

The result also has two attributes `"G"` and `"F"` which are respectively the outputs of `Gdot` and `Fest` for the point pattern.

Warnings

The argument `i` is interpreted as a level of the factor `X$marks`. It is converted to a character string if it is not already a character string. The value `i=1` does **not** refer to the first level of the factor.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>.

References

Van Lieshout, M.N.M. and Baddeley, A.J. (1996) A nonparametric measure of spatial interaction in point patterns. *Statistica Neerlandica* **50**, 344–361.

Van Lieshout, M.N.M. and Baddeley, A.J. (1999) Indices of dependence between types in multivariate point patterns. *Scandinavian Journal of Statistics* **26**, 511–532.

See Also

Jcross, Jest, Jmulti

Examples

```
  # Lansing woods data: 6 types of trees
woods <- lansing


Jh. <- Jdot(woods, "hickory")
plot(Jh.)
# diagnostic plot for independence between hickories and other trees
Jhh <- Jest(split(woods)$hickory)
plot(Jhh, add=TRUE, legendpos="bottom")

# synthetic example with two marks "a" and "b"

pp <- runifpoint(30) %mark% factor(sample(c("a","b"), 30, replace=TRUE))
J <- Jdot(pp, "a")
```

Jdot.inhom *Inhomogeneous Multitype J function (i-to-any)*

Description

For a multitype point pattern, estimate the inhomogeneous multitype J function summarising the interpoint dependence between points of type i and points of any type.

Usage

```
Jdot.inhom(X, i,
           lambdaI = NULL, lambdadot = NULL,
           lambdamin = NULL,
           ...,
           r = NULL, ReferenceMeasureMarkSetI = NULL, ratio = FALSE)
```

Arguments

X	The observed point pattern, from which an estimate of the inhomogeneous multitype J function $J_{i\bullet}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of marks(X).
lambdaI	Optional. Values of the estimated intensity of the sub-process of points of type i. Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the type i points in X, a fitted point process model (object of class "ppm" or "kppm" or "dppm"), or a function(x,y) which can be evaluated to give the intensity value at any location.
lambdadot	Optional. Values of the estimated intensity of the point process. Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the points in X, a fitted point process model (object of class "ppm" or "kppm" or "dppm"), or a function(x,y) which can be evaluated to give the intensity value at any location.
lambdamin	Optional. The minimum possible value of the intensity over the spatial domain. A positive numerical value.
...	Extra arguments passed to as.mask to control the pixel resolution for the computation.
r	vector of values for the argument r at which the inhomogeneous K function should be evaluated. Not normally given by the user; there is a sensible default.
ReferenceMeasureMarkSetI	Optional. The total measure of the mark set. A positive number.
ratio	Logical value indicating whether to save ratio information.

Details

This function is the counterpart of Jdot for inhomogeneous patterns. It is computed as a special case of Jmulti.inhom.

Value

Object of class "fv" containing the estimate of the inhomogeneous multitype J function.

Author(s)

Jonatan González and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

References

Cronie, O. and Van Lieshout, M.N.M. (2015) Summary statistics for inhomogeneous marked point processes. *Annals of the Institute of Statistical Mathematics* DOI: 10.1007/s10463-015-0515-z

See Also

Jdot.inhom, Jmulti.inhom, Jdot.

Examples

```
X <- rescale(amacrine)
if(interactive() && require(spatstat.model)) {
  ## how to do it normally
  mod <- ppm(X ~ marks * x)
  lam <- fitted(mod, dataonly=TRUE)
  lmin <- min(predict(mod)[["off"]]) * 0.9
  dd <- NULL
} else {
  ## for package testing
  lam <- intensity(X)[as.integer(marks(X))]
  lmin <- intensity(X)[2] * 0.9
  dd <- 32
}
lamI <- lam[marks(X) == "on"]
JD <- Jdot.inhom(X, "on", lambdaI=lamI, lambdadot=lam, lambdamin=lmin,
                 dimyx=dd)
```

Jest *Estimate the J-function*

Description

Estimates the summary function $J(r)$ for a point pattern in a window of arbitrary shape.

Usage

```
Jest(X, ..., eps=NULL, r=NULL, breaks=NULL, correction=NULL)
```

Arguments

X	The observed point pattern, from which an estimate of $J(r)$ will be computed. An object of class "ppp", or data in any format acceptable to as.ppp().
...	Ignored.
eps	the resolution of the discrete approximation to Euclidean distance (see below). There is a sensible default.
r	vector of values for the argument r at which $J(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
breaks	This argument is for internal use only.
correction	Optional. Character string specifying the choice of edge correction(s) in Fest and Gest. See Details.

Details

The J function (Van Lieshout and Baddeley, 1996) of a stationary point process is defined as

$$J(r) = \frac{1 - G(r)}{1 - F(r)}$$

where $G(r)$ is the nearest neighbour distance distribution function of the point process (see `Gest`) and $F(r)$ is its empty space function (see `Fest`).

For a completely random (uniform Poisson) point process, the J-function is identically equal to 1. Deviations $J(r) < 1$ or $J(r) > 1$ typically indicate spatial clustering or spatial regularity, respectively. The J-function is one of the few characteristics that can be computed explicitly for a wide range of point processes. See Van Lieshout and Baddeley (1996), Baddeley et al (2000), Thonnes and Van Lieshout (1999) for further information.

An estimate of J derived from a spatial point pattern dataset can be used in exploratory data analysis and formal inference about the pattern. The estimate of $J(r)$ is compared against the constant function 1. Deviations $J(r) < 1$ or $J(r) > 1$ may suggest spatial clustering or spatial regularity, respectively.

This algorithm estimates the J-function from the point pattern `X`. It assumes that `X` can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in `X` as `Window(X)`) may have arbitrary shape.

The argument `X` is interpreted as a point pattern object (of class `"ppp"`, see `ppp.object`) and can be supplied in any of the formats recognised by `as.ppp()`.

The functions `Fest` and `Gest` are called to compute estimates of $F(r)$ and $G(r)$ respectively. These estimates are then combined by simply taking the ratio $J(r) = (1 - G(r))/(1 - F(r))$.

In fact several different estimates are computed using different edge corrections (Baddeley, 1998).

The Kaplan-Meier estimate (returned as `km`) is the ratio `J = (1-G)/(1-F)` of the Kaplan-Meier estimates of $1 - F$ and $1 - G$ computed by `Fest` and `Gest` respectively. This is computed if `correction=NULL` or if `correction` includes `"km"`.

The Hanisch-style estimate (returned as `han`) is the ratio `J = (1-G)/(1-F)` where `F` is the Chiu-Stoyan estimate of F and `G` is the Hanisch estimate of G. This is computed if `correction=NULL` or if `correction` includes `"cs"` or `"han"`.

The reduced-sample or border corrected estimate (returned as `rs`) is the same ratio `J = (1-G)/(1-F)` of the border corrected estimates. This is computed if `correction=NULL` or if `correction` includes `"rs"` or `"border"`.

These edge-corrected estimators are slightly biased for J, since they are ratios of approximately unbiased estimators. The logarithm of the Kaplan-Meier estimate is exactly unbiased for $\log J$.

The uncorrected estimate (returned as `un` and computed only if `correction` includes `"none"`) is the ratio `J = (1-G)/(1-F)` of the uncorrected ("raw") estimates of the survival functions of F and G, which are the empirical distribution functions of the empty space distances `Fest(X,...)$raw` and of the nearest neighbour distances `Gest(X,...)$raw`. The uncorrected estimates of F and G are severely biased. However the uncorrected estimate of J is approximately unbiased (if the process is close to Poisson); it is insensitive to edge effects, and should be used when edge effects are severe (see Baddeley et al, 2000).

The algorithm for Fest uses two discrete approximations which are controlled by the parameter eps and by the spacing of values of r respectively. See Fest for details. First-time users are strongly advised not to specify these arguments.

Note that the value returned by Jest includes the output of Fest and Gest as attributes (see the last example below). If the user is intending to compute the F,G and J functions for the point pattern, it is only necessary to call Jest.

Value

An object of class "fv", see fv.object, which can be plotted directly using plot.fv.

Essentially a data frame containing

r	the vector of values of the argument r at which the function J has been estimated
rs	the "reduced sample" or "border correction" estimator of $J(r)$ computed from the border-corrected estimates of F and G
km	the spatial Kaplan-Meier estimator of $J(r)$ computed from the Kaplan-Meier estimates of F and G
han	the Hanisch-style estimator of $J(r)$ computed from the Hanisch estimate of G and the Chiu-Stoyan estimate of F
un	the uncorrected estimate of $J(r)$ computed from the uncorrected estimates of F and G
theo	the theoretical value of $J(r)$ for a stationary Poisson process: identically equal to 1

The data frame also has **attributes**

F	the output of Fest for this point pattern, containing three estimates of the empty space function $F(r)$ and an estimate of its hazard function
G	the output of Gest for this point pattern, containing three estimates of the nearest neighbour distance distribution function $G(r)$ and an estimate of its hazard function

Note

Sizeable amounts of memory may be needed during the calculation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

References

Baddeley, A.J. Spatial sampling and censoring. In O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*. Chapman and Hall, 1998. Chapter 2, pages 37–78.

Baddeley, A.J. and Gill, R.D. The empty space hazard of a spatial pattern. Research Report 1994/3, Department of Mathematics, University of Western Australia, May 1994.

Baddeley, A.J. and Gill, R.D. Kaplan-Meier estimators of interpoint distance distributions for spatial point processes. *Annals of Statistics* **25** (1997) 263–292.

Baddeley, A., Kerscher, M., Schladitz, K. and Scott, B.T. Estimating the *J* function without edge correction. *Statistica Neerlandica* **54** (2000) 315–328.

Borgefors, G. Distance transformations in digital images. *Computer Vision, Graphics and Image Processing* **34** (1986) 344–371.

Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.

Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.

Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.

Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.

Thonnes, E. and Van Lieshout, M.N.M, A comparative study on the power of Van Lieshout and Baddeley's J-function. *Biometrical Journal* **41** (1999) 721–734.

Van Lieshout, M.N.M. and Baddeley, A.J. A nonparametric measure of spatial interaction in point patterns. *Statistica Neerlandica* **50** (1996) 344–361.

See Also

Jinhom, Fest, Gest, Kest, km.rs, reduced.sample, kaplan.meier

Examples

```
J <- Jest(cells, 0.01)
plot(J, main="cells data")
# values are far above J = 1, indicating regular pattern

data(redwood)
J <- Jest(redwood, 0.01, legendpos="center")
plot(J, main="redwood data")
# values are below J = 1, indicating clustered pattern
```

Jinhom *Inhomogeneous J-function*

Description

Estimates the inhomogeneous *J* function of a non-stationary point pattern.

Usage

```
Jinhom(X, lambda = NULL, lmin = NULL, ...,
       sigma = NULL, varcov = NULL,
       r = NULL, breaks = NULL, ratio=FALSE,
       update = TRUE, warn.bias=TRUE, savelambda=FALSE)
```

Arguments

X	The observed data point pattern, from which an estimate of the inhomogeneous J function will be computed. An object of class "ppp" or in a format recognised by as.ppp()
lambda	Optional. Values of the estimated intensity function. Either a vector giving the intensity values at the points of the pattern X, a pixel image (object of class "im") giving the intensity values at all locations, a fitted point process model (object of class "ppm" or "kppm") or a function(x,y) which can be evaluated to give the intensity value at any location.
lmin	Optional. The minimum possible value of the intensity over the spatial domain. A positive numerical value.
sigma, varcov	Optional arguments passed to density.ppp to control the smoothing bandwidth, when lambda is estimated by kernel smoothing.
...	Extra arguments passed to as.mask to control the pixel resolution, or passed to density.ppp to control the smoothing bandwidth.
r	vector of values for the argument r at which the inhomogeneous K function should be evaluated. Not normally given by the user; there is a sensible default.
breaks	This argument is for internal use only.
ratio	Logical. If TRUE, the numerator and denominator of the estimate will also be saved, for use in analysing replicated point patterns.
update	Logical. If lambda is a fitted model (class "ppm" or "kppm") and update=TRUE (the default), the model will first be refitted to the data X (using update.ppm or update.kppm) before the fitted intensity is computed. If update=FALSE, the fitted intensity of the model will be computed without fitting it to X.
warn.bias	Logical value specifying whether to issue a warning when the inhomogeneity correction factor takes extreme values, which can often lead to biased results. This usually occurs when insufficient smoothing is used to estimate the intensity.
savelambda	Logical value specifying whether to save the values of lmin and lambda as attributes of the result.

Details

This command computes estimates of the inhomogeneous J-function (Van Lieshout, 2010) of a point pattern. It is the counterpart, for inhomogeneous spatial point patterns, of the J function for homogeneous point patterns computed by Jest.

The argument X should be a point pattern (object of class "ppp").

The inhomogeneous J function is computed as $Jinhom(r) = (1-Ginhom(r))/(1-Finhom(r))$ where $Ginhom, Finhom$ are the inhomogeneous G and F functions computed using the border correction (equations (7) and (6) respectively in Van Lieshout, 2010).

The argument lambda should supply the (estimated) values of the intensity function λ of the point process. It may be either

a numeric vector containing the values of the intensity function at the points of the pattern X.

a pixel image (object of class "im") assumed to contain the values of the intensity function at all locations in the window.

a fitted point process model (object of class "ppm" or "kppm") whose fitted *trend* can be used as the fitted intensity. (If update=TRUE the model will first be refitted to the data X before the trend is computed.)

a function which can be evaluated to give values of the intensity at any locations.

omitted: if lambda is omitted, then it will be estimated using a 'leave-one-out' kernel smoother.

If lambda is a numeric vector, then its length should be equal to the number of points in the pattern X. The value lambda[i] is assumed to be the the (estimated) value of the intensity $\lambda(x_i)$ for the point x_i of the pattern X. Each value must be a positive number; NA's are not allowed.

If lambda is a pixel image, the domain of the image should cover the entire window of the point pattern. If it does not (which may occur near the boundary because of discretisation error), then the missing pixel values will be obtained by applying a Gaussian blur to lambda using blur, then looking up the values of this blurred image for the missing locations. (A warning will be issued in this case.)

If lambda is a function, then it will be evaluated in the form lambda(x,y) where x and y are vectors of coordinates of the points of X. It should return a numeric vector with length equal to the number of points in X.

If lambda is omitted, then it will be estimated using a 'leave-one-out' kernel smoother. The estimate lambda[i] for the point X[i] is computed by removing X[i] from the point pattern, applying kernel smoothing to the remaining points using density.ppp, and evaluating the smoothed intensity at the point X[i]. The smoothing kernel bandwidth is controlled by the arguments sigma and varcov, which are passed to density.ppp along with any extra arguments.

Value

An object of class "fv", see fv.object, which can be plotted directly using plot.fv.

Author(s)

Original code by Marie-Colette van Lieshout. C implementation and R adaptation by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Ege Rubak <rubak@math.aau.dk>.

References

van Lieshout, M.N.M. and Baddeley, A.J. (1996) A nonparametric measure of spatial interaction in point patterns. *Statistica Neerlandica* **50**, 344–361.

van Lieshout, M.N.M. (2010) A J-function for inhomogeneous point processes. *Statistica Neerlandica* **65**, 183–201.

See Also

Ginhom, Finhom, Jest

Examples

```
online <- interactive()
if(online) {
  plot(Jinhom(swedishpines, sigma=10))
```

```
  plot(Jinhom(swedishpines, sigma=bw.diggle, adjust=2))
} else {
  ## use a coarse grid for faster computation and package testing
  plot(Jinhom(swedishpines, sigma=10, dimyx=32))
}
```

Jmulti *Marked J Function*

Description

For a marked point pattern, estimate the multitype J function summarising dependence between the points in subset I and those in subset J.

Usage

```
Jmulti(X, I, J, eps=NULL, r=NULL, breaks=NULL, ..., disjoint=NULL,
       correction=NULL)
```

Arguments

X	The observed point pattern, from which an estimate of the multitype distance distribution function $J_{IJ}(r)$ will be computed. It must be a marked point pattern. See under Details.
I	Subset of points of X from which distances are measured. See Details.
J	Subset of points in X to which distances are measured. See Details.
eps	A positive number. The pixel resolution of the discrete approximation to Euclidean distance (see Jest). There is a sensible default.
r	numeric vector. The values of the argument r at which the distribution function $J_{IJ}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
breaks	This argument is for internal use only.
...	Ignored.
disjoint	Optional flag indicating whether the subsets I and J are disjoint. If missing, this value will be computed by inspecting the vectors I and J.
correction	Optional. Character string specifying the edge correction(s) to be used. Options are "none", "rs", "km", "Hanisch" and "best". Alternatively correction="all" selects all options.

Details

The function Jmulti generalises Jest (for unmarked point patterns) and Jdot and Jcross (for multitype point patterns) to arbitrary marked point patterns.

Suppose X_I, X_J are subsets, possibly overlapping, of a marked point process. Define

$$J_{IJ}(r) = \frac{1 - G_{IJ}(r)}{1 - F_J(r)}$$

where $F_J(r)$ is the cumulative distribution function of the distance from a fixed location to the nearest point of X_J, and $G_{IJ}(r)$ is the distribution function of the distance from a typical point of X_I to the nearest distinct point of X_J.

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to as.ppp.

The arguments I and J specify two subsets of the point pattern. They may be any type of subset indices, for example, logical vectors of length equal to npoints(X), or integer vectors with entries in the range 1 to npoints(X), or negative integer vectors.

Alternatively, I and J may be **functions** that will be applied to the point pattern X to obtain index vectors. If I is a function, then evaluating I(X) should yield a valid subset index. This option is useful when generating simulation envelopes using envelope.

It is assumed that X can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in X as Window(X)) may have arbitrary shape. Biases due to edge effects are treated in the same manner as in Jest.

The argument r is the vector of values for the distance r at which $J_{IJ}(r)$ should be evaluated. It is also used to determine the breakpoints (in the sense of hist) for the computation of histograms of distances. The reduced-sample and Kaplan-Meier estimators are computed from histogram counts. In the case of the Kaplan-Meier estimator this introduces a discretisation error which is controlled by the fineness of the breakpoints.

First-time users would be strongly advised not to specify r. However, if it is specified, r must satisfy r[1] = 0, and max(r) must be larger than the radius of the largest disc contained in the window. Furthermore, the successive entries of r must be finely spaced.

Value

An object of class "fv" (see fv.object).

Essentially a data frame containing six numeric columns

r	the values of the argument r at which the function $J_{IJ}(r)$ has been estimated
rs	the "reduced sample" or "border correction" estimator of $J_{IJ}(r)$
km	the spatial Kaplan-Meier estimator of $J_{IJ}(r)$
han	the Hanisch-style estimator of $J_{IJ}(r)$
un	the uncorrected estimate of $J_{IJ}(r)$, formed by taking the ratio of uncorrected empirical estimators of $1 - G_{IJ}(r)$ and $1 - F_J(r)$, see Gdot and Fest.
theo	the theoretical value of $J_{IJ}(r)$ for a marked Poisson process with the same estimated intensity, namely 1.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

Van Lieshout, M.N.M. and Baddeley, A.J. (1999) Indices of dependence between types in multivariate point patterns. *Scandinavian Journal of Statistics* **26**, 511–532.

See Also

Jcross, Jdot, Jest

Examples

```
trees <- longleaf
 # Longleaf Pine data: marks represent diameter

Jm <- Jmulti(trees, marks(trees) <= 15, marks(trees) >= 25)
plot(Jm)
```

Jmulti.inhom *Inhomogeneous Marked J-Function*

Description

For a marked point pattern, estimate the inhomogeneous version of the multitype J function.

Usage

```
Jmulti.inhom(X, I, J,
             lambda = NULL, lambdaI = NULL, lambdaJ = NULL,
             lambdamin = NULL,
             ...,
             r = NULL,
             ReferenceMeasureMarkSetI = NULL,
             ratio = FALSE)
```

Arguments

X	The observed point pattern, from which an estimate of the inhomogeneous multitype J function $J_{IJ}(r)$ will be computed. It must be a marked point pattern. See under Details.
I	Subset index specifying the points of X from which distances are measured, for the inhomogeneous G function.
J	Subset index specifying the points in X to which distances are measured, for the inhomogeneous G and F functions.

lambda	Optional. Values of the estimated intensity function. Either a vector giving the intensity values at the points of the pattern X, a pixel image (object of class "im") giving the intensity values at all locations, a fitted point process model (object of class "ppm") or a function(x,y) which can be evaluated to give the intensity value at any location.
lambdaI	Optional. Values of the estimated intensity of the sub-process X[I]. Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the points in X[I], a fitted point process model (object of class "ppm" or "kppm" or "dppm"), or a function(x,y) which can be evaluated to give the intensity value at any location,
lambdaJ	Optional. Values of the estimated intensity of the sub-process X[J]. Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the points in X[J], a fitted point process model (object of class "ppm" or "kppm" or "dppm"), or a function(x,y) which can be evaluated to give the intensity value at any location.
lambdamin	Optional. The minimum possible value of the intensity over the spatial domain. A positive numerical value.
...	Extra arguments passed to as.mask to control the pixel resolution for the computation.
r	vector of values for the argument r at which the inhomogeneous K function should be evaluated. Not normally given by the user; there is a sensible default.
ReferenceMeasureMarkSetI	Optional. The total measure of the mark set. A positive number.
ratio	Logical value indicating whether to save ratio information.

Details

This function is the counterpart of Jmulti for inhomogeneous patterns. It is computed by evaluating the inhomogeneous G function GmultiInhom and the inhomogeneous F function FmultiInhom and computing the ratio $J = (1 - G)/(1 - F)$.

Value

Object of class "fv" containing the estimate of the inhomogeneous multitype J function.

Author(s)

Jonatan González and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

References

Cronie, O. and Van Lieshout, M.N.M. (2015) Summary statistics for inhomogeneous marked point processes. *Annals of the Institute of Statistical Mathematics* DOI: 10.1007/s10463-015-0515-z

See Also

Jcross.inhom, Jdot.inhom for special cases.

GmultiInhom, FmultiInhom, Jmulti.

Examples

```
X <- rescale(amacrine)
I <- (marks(X) == "on")
J <- (marks(X) == "off")
if(interactive() && require(spatstat.model)) {
  ## how to do it normally
  mod <- ppm(X ~ marks * x)
  lam <- fitted(mod, dataonly=TRUE)
  lmin <- min(predict(mod)[["off"]]) * 0.9
  dd <- NULL
} else {
  ## for package testing
  lam <- intensity(X)[as.integer(marks(X))]
  lmin <- intensity(X)[2] * 0.9
  dd <- 32
}
JM <- Jmulti.inhom(X, I, J, lambda=lam, lambdamin=lmin, dimyx=dd)
```

K3est *K-function of a Three-Dimensional Point Pattern*

Description

Estimates the K-function from a three-dimensional point pattern.

Usage

```
K3est(X, ...,
      rmax = NULL, nrval = 128,
      correction = c("translation", "isotropic"),
      ratio=FALSE)
```

Arguments

X	Three-dimensional point pattern (object of class "pp3").
...	Ignored.
rmax	Optional. Maximum value of argument r for which $K_3(r)$ will be estimated.
nrval	Optional. Number of values of r for which $K_3(r)$ will be estimated. A large value of nrval is required to avoid discretisation effects.
correction	Optional. Character vector specifying the edge correction(s) to be applied. See Details.
ratio	Logical. If TRUE, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.

Details

For a stationary point process Φ in three-dimensional space, the three-dimensional K function is

$$K_3(r) = \frac{1}{\lambda} E(N(\Phi, x, r) \mid x \in \Phi)$$

where λ is the intensity of the process (the expected number of points per unit volume) and $N(\Phi, x, r)$ is the number of points of Φ, other than x itself, which fall within a distance r of x. This is the three-dimensional generalisation of Ripley's K function for two-dimensional point processes (Ripley, 1977).

The three-dimensional point pattern `X` is assumed to be a partial realisation of a stationary point process Φ. The distance between each pair of distinct points is computed. The empirical cumulative distribution function of these values, with appropriate edge corrections, is renormalised to give the estimate of $K_3(r)$.

The available edge corrections are:

`"translation"`: the Ohser translation correction estimator (Ohser, 1983; Baddeley et al, 1993)

`"isotropic"`: the three-dimensional counterpart of Ripley's isotropic edge correction (Ripley, 1977; Baddeley et al, 1993).

Alternatively `correction="all"` selects all options.

Value

A function value table (object of class `"fv"`) that can be plotted, printed or coerced to a data frame containing the function values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rana Moyeed.

References

Baddeley, A.J, Moyeed, R.A., Howard, C.V. and Boyde, A. (1993) Analysis of a three-dimensional point pattern with replication. *Applied Statistics* **42**, 641–668.

Ohser, J. (1983) On estimators for the reduced second moment measure of point processes. *Mathematische Operationsforschung und Statistik, series Statistics*, **14**, 63 – 71.

Ripley, B.D. (1977) Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, Series B*, **39**, 172 – 212.

See Also

`pp3` to create a three-dimensional point pattern (object of class `"pp3"`).

`pcf3est`, `F3est`, `G3est` for other summary functions of a three-dimensional point pattern.

`Kest` to estimate the K-function of point patterns in two dimensions or other spaces.

Examples

```
X <- rpoispp3(42)
Z <- K3est(X)
if(interactive()) plot(Z)
```

Kcross *Multitype K Function (Cross-type)*

Description

For a multitype point pattern, estimate the multitype K function which counts the expected number of points of type j within a given distance of a point of type i.

Usage

```
Kcross(X, i, j, r=NULL, breaks=NULL, correction,
       ..., ratio=FALSE, from, to )
```

Arguments

X	The observed point pattern, from which an estimate of the cross type K function $K_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of marks(X).
j	The type (mark value) of the points in X to which distances are measured. A character string (or something that will be converted to a character string). Defaults to the second level of marks(X).
r	numeric vector. The values of the argument r at which the distribution function $K_{ij}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
breaks	This argument is for internal use only.
correction	A character vector containing any selection of the options "border", "bord.modif", "isotropic", "Ripley", "translate", "translation", "periodic", "none" or "best". It specifies the edge correction(s) to be applied. Alternatively correction="all" selects all options.
...	Ignored.
ratio	Logical. If TRUE, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.
from, to	An alternative way to specify i and j respectively.

Details

This function Kcross and its companions Kdot and Kmulti are generalisations of the function Kest to multitype point patterns.

A multitype point pattern is a spatial pattern of points classified into a finite number of possible "colours" or "types". In the **spatstat** package, a multitype pattern is represented as a single point pattern object in which the points carry marks, and the mark value attached to each point determines the type of that point.

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to as.ppp. It must be a marked point pattern, and the mark vector X$marks must be a factor.

The arguments i and j will be interpreted as levels of the factor X$marks. If i and j are missing, they default to the first and second level of the marks factor, respectively.

The "cross-type" (type i to type j) K function of a stationary multitype point process X is defined so that $\lambda_j K_{ij}(r)$ equals the expected number of additional random points of type j within a distance r of a typical point of type i in the process X. Here λ_j is the intensity of the type j points, i.e. the expected number of points of type j per unit area. The function K_{ij} is determined by the second order moment properties of X.

An estimate of $K_{ij}(r)$ is a useful summary statistic in exploratory data analysis of a multitype point pattern. If the process of type i points were independent of the process of type j points, then $K_{ij}(r)$ would equal πr^2. Deviations between the empirical K_{ij} curve and the theoretical curve πr^2 may suggest dependence between the points of types i and j.

This algorithm estimates the distribution function $K_{ij}(r)$ from the point pattern X. It assumes that X can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in X as Window(X)) may have arbitrary shape. Biases due to edge effects are treated in the same manner as in Kest, using the border correction.

The argument r is the vector of values for the distance r at which $K_{ij}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

The pair correlation function can also be applied to the result of Kcross; see pcf.

Value

An object of class "fv" (see fv.object).

Essentially a data frame containing numeric columns

r	the values of the argument r at which the function $K_{ij}(r)$ has been estimated
theo	the theoretical value of $K_{ij}(r)$ for a marked Poisson process, namely πr^2

together with a column or columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $K_{ij}(r)$ obtained by the edge corrections named.

If ratio=TRUE then the return value also has two attributes called "numerator" and "denominator" which are "fv" objects containing the numerators and denominators of each estimate of $K(r)$.

Warnings

The arguments `i` and `j` are always interpreted as levels of the factor `X$marks`. They are converted to character strings if they are not already character strings. The value `i=1` does **not** refer to the first level of the factor.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>.

References

Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.

Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.

Harkness, R.D and Isham, V. (1983) A bivariate spatial point pattern of ants' nests. *Applied Statistics* **32**, 293–303

Lotwick, H. W. and Silverman, B. W. (1982). Methods for analysing spatial processes of several types of points. *J. Royal Statist. Soc. Ser. B* **44**, 406–413.

Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.

Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.

See Also

`Kdot`, `Kest`, `Kmulti`, `pcf`

Examples

```
# amacrine cells data
K01 <- Kcross(amacrine, "off", "on")
plot(K01)



# synthetic example: point pattern with marks 0 and 1

 pp <- runifpoispp(50)
 pp <- pp %mark% factor(sample(0:1, npoints(pp), replace=TRUE))
 K <- Kcross(pp, "0", "1")
 K <- Kcross(pp, 0, 1) # equivalent
```

Kcross.inhom *Inhomogeneous Cross K Function*

Description

For a multitype point pattern, estimate the inhomogeneous version of the cross K function, which counts the expected number of points of type j within a given distance of a point of type i, adjusted for spatially varying intensity.

Usage

```
Kcross.inhom(X, i, j, lambdaI=NULL, lambdaJ=NULL, ...,  r=NULL, breaks=NULL,
         correction = c("border", "isotropic", "Ripley", "translate"),
         sigma=NULL, varcov=NULL,
         lambdaIJ=NULL,
         lambdaX=NULL, update=TRUE, leaveoneout=TRUE)
```

Arguments

X	The observed point pattern, from which an estimate of the inhomogeneous cross type K function $K_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of marks(X).
j	The type (mark value) of the points in X to which distances are measured. A character string (or something that will be converted to a character string). Defaults to the second level of marks(X).
lambdaI	Optional. Values of the estimated intensity of the sub-process of points of type i. Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the type i points in X, a fitted point process model (object of class "ppm" or "kppm" or "dppm"), or a function(x,y) which can be evaluated to give the intensity value at any location.
lambdaJ	Optional. Values of the the estimated intensity of the sub-process of points of type j. Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the type j points in X, a fitted point process model (object of class "ppm" or "kppm" or "dppm"), or a function(x,y) which can be evaluated to give the intensity value at any location.
r	Optional. Numeric vector giving the values of the argument r at which the cross K function $K_{ij}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
breaks	This argument is for advanced use only.

correction	A character vector containing any selection of the options "border", "bord.modif", "isotropic", "Ripley" ,"translate", "translation", "none" or "best". It specifies the edge correction(s) to be applied. Alternatively correction="all" selects all options.
...	Ignored.
sigma	Standard deviation of isotropic Gaussian smoothing kernel, used in computing leave-one-out kernel estimates of lambdaI, lambdaJ if they are omitted.
varcov	Variance-covariance matrix of anisotropic Gaussian kernel, used in computing leave-one-out kernel estimates of lambdaI, lambdaJ if they are omitted. Incompatible with sigma.
lambdaIJ	Optional. A matrix containing estimates of the product of the intensities lambdaI and lambdaJ for each pair of points of types i and j respectively.
lambdaX	Optional. Values of the intensity for all points of X. Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the points in X, a fitted point process model (object of class "ppm" or "kppm" or "dppm"), or a function(x,y) which can be evaluated to give the intensity value at any location. If present, this argument overrides both lambdaI and lambdaJ.
update	Logical value indicating what to do when lambdaI, lambdaJ or lambdaX is a fitted point process model (class "ppm", "kppm" or "dppm"). If update=TRUE (the default), the model will first be refitted to the data X (using update.ppm or update.kppm) before the fitted intensity is computed. If update=FALSE, the fitted intensity of the model will be computed without re-fitting it to X.
leaveoneout	Logical value (passed to density.ppp or fitted.ppm) specifying whether to use a leave-one-out rule when calculating the intensity.

Details

This is a generalisation of the function Kcross to include an adjustment for spatially inhomogeneous intensity, in a manner similar to the function Kinhom.

The inhomogeneous cross-type K function is described by Møller and Waagepetersen (2003, pages 48-49 and 51-53).

Briefly, given a multitype point process, suppose the sub-process of points of type j has intensity function $\lambda_j(u)$ at spatial locations u. Suppose we place a mass of $1/\lambda_j(\zeta)$ at each point ζ of type j. Then the expected total mass per unit area is 1. The inhomogeneous "cross-type" K function $K_{ij}^{\mathrm{inhom}}(r)$ equals the expected total mass within a radius r of a point of the process of type i.

If the process of type i points were independent of the process of type j points, then $K_{ij}^{\mathrm{inhom}}(r)$ would equal πr^2. Deviations between the empirical K_{ij} curve and the theoretical curve πr^2 suggest dependence between the points of types i and j.

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to as.ppp. It must be a marked point pattern, and the mark vector X$marks must be a factor.

The arguments i and j will be interpreted as levels of the factor X$marks. (Warning: this means that an integer value i=3 will be interpreted as the number 3, **not** the 3rd smallest level). If i and j are missing, they default to the first and second level of the marks factor, respectively.

The argument lambdaI supplies the values of the intensity of the sub-process of points of type i. It may be either

a pixel image (object of class "im") which gives the values of the type i intensity at all locations in the window containing X;

a numeric vector containing the values of the type i intensity evaluated only at the data points of type i. The length of this vector must equal the number of type i points in X.

a function which can be evaluated to give values of the intensity at any locations.

a fitted point process model (object of class "ppm", "kppm" or "dppm") whose fitted *trend* can be used as the fitted intensity. (If update=TRUE the model will first be refitted to the data X before the trend is computed.)

omitted: if lambdaI is omitted then it will be estimated using a leave-one-out kernel smoother.

If lambdaI is omitted, then it will be estimated using a 'leave-one-out' kernel smoother, as described in Baddeley, Møller and Waagepetersen (2000). The estimate of lambdaI for a given point is computed by removing the point from the point pattern, applying kernel smoothing to the remaining points using density.ppp, and evaluating the smoothed intensity at the point in question. The smoothing kernel bandwidth is controlled by the arguments sigma and varcov, which are passed to density.ppp along with any extra arguments.

Similarly lambdaJ should contain estimated values of the intensity of the sub-process of points of type j. It may be either a pixel image, a function, a numeric vector, or omitted.

Alternatively if the argument lambdaX is given, then it specifies the intensity values for all points of X, and the arguments lambdaI, lambdaJ will be ignored.

The optional argument lambdaIJ is for advanced use only. It is a matrix containing estimated values of the products of these two intensities for each pair of data points of types i and j respectively.

The argument r is the vector of values for the distance r at which $K_{ij}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

The argument correction chooses the edge correction as explained e.g. in Kest.

The pair correlation function can also be applied to the result of Kcross.inhom; see pcf.

Value

An object of class "fv" (see fv.object).

Essentially a data frame containing numeric columns

r	the values of the argument r at which the function $K_{ij}(r)$ has been estimated
theo	the theoretical value of $K_{ij}(r)$ for a marked Poisson process, namely πr^2

together with a column or columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $K_{ij}(r)$ obtained by the edge corrections named.

Warnings

The arguments i and j are always interpreted as levels of the factor X$marks. They are converted to character strings if they are not already character strings. The value i=1 does **not** refer to the first level of the factor.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A., Møller, J. and Waagepetersen, R. (2000) Non- and semiparametric estimation of interaction in inhomogeneous point patterns. *Statistica Neerlandica* **54**, 329–350.

Møller, J. and Waagepetersen, R. Statistical Inference and Simulation for Spatial Point Processes Chapman and Hall/CRC Boca Raton, 2003.

See Also

Kcross, Kinhom, Kdot.inhom, Kmulti.inhom, pcf

Examples

```
# Lansing Woods data
woods <- lansing

ma <- split(woods)$maple
wh <- split(woods)$whiteoak

# method (1): estimate intensities by nonparametric smoothing
lambdaM <- density.ppp(ma, sigma=0.15, at="points")
lambdaW <- density.ppp(wh, sigma=0.15, at="points")
K <- Kcross.inhom(woods, "whiteoak", "maple", lambdaW, lambdaM)

# method (2): leave-one-out
K <- Kcross.inhom(woods, "whiteoak", "maple", sigma=0.15)

# method (3): fit parametric intensity model
if(require("spatstat.model")) {
fit <- ppm(woods ~marks * polynom(x,y,2))
# alternative (a): use fitted model as 'lambda' argument
online <- interactive()
K <- Kcross.inhom(woods, "whiteoak", "maple",
                  lambdaI=fit, lambdaJ=fit,
                  update=online, leaveoneout=online)
K <- Kcross.inhom(woods, "whiteoak", "maple",
                  lambdaX=fit,
                  update=online, leaveoneout=online)
# alternative (b): evaluate fitted intensities at data points
# (these are the intensities of the sub-processes of each type)
inten <- fitted(fit, dataonly=TRUE, leaveoneout=FALSE)
# split according to types of points
lambda <- split(inten, marks(woods))
K <- Kcross.inhom(woods, "whiteoak", "maple",
          lambda$whiteoak, lambda$maple)
}
```

```
# synthetic example: type A points have intensity 50,
#                    type B points have intensity 100 * x
lamB <- as.im(function(x,y){50 + 100 * x}, owin())
X <- superimpose(A=runifpoispp(50), B=rpoispp(lamB))
K <- Kcross.inhom(X, "A", "B",
    lambdaI=as.im(50, Window(X)), lambdaJ=lamB)
```

Kdot *Multitype K Function (i-to-any)*

Description

For a multitype point pattern, estimate the multitype K function which counts the expected number of other points of the process within a given distance of a point of type i.

Usage

```
Kdot(X, i, r=NULL, breaks=NULL, correction, ..., ratio=FALSE, from)
```

Arguments

X	The observed point pattern, from which an estimate of the multitype K function $K_{i\bullet}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of marks(X).
r	numeric vector. The values of the argument r at which the distribution function $K_{i\bullet}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
breaks	This argument is for internal use only.
correction	A character vector containing any selection of the options "border", "bord.modif", "isotropic", "Ripley", "translate", "translation", "periodic", "none" or "best". It specifies the edge correction(s) to be applied. Alternatively correction="all" selects all options.
...	Ignored.
ratio	Logical. If TRUE, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.
from	An alternative way to specify i.

Details

This function Kdot and its companions Kcross and Kmulti are generalisations of the function Kest to multitype point patterns.

A multitype point pattern is a spatial pattern of points classified into a finite number of possible "colours" or "types". In the **spatstat** package, a multitype pattern is represented as a single point pattern object in which the points carry marks, and the mark value attached to each point determines the type of that point.

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to as.ppp. It must be a marked point pattern, and the mark vector X$marks must be a factor.

The argument i will be interpreted as a level of the factor X$marks. If i is missing, it defaults to the first level of the marks factor, i = levels(X$marks)[1].

The "type i to any type" multitype K function of a stationary multitype point process X is defined so that $\lambda K_{i\bullet}(r)$ equals the expected number of additional random points within a distance r of a typical point of type i in the process X. Here λ is the intensity of the process, i.e. the expected number of points of X per unit area. The function $K_{i\bullet}$ is determined by the second order moment properties of X.

An estimate of $K_{i\bullet}(r)$ is a useful summary statistic in exploratory data analysis of a multitype point pattern. If the subprocess of type i points were independent of the subprocess of points of all types not equal to i, then $K_{i\bullet}(r)$ would equal πr^2. Deviations between the empirical $K_{i\bullet}$ curve and the theoretical curve πr^2 may suggest dependence between types.

This algorithm estimates the distribution function $K_{i\bullet}(r)$ from the point pattern X. It assumes that X can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in X as Window(X)) may have arbitrary shape. Biases due to edge effects are treated in the same manner as in Kest, using the chosen edge correction(s).

The argument r is the vector of values for the distance r at which $K_{i\bullet}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

The pair correlation function can also be applied to the result of Kdot; see pcf.

Value

An object of class "fv" (see fv.object).

Essentially a data frame containing numeric columns

r	the values of the argument r at which the function $K_{i\bullet}(r)$ has been estimated
theo	the theoretical value of $K_{i\bullet}(r)$ for a marked Poisson process, namely πr^2

together with a column or columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $K_{i\bullet}(r)$ obtained by the edge corrections named.

If ratio=TRUE then the return value also has two attributes called "numerator" and "denominator" which are "fv" objects containing the numerators and denominators of each estimate of $K(r)$.

Warnings

The argument `i` is interpreted as a level of the factor `X$marks`. It is converted to a character string if it is not already a character string. The value `i=1` does **not** refer to the first level of the factor.

The reduced sample estimator of $K_{i\bullet}$ is pointwise approximately unbiased, but need not be a valid distribution function; it may not be a nondecreasing function of r.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>.

References

Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.

Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.

Harkness, R.D and Isham, V. (1983) A bivariate spatial point pattern of ants' nests. *Applied Statistics* **32**, 293–303

Lotwick, H. W. and Silverman, B. W. (1982). Methods for analysing spatial processes of several types of points. *J. Royal Statist. Soc. Ser. B* **44**, 406–413.

Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.

Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.

See Also

`Kdot`, `Kest`, `Kmulti`, `pcf`

Examples

```
 # Lansing woods data: 6 types of trees
woods <- lansing



Kh. <- Kdot(woods, "hickory")
# diagnostic plot for independence between hickories and other trees
plot(Kh.)

# synthetic example with two marks "a" and "b"

  pp <- runifpoispp(50)
  pp <- pp %mark% factor(sample(c("a","b"), npoints(pp), replace=TRUE))
  K <- Kdot(pp, "a")
```

Kdot.inhom *Inhomogeneous Multitype K Dot Function*

Description

For a multitype point pattern, estimate the inhomogeneous version of the dot K function, which counts the expected number of points of any type within a given distance of a point of type i, adjusted for spatially varying intensity.

Usage

```
Kdot.inhom(X, i, lambdaI=NULL, lambdadot=NULL, ..., r=NULL, breaks=NULL,
         correction = c("border", "isotropic", "Ripley", "translate"),
         sigma=NULL, varcov=NULL, lambdaIdot=NULL,
         lambdaX=NULL, update=TRUE, leaveoneout=TRUE)
```

Arguments

X	The observed point pattern, from which an estimate of the inhomogeneous dot type K function $K_{i\bullet}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of marks(X).
lambdaI	Optional. Values of the estimated intensity of the sub-process of points of type i. Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the type i points in X, a fitted point process model (object of class "ppm" or "kppm" or "dppm"), or a function(x,y) which can be evaluated to give the intensity value at any location.
lambdadot	Optional. Values of the estimated intensity of the entire point process, Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the points in X, a fitted point process model (object of class "ppm" or "kppm" or "dppm"), or a function(x,y) which can be evaluated to give the intensity value at any location.
...	Ignored.
r	Optional. Numeric vector giving the values of the argument r at which the dot K function $K_{i\bullet}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
breaks	This argument is for internal use only.
correction	A character vector containing any selection of the options "border", "bord.modif", "isotropic", "Ripley", "translate", "translation", "none" or "best". It specifies the edge correction(s) to be applied. Alternatively correction="all" selects all options.

sigma	Standard deviation of isotropic Gaussian smoothing kernel, used in computing leave-one-out kernel estimates of lambdaI, lambdadot if they are omitted.
varcov	Variance-covariance matrix of anisotropic Gaussian kernel, used in computing leave-one-out kernel estimates of lambdaI, lambdadot if they are omitted. Incompatible with sigma.
lambdaIdot	Optional. A matrix containing estimates of the product of the intensities lambdaI and lambdadot for each pair of points, the first point of type i and the second of any type.
lambdaX	Optional. Values of the intensity for all points of X. Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the points in X, a fitted point process model (object of class "ppm" or "kppm" or "dppm"), or a function(x,y) which can be evaluated to give the intensity value at any location. If present, this argument overrides both lambdaI and lambdadot.
update	Logical value indicating what to do when lambdaI, lambdadot or lambdaX is a fitted point process model (class "ppm", "kppm" or "dppm"). If update=TRUE (the default), the model will first be refitted to the data X (using update.ppm or update.kppm) before the fitted intensity is computed. If update=FALSE, the fitted intensity of the model will be computed without re-fitting it to X.
leaveoneout	Logical value (passed to density.ppp or fitted.ppm) specifying whether to use a leave-one-out rule when calculating the intensity.

Details

This is a generalisation of the function Kdot to include an adjustment for spatially inhomogeneous intensity, in a manner similar to the function Kinhom.

Briefly, given a multitype point process, consider the points without their types, and suppose this unmarked point process has intensity function $\lambda(u)$ at spatial locations u. Suppose we place a mass of $1/\lambda(\zeta)$ at each point ζ of the process. Then the expected total mass per unit area is 1. The inhomogeneous "dot-type" K function $K_{i\bullet}^{\mathrm{inhom}}(r)$ equals the expected total mass within a radius r of a point of the process of type i, discounting this point itself.

If the process of type i points were independent of the points of other types, then $K_{i\bullet}^{\mathrm{inhom}}(r)$ would equal πr^2. Deviations between the empirical $K_{i\bullet}$ curve and the theoretical curve πr^2 suggest dependence between the points of types i and j for $j \neq i$.

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to as.ppp. It must be a marked point pattern, and the mark vector X$marks must be a factor.

The argument i will be interpreted as a level of the factor X$marks. (Warning: this means that an integer value i=3 will be interpreted as the number 3, **not** the 3rd smallest level). If i is missing, it defaults to the first level of the marks factor, i = levels(X$marks)[1].

The argument lambdaI supplies the values of the intensity of the sub-process of points of type i. It may be either

a pixel image (object of class "im") which gives the values of the type i intensity at all locations in the window containing X;

a numeric vector containing the values of the type i intensity evaluated only at the data points of type i. The length of this vector must equal the number of type i points in X.

a function of the form `function(x,y)` which can be evaluated to give values of the intensity at any locations.

a fitted point process model (object of class `"ppm"`, `"kppm"` or `"dppm"`) whose fitted *trend* can be used as the fitted intensity. (If `update=TRUE` the model will first be refitted to the data `X` before the trend is computed.)

omitted: if `lambdaI` is omitted then it will be estimated using a leave-one-out kernel smoother.

If `lambdaI` is omitted, then it will be estimated using a 'leave-one-out' kernel smoother, as described in Baddeley, Møller and Waagepetersen (2000). The estimate of `lambdaI` for a given point is computed by removing the point from the point pattern, applying kernel smoothing to the remaining points using `density.ppp`, and evaluating the smoothed intensity at the point in question. The smoothing kernel bandwidth is controlled by the arguments `sigma` and `varcov`, which are passed to `density.ppp` along with any extra arguments.

Similarly the argument `lambdadot` should contain estimated values of the intensity of the entire point process. It may be either a pixel image, a numeric vector of length equal to the number of points in `X`, a function, or omitted.

Alternatively if the argument `lambdaX` is given, then it specifies the intensity values for all points of `X`, and the arguments `lambdaI`, `lambdadot` will be ignored. (The two arguments `lambdaI`, `lambdadot` allow the user to specify two different methods for calculating the intensities of the two kinds of points, while `lambdaX` ensures that the same method is used for both kinds of points.)

For advanced use only, the optional argument `lambdaIdot` is a matrix containing estimated values of the products of these two intensities for each pair of points, the first point of type `i` and the second of any type.

The argument `r` is the vector of values for the distance r at which $K_{i\bullet}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

The argument `correction` chooses the edge correction as explained e.g. in `Kest`.

The pair correlation function can also be applied to the result of `Kdot.inhom`; see `pcf`.

Value

An object of class `"fv"` (see `fv.object`).

Essentially a data frame containing numeric columns

`r`	the values of the argument r at which the function $K_{i\bullet}(r)$ has been estimated
`theo`	the theoretical value of $K_{i\bullet}(r)$ for a marked Poisson process, namely πr^2

together with a column or columns named `"border"`, `"bord.modif"`, `"iso"` and/or `"trans"`, according to the selected edge corrections. These columns contain estimates of the function $K_{i\bullet}(r)$ obtained by the edge corrections named.

Warnings

The argument `i` is interpreted as a level of the factor `X$marks`. It is converted to a character string if it is not already a character string. The value `i=1` does **not** refer to the first level of the factor.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>

References

Møller, J. and Waagepetersen, R. Statistical Inference and Simulation for Spatial Point Processes Chapman and Hall/CRC Boca Raton, 2003.

See Also

Kdot, Kinhom, Kcross.inhom, Kmulti.inhom, pcf

Examples

```
# Lansing Woods data
woods <- lansing
woods <- woods[seq(1,npoints(woods), by=10)]
ma <- split(woods)$maple
lg <- unmark(woods)

# Estimate intensities by nonparametric smoothing
lambdaM <- density.ppp(ma, sigma=0.15, at="points")
lambdadot <- density.ppp(lg, sigma=0.15, at="points")
K <- Kdot.inhom(woods, "maple", lambdaI=lambdaM,
                                  lambdadot=lambdadot)

# Equivalent
K <- Kdot.inhom(woods, "maple", sigma=0.15)

# Fit model
if(require("spatstat.model")) {
fit <- ppm(woods ~ marks * polynom(x,y,2))
K <- Kdot.inhom(woods, "maple", lambdaX=fit,
               update=FALSE, leaveoneout=FALSE)
}

# synthetic example: type A points have intensity 50,
#                    type B points have intensity 50 + 100 * x
lamB <- as.im(function(x,y){50 + 100 * x}, owin())
lamdot <- as.im(function(x,y) { 100 + 100 * x}, owin())
X <- superimpose(A=runifpoispp(50), B=rpoispp(lamB))
K <- Kdot.inhom(X, "B",  lambdaI=lamB,     lambdadot=lamdot)
```

`Kest` *K-function*

Description

Estimates Ripley's reduced second moment function $K(r)$ from a point pattern in a window of arbitrary shape.

Usage

```
Kest(X, ..., r=NULL, rmax=NULL, breaks=NULL,
   correction=c("border", "isotropic", "Ripley", "translate"),
  nlarge=3000, domain=NULL, var.approx=FALSE, ratio=FALSE)
```

Arguments

X	The observed point pattern, from which an estimate of $K(r)$ will be computed. An object of class "ppp", or data in any format acceptable to as.ppp().
...	Ignored.
r	Optional. Vector of values for the argument r at which $K(r)$ should be evaluated. Users are advised *not* to specify this argument; there is a sensible default. If necessary, specify rmax.
rmax	Optional. Maximum desired value of the argument r.
breaks	This argument is for internal use only.
correction	Optional. A character vector containing any selection of the options "none", "border", "bord.modif", "isotropic", "Ripley", "translate", "translation", "rigid", "none", "periodic", "good" or "best". It specifies the edge correction(s) to be applied. Alternatively correction="all" selects all options.
nlarge	Optional. Efficiency threshold. If the number of points exceeds nlarge, then only the border correction will be computed (by default), using a fast algorithm.
domain	Optional. Calculations will be restricted to this subset of the window. See Details.
var.approx	Logical. If TRUE, the approximate variance of $\hat{K}(r)$ under CSR will also be computed.
ratio	Logical. If TRUE, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.

Details

The K function (variously called "Ripley's K-function" and the "reduced second moment function") of a stationary point process X is defined so that $\lambda K(r)$ equals the expected number of additional random points within a distance r of a typical random point of X. Here λ is the intensity of the process, i.e. the expected number of points of X per unit area. The K function is determined by the second order moment properties of X.

An estimate of K derived from a spatial point pattern dataset can be used in exploratory data analysis and formal inference about the pattern (Cressie, 1991; Diggle, 1983; Ripley, 1977, 1988). In exploratory analyses, the estimate of K is a useful statistic summarising aspects of inter-point "dependence" and "clustering". For inferential purposes, the estimate of K is usually compared to the true value of K for a completely random (Poisson) point process, which is $K(r) = \pi r^2$. Deviations between the empirical and theoretical K curves may suggest spatial clustering or spatial regularity.

This routine `Kest` estimates the K function of a stationary point process, given observation of the process inside a known, bounded window. The argument `X` is interpreted as a point pattern object (of class `"ppp"`, see `ppp.object`) and can be supplied in any of the formats recognised by `as.ppp()`.

The estimation of K is hampered by edge effects arising from the unobservability of points of the random pattern outside the window. An edge correction is needed to reduce bias (Baddeley, 1998; Ripley, 1988). The corrections implemented here are

border the border method or "reduced sample" estimator (see Ripley, 1988). This is the least efficient (statistically) and the fastest to compute. It can be computed for a window of arbitrary shape.

isotropic/Ripley Ripley's isotropic correction (see Ripley, 1988; Ohser, 1983). This is implemented for rectangular and polygonal windows (not for binary masks).

translate/translation Translation correction (Ohser, 1983). Implemented for all window geometries, but slow for complex windows.

rigid Rigid motion correction (Ohser and Stoyan, 1981). Implemented for all window geometries, but slow for complex windows.

none Uncorrected estimate. An estimate of the K function *without* edge correction. (i.e. setting $e_{ij} = 1$ in the equation below. This estimate is **biased** and should not be used for data analysis, *unless* you have an extremely large point pattern (more than 100,000 points).

periodic Periodic (toroidal) edge correction. Defined only for rectangular windows.

best Selects the best edge correction that is available for the geometry of the window. Currently this is Ripley's isotropic correction for a rectangular or polygonal window, and the translation correction for masks.

good Selects the best edge correction that can be computed in a reasonable time. This is the same as `"best"` for datasets with fewer than 3000 points; otherwise the selected edge correction is `"border"`, unless there are more than 100,000 points, when it is `"none"`.

The estimates of $K(r)$ are of the form

$$\hat{K}(r) = \frac{a}{n(n-1)} \sum_i \sum_j I(d_{ij} \le r) e_{ij}$$

where a is the area of the window, n is the number of data points, and the sum is taken over all ordered pairs of points i and j in `X`. Here d_{ij} is the distance between the two points, and $I(d_{ij} \le r)$ is the indicator that equals 1 if the distance is less than or equal to r. The term e_{ij} is the edge correction weight (which depends on the choice of edge correction listed above).

Note that this estimator assumes the process is stationary (spatially homogeneous). For inhomogeneous point patterns, see `Kinhom`.

If the point pattern `X` contains more than about 3000 points, the isotropic and translation edge corrections can be computationally prohibitive. The computations for the border method are much faster, and are statistically efficient when there are large numbers of points. Accordingly, if the number of points in `X` exceeds the threshold `nlarge`, then only the border correction will be computed. Setting `nlarge=Inf` or `correction="best"` will prevent this from happening. Setting `nlarge=0` is equivalent to selecting only the border correction with `correction="border"`.

If `X` contains more than about 100,000 points, even the border correction is time-consuming. You may want to consider setting `correction="none"` in this case. There is an even faster algorithm for the uncorrected estimate.

Approximations to the variance of $\hat{K}(r)$ are available, for the case of the isotropic edge correction estimator, **assuming complete spatial randomness** (Ripley, 1988; Lotwick and Silverman, 1982; Diggle, 2003, pp 51-53). If `var.approx=TRUE`, then the result of `Kest` also has a column named `rip` giving values of Ripley's (1988) approximation to $\mathrm{var}(\hat{K}(r))$, and (if the window is a rectangle) a column named `ls` giving values of Lotwick and Silverman's (1982) approximation.

If the argument `domain` is given, the calculations will be restricted to a subset of the data. In the formula for $K(r)$ above, the *first* point i will be restricted to lie inside `domain`. The result is an approximately unbiased estimate of $K(r)$ based on pairs of points in which the first point lies inside `domain` and the second point is unrestricted. This is useful in bootstrap techniques. The argument `domain` should be a window (object of class `"owin"`) or something acceptable to `as.owin`. It must be a subset of the window of the point pattern `X`.

The estimator `Kest` ignores marks. Its counterparts for multitype point patterns are `Kcross`, `Kdot`, and for general marked point patterns see `Kmulti`.

Some writers, particularly Stoyan (1994, 1995) advocate the use of the "pair correlation function"

$$g(r) = \frac{K'(r)}{2\pi r}$$

where $K'(r)$ is the derivative of $K(r)$. See `pcf` on how to estimate this function.

Value

An object of class `"fv"`, see `fv.object`, which can be plotted directly using `plot.fv`.

Essentially a data frame containing columns

`r`	the vector of values of the argument r at which the function K has been estimated
`theo`	the theoretical value $K(r) = \pi r^2$ for a stationary Poisson process

together with columns named `"border"`, `"bord.modif"`, `"iso"` and/or `"trans"`, according to the selected edge corrections. These columns contain estimates of the function $K(r)$ obtained by the edge corrections named.

If `var.approx=TRUE` then the return value also has columns `rip` and `ls` containing approximations to the variance of $\hat{K}(r)$ under CSR.

If `ratio=TRUE` then the return value also has two attributes called `"numerator"` and `"denominator"` which are `"fv"` objects containing the numerators and denominators of each estimate of $K(r)$.

Envelopes, significance bands and confidence intervals

To compute simulation envelopes for the K-function under CSR, use `envelope`.

To compute a confidence interval for the true K-function, use `varblock` or `lohboot`.

Warnings

The estimator of $K(r)$ is approximately unbiased for each fixed r, for point processes which do not have very strong interaction. (For point processes with a strong clustering interaction, the estimator is negatively biased; for point processes with a strong inhibitive interaction, the estimator is positively biased.)

Bias increases with r and depends on the window geometry. For a rectangular window it is prudent to restrict the r values to a maximum of $1/4$ of the smaller side length of the rectangle (Ripley, 1977, 1988; Diggle, 1983). Bias may become appreciable for point patterns consisting of fewer than 15 points.

While $K(r)$ is always a non-decreasing function, the estimator of K is not guaranteed to be non-decreasing. This is rarely a problem in practice, except for the border correction estimators when the number of points is small.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

References

Baddeley, A.J. Spatial sampling and censoring. In O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*. Chapman and Hall, 1998. Chapter 2, pages 37–78.

Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.

Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.

Ohser, J. (1983) On estimators for the reduced second moment measure of point processes. *Mathematische Operationsforschung und Statistik, series Statistics*, **14**, 63 – 71.

Ohser, J. and Stoyan, D. (1981) On the second-order and orientation analysis of planar stationary point processes. *Biometrical Journal* **23**, 523–533.

Ripley, B.D. (1977) Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, Series B*, **39**, 172 – 212.

Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.

Stoyan, D, Kendall, W.S. and Mecke, J. (1995) *Stochastic geometry and its applications*. 2nd edition. Springer Verlag.

Stoyan, D. and Stoyan, H. (1994) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

See Also

`localK` to extract individual summands in the K function.

`pcf` for the pair correlation.

`Fest`, `Gest`, `Jest` for alternative summary functions.

`Kcross`, `Kdot`, `Kinhom`, `Kmulti` for counterparts of the K function for multitype point patterns.

`reduced.sample` for the calculation of reduced sample estimators.

Examples

```
X <- runifpoint(50)
K <- Kest(X)
K <- Kest(cells, correction="isotropic")
plot(K)
plot(K, main="K function for cells")
```

```
# plot the L function
plot(K, sqrt(iso/pi) ~ r)
plot(K, sqrt(./pi) ~ r, ylab="L(r)", main="L function for cells")
```

`Kest.fft` *K-function using FFT*

Description

Estimates the reduced second moment function $K(r)$ from a point pattern in a window of arbitrary shape, using the Fast Fourier Transform.

Usage

```
Kest.fft(X, sigma, r=NULL, ..., breaks=NULL)
```

Arguments

`X`	The observed point pattern, from which an estimate of $K(r)$ will be computed. An object of class `"ppp"`, or data in any format acceptable to `as.ppp()`.
`sigma`	Standard deviation of the isotropic Gaussian smoothing kernel.
`r`	Optional. Vector of values for the argument r at which $K(r)$ should be evaluated. There is a sensible default.
`...`	Arguments passed to `as.mask` determining the spatial resolution for the FFT calculation.
`breaks`	This argument is for internal use only.

Details

This is an alternative to the function `Kest` for estimating the K function. It may be useful for very large patterns of points.

Whereas `Kest` computes the distance between each pair of points analytically, this function discretises the point pattern onto a rectangular pixel raster and applies Fast Fourier Transform techniques to estimate $K(t)$. The hard work is done by the function `Kmeasure`.

The result is an approximation whose accuracy depends on the resolution of the pixel raster. The resolution is controlled by the arguments `...`, or by setting the parameter `npixel` in `spatstat.options`.

Value

An object of class `"fv"` (see `fv.object`).

Essentially a data frame containing columns

`r`	the vector of values of the argument r at which the function K has been estimated
`border`	the estimates of $K(r)$ for these values of r
`theo`	the theoretical value $K(r) = \pi r^2$ for a stationary Poisson process

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>

References

Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.

Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.

Ohser, J. (1983) On estimators for the reduced second moment measure of point processes. *Mathematische Operationsforschung und Statistik, series Statistics*, **14**, 63 – 71.

Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.

Stoyan, D, Kendall, W.S. and Mecke, J. (1995) *Stochastic geometry and its applications*. 2nd edition. Springer Verlag.

Stoyan, D. and Stoyan, H. (1994) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

See Also

Kest, Kmeasure, spatstat.options

Examples

```
pp <- runifpoint(10000)

Kpp <- Kest.fft(pp, 0.01)
plot(Kpp)
```

Kinhom *Inhomogeneous K-function*

Description

Estimates the inhomogeneous K function of a non-stationary point pattern.

Usage

```
Kinhom(X, lambda=NULL, ..., r = NULL, breaks = NULL,
  correction=c("border", "bord.modif", "isotropic", "translate"),
  renormalise=TRUE,
  normpower=1,
  update=TRUE,
  leaveoneout=TRUE,
  nlarge = 1000,
  lambda2=NULL, reciplambda=NULL, reciplambda2=NULL,
  diagonal=TRUE,
  sigma=NULL, varcov=NULL,
  ratio=FALSE)
```

Arguments

X	The observed data point pattern, from which an estimate of the inhomogeneous K function will be computed. An object of class "ppp" or in a format recognised by as.ppp()
lambda	Optional. Values of the estimated intensity function. Either a vector giving the intensity values at the points of the pattern X, a pixel image (object of class "im") giving the intensity values at all locations, a fitted point process model (object of class "ppm" or "kppm") or a function(x,y) which can be evaluated to give the intensity value at any location.
...	Extra arguments. Ignored if lambda is present. Passed to density.ppp if lambda is omitted.
r	vector of values for the argument r at which the inhomogeneous K function should be evaluated. Not normally given by the user; there is a sensible default.
breaks	This argument is for internal use only.
correction	A character vector containing any selection of the options "border", "bord.modif", "isotropic", "Ripley", "translate", "translation", "none" or "best". It specifies the edge correction(s) to be applied. Alternatively correction="all" selects all options.
renormalise	Logical. Whether to renormalise the estimate. See Details.
normpower	Integer (usually either 1 or 2). Normalisation power. See Details.
update	Logical value indicating what to do when lambda is a fitted model (class "ppm", "kppm" or "dppm"). If update=TRUE (the default), the model will first be refitted to the data X (using update.ppm or update.kppm) before the fitted intensity is computed. If update=FALSE, the fitted intensity of the model will be computed without re-fitting it to X.
leaveoneout	Logical value (passed to density.ppp or fitted.ppm) specifying whether to use a leave-one-out rule when calculating the intensity.
nlarge	Optional. Efficiency threshold. If the number of points exceeds nlarge, then only the border correction will be computed, using a fast algorithm.
lambda2	Advanced use only. Matrix containing estimates of the products $\lambda(x_i)\lambda(x_j)$ of the intensities at each pair of data points x_i and x_j.
reciplambda	Alternative to lambda. Values of the estimated *reciprocal* $1/\lambda$ of the intensity function. Either a vector giving the reciprocal intensity values at the points of the pattern X, a pixel image (object of class "im") giving the reciprocal intensity values at all locations, or a function(x,y) which can be evaluated to give the reciprocal intensity value at any location.
reciplambda2	Advanced use only. Alternative to lambda2. A matrix giving values of the estimated *reciprocal products* $1/\lambda(x_i)\lambda(x_j)$ of the intensities at each pair of data points x_i and x_j.
diagonal	Do not use this argument.
sigma, varcov	Optional arguments passed to density.ppp to control the smoothing bandwidth, when lambda is estimated by kernel smoothing.
ratio	Logical. If TRUE, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.

Details

This computes a generalisation of the K function for inhomogeneous point patterns, proposed by Baddeley, Møller and Waagepetersen (2000).

The "ordinary" K function (variously known as the reduced second order moment function and Ripley's K function), is described under `Kest`. It is defined only for stationary point processes.

The inhomogeneous K function $K_{\mathrm{inhom}}(r)$ is a direct generalisation to nonstationary point processes. Suppose x is a point process with non-constant intensity $\lambda(u)$ at each location u. Define $K_{\mathrm{inhom}}(r)$ to be the expected value, given that u is a point of x, of the sum of all terms $1/\lambda(x_j)$ over all points x_j in the process separated from u by a distance less than r. This reduces to the ordinary K function if $\lambda()$ is constant. If x is an inhomogeneous Poisson process with intensity function $\lambda(u)$, then $K_{\mathrm{inhom}}(r) = \pi r^2$.

Given a point pattern dataset, the inhomogeneous K function can be estimated essentially by summing the values $1/(\lambda(x_i)\lambda(x_j))$ for all pairs of points x_i, x_j separated by a distance less than r.

This allows us to inspect a point pattern for evidence of interpoint interactions after allowing for spatial inhomogeneity of the pattern. Values $K_{\mathrm{inhom}}(r) > \pi r^2$ are suggestive of clustering.

The argument `lambda` should supply the (estimated) values of the intensity function λ. It may be either

a numeric vector containing the values of the intensity function at the points of the pattern `X`.

a pixel image (object of class `"im"`) assumed to contain the values of the intensity function at all locations in the window.

a fitted point process model (object of class `"ppm"`, `"kppm"` or `"dppm"`) whose fitted *trend* can be used as the fitted intensity. (If `update=TRUE` the model will first be refitted to the data `X` before the trend is computed.)

a function which can be evaluated to give values of the intensity at any locations.

omitted: if `lambda` is omitted, then it will be estimated using a 'leave-one-out' kernel smoother.

If `lambda` is a numeric vector, then its length should be equal to the number of points in the pattern `X`. The value `lambda[i]` is assumed to be the the (estimated) value of the intensity $\lambda(x_i)$ for the point x_i of the pattern X. Each value must be a positive number; `NA`'s are not allowed.

If `lambda` is a pixel image, the domain of the image should cover the entire window of the point pattern. If it does not (which may occur near the boundary because of discretisation error), then the missing pixel values will be obtained by applying a Gaussian blur to `lambda` using `blur`, then looking up the values of this blurred image for the missing locations. (A warning will be issued in this case.)

If `lambda` is a function, then it will be evaluated in the form `lambda(x,y)` where `x` and `y` are vectors of coordinates of the points of `X`. It should return a numeric vector with length equal to the number of points in `X`.

If `lambda` is omitted, then it will be estimated using a 'leave-one-out' kernel smoother, as described in Baddeley, Møller and Waagepetersen (2000). The estimate `lambda[i]` for the point `X[i]` is computed by removing `X[i]` from the point pattern, applying kernel smoothing to the remaining points using `density.ppp`, and evaluating the smoothed intensity at the point `X[i]`. The smoothing kernel bandwidth is controlled by the arguments `sigma` and `varcov`, which are passed to `density.ppp` along with any extra arguments.

Edge corrections are used to correct bias in the estimation of K_{inhom}. Each edge-corrected estimate of $K_{\mathrm{inhom}}(r)$ is of the form

$$\widehat{K}_{\mathrm{inhom}}(r) = (1/A) \sum_i \sum_j \frac{1\{d_{ij} \le r\} e(x_i, x_j, r)}{\lambda(x_i)\lambda(x_j)}$$

where `A` is a constant denominator, d_{ij} is the distance between points x_i and x_j, and $e(x_i, x_j, r)$ is an edge correction factor. For the 'border' correction,

$$e(x_i, x_j, r) = \frac{1(b_i > r)}{\sum_j 1(b_j > r)/\lambda(x_j)}$$

where b_i is the distance from x_i to the boundary of the window. For the 'modified border' correction,

$$e(x_i, x_j, r) = \frac{1(b_i > r)}{\mathrm{area}(W \ominus r)}$$

where $W \ominus r$ is the eroded window obtained by trimming a margin of width r from the border of the original window. For the 'translation' correction,

$$e(x_i, x_j, r) = \frac{1}{\mathrm{area}(W \cap (W + (x_j - x_i)))}$$

and for the 'isotropic' correction,

$$e(x_i, x_j, r) = \frac{1}{\mathrm{area}(W) g(x_i, x_j)}$$

where $g(x_i, x_j)$ is the fraction of the circumference of the circle with centre x_i and radius $||x_i - x_j||$ which lies inside the window.

If `renormalise=TRUE` (the default), then the estimates described above are multiplied by $c^{\mathrm{normpower}}$ where $c = \mathrm{area}(W)/\sum(1/\lambda(x_i))$. This rescaling reduces the variability and bias of the estimate in small samples and in cases of very strong inhomogeneity. The default value of `normpower` is 1 (for consistency with previous versions of **spatstat**) but the most sensible value is 2, which would correspond to rescaling the `lambda` values so that $\sum(1/\lambda(x_i)) = \mathrm{area}(W)$.

If the point pattern `X` contains more than about 1000 points, the isotropic and translation edge corrections can be computationally prohibitive. The computations for the border method are much faster, and are statistically efficient when there are large numbers of points. Accordingly, if the number of points in `X` exceeds the threshold `nlarge`, then only the border correction will be computed. Setting `nlarge=Inf` or `correction="best"` will prevent this from happening. Setting `nlarge=0` is equivalent to selecting only the border correction with `correction="border"`.

The pair correlation function can also be applied to the result of `Kinhom`; see `pcf`.

Value

An object of class `"fv"` (see `fv.object`).

Essentially a data frame containing at least the following columns,

`r`	the vector of values of the argument r at which $K_{\mathrm{inhom}}(r)$ has been estimated

theo — vector of values of πr^2, the theoretical value of $K_{\mathrm{inhom}}(r)$ for an inhomogeneous Poisson process

and containing additional columns according to the choice specified in the `correction` argument. The additional columns are named `border`, `trans` and `iso` and give the estimated values of $K_{\mathrm{inhom}}(r)$ using the border correction, translation correction, and Ripley isotropic correction, respectively.

If `ratio=TRUE` then the return value also has two attributes called `"numerator"` and `"denominator"` which are `"fv"` objects containing the numerators and denominators of each estimate of $K_{\mathrm{inhom}}(r)$.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

References

Baddeley, A., Møller, J. and Waagepetersen, R. (2000) Non- and semiparametric estimation of interaction in inhomogeneous point patterns. *Statistica Neerlandica* **54**, 329–350.

See Also

Kest, pcf

Examples

```
# inhomogeneous pattern of maples
X <- unmark(split(lansing)$maple)


if(require("spatstat.model")) {
  # (1) intensity function estimated by model-fitting
  # Fit spatial trend: polynomial in x and y coordinates
  fit <- ppm(X, ~ polynom(x,y,2), Poisson())
  # (a) predict intensity values at points themselves,
  #     obtaining a vector of lambda values
  lambda <- predict(fit, locations=X, type="trend")
  # inhomogeneous K function
  Ki <- Kinhom(X, lambda)
  plot(Ki)
  # (b) predict intensity at all locations,
  #     obtaining a pixel image
  lambda <- predict(fit, type="trend")
  Ki <- Kinhom(X, lambda)
  plot(Ki)
}

# (2) intensity function estimated by heavy smoothing
Ki <- Kinhom(X, sigma=0.1)
plot(Ki)

# (3) simulated data: known intensity function
lamfun <- function(x,y) { 50 + 100 * x }
```

```
# inhomogeneous Poisson process
Y <- rpoispp(lamfun, 150, owin())
# inhomogeneous K function
Ki <- Kinhom(Y, lamfun)
plot(Ki)

# How to make simulation envelopes:
#      Example shows method (2)
if(interactive()) {
  smo <- density.ppp(X, sigma=0.1)
  Ken <- envelope(X, Kinhom, nsim=99,
                  simulate=expression(rpoispp(smo)),
                  sigma=0.1, correction="trans")
  plot(Ken)
}
```

Kmark *Mark-Weighted K Function*

Description

Estimates the mark-weighted K function of a marked point pattern.

Usage

```
Kmark(X, f = NULL, r = NULL,
      correction = c("isotropic", "Ripley", "translate"), ...,
      f1 = NULL, normalise = TRUE, returnL = FALSE, fargs = NULL)

markcorrint(X, f = NULL, r = NULL,
            correction = c("isotropic", "Ripley", "translate"), ...,
            f1 = NULL, normalise = TRUE, returnL = FALSE, fargs = NULL)
```

Arguments

X	The observed point pattern. An object of class "ppp" or something acceptable to as.ppp.
f	Optional. Test function f used in the definition of the mark correlation function. An R function with at least two arguments. There is a sensible default.
r	Optional. Numeric vector. The values of the argument r at which the mark correlation function $k_f(r)$ should be evaluated. There is a sensible default.
correction	A character vector containing any selection of the options "isotropic", "Ripley" or "translate". It specifies the edge correction(s) to be applied. Alternatively correction="all" selects all options.
...	Ignored.
f1	An alternative to f. If this argument is given, then f is assumed to take the form $f(u, v) = f_1(u)f_1(v)$.

normalise	If normalise=FALSE, compute only the numerator of the expression for the mark correlation.
returnL	Compute the analogue of the K-function if returnL=FALSE or the analogue of the L-function if returnL=TRUE.
fargs	Optional. A list of extra arguments to be passed to the function f or f1.

Details

The functions Kmark and markcorrint are identical. (Eventually markcorrint will be deprecated.)

The *mark-weighted K function* $K_f(r)$ of a marked point process (Penttinen et al, 1992) is a generalisation of Ripley's K function, in which the contribution from each pair of points is weighted by a function of their marks. If the marks of the two points are m_1, m_2 then the weight is proportional to $f(m_1, m_2)$ where f is a specified *test function*.

The mark-weighted K function is defined so that

$$\lambda K_f(r) = \frac{C_f(r)}{E[f(M_1, M_2)]}$$

where

$$C_f(r) = E\left[\sum_{x \in X} f(m(u), m(x)) 10 < ||u - x|| \le r \;\middle|\; u \in X\right]$$

for any spatial location u taken to be a typical point of the point process X. Here $||u - x||$ is the euclidean distance between u and x, so that the sum is taken over all random points x that lie within a distance r of the point u. The function $C_f(r)$ is the *unnormalised* mark-weighted K function. To obtain $K_f(r)$ we standardise $C_f(r)$ by dividing by $E[f(M_1, M_2)]$, the expected value of $f(M_1, M_2)$ when M_1 and M_2 are independent random marks with the same distribution as the marks in the point process.

Under the hypothesis of random labelling, the mark-weighted K function is equal to Ripley's K function, $K_f(r) = K(r)$.

The mark-weighted K function is sometimes called the *mark correlation integral* because it is related to the mark correlation function $k_f(r)$ and the pair correlation function $g(r)$ by

$$K_f(r) = 2\pi \int_0^r s k_f(s)\, g(s)\, \mathrm{d}s$$

See markcorr for a definition of the mark correlation function.

Given a marked point pattern X, this command computes edge-corrected estimates of the mark-weighted K function. If returnL=FALSE then the estimated function $K_f(r)$ is returned; otherwise the function

$$L_f(r) = \sqrt{K_f(r)/\pi}$$

is returned.

Value

An object of class "fv" (see fv.object).

Essentially a data frame containing numeric columns

r	the values of the argument r at which the mark correlation integral $K_f(r)$ has been estimated
theo	the theoretical value of $K_f(r)$ when the marks attached to different points are independent, namely πr^2

together with a column or columns named "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the mark-weighted K function $K_f(r)$ obtained by the edge corrections named (if returnL=FALSE).

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

References

Penttinen, A., Stoyan, D. and Henttonen, H. M. (1992) Marked point processes in forest statistics. *Forest Science* **38** (1992) 806-824.

Illian, J., Penttinen, A., Stoyan, H. and Stoyan, D. (2008) *Statistical analysis and modelling of spatial point patterns*. Chichester: John Wiley.

See Also

markcorr to estimate the mark correlation function.

Examples

```
# CONTINUOUS-VALUED MARKS:
# (1) Spruces
# marks represent tree diameter
# mark correlation function
ms <- Kmark(spruces)
plot(ms)

# (2) simulated data with independent marks
X <- rpoispp(100)
X <- X %mark% runif(npoints(X))
Xc <- Kmark(X)
plot(Xc)

# MULTITYPE DATA:
# Hughes' amacrine data
# Cells marked as 'on'/'off'
M <- Kmark(amacrine, function(m1,m2) {m1==m2},
                     correction="translate")
plot(M)
```

`Kmeasure` *Reduced Second Moment Measure*

Description

Estimates the reduced second moment measure κ from a point pattern in a window of arbitrary shape.

Usage

```
Kmeasure(X, sigma, edge=TRUE, ..., varcov=NULL)
```

Arguments

`X`	The observed point pattern, from which an estimate of κ will be computed. An object of class `"ppp"`, or data in any format acceptable to `as.ppp()`.
`sigma`	Standard deviation σ of the Gaussian smoothing kernel. Incompatible with `varcov`.
`edge`	Logical value indicating whether an edge correction should be applied.
`...`	Arguments passed to `as.mask` controlling the pixel resolution.
`varcov`	Variance-covariance matrix of the Gaussian smoothing kernel. Incompatible with `sigma`.

Details

Given a point pattern dataset, this command computes an estimate of the reduced second moment measure κ of the point process. The result is a pixel image whose pixel values are estimates of the density of the reduced second moment measure.

The reduced second moment measure κ can be regarded as a generalisation of the more familiar K-function. An estimate of κ derived from a spatial point pattern dataset can be useful in exploratory data analysis. Its advantage over the K-function is that it is also sensitive to anisotropy and directional effects.

In a nutshell, the command `Kmeasure` computes a smoothed version of the *Fry plot*. As explained under `fryplot`, the Fry plot is a scatterplot of the vectors joining all pairs of points in the pattern. The reduced second moment measure is (essentially) defined as the average of the Fry plot over different realisations of the point process. The command `Kmeasure` effectively smooths the Fry plot of a dataset to obtain an estimate of the reduced second moment measure.

In formal terms, the reduced second moment measure κ of a stationary point process X is a measure defined on the two-dimensional plane such that, for a 'typical' point x of the process, the expected number of other points y of the process such that the vector $y - x$ lies in a region A, equals $\lambda\kappa(A)$. Here λ is the intensity of the process, i.e. the expected number of points of X per unit area.

The K-function is a special case. The function value $K(t)$ is the value of the reduced second moment measure for the disc of radius t centred at the origin; that is, $K(t) = \kappa(b(0,t))$.

The command Kmeasure computes an estimate of κ from a point pattern dataset X, which is assumed to be a realisation of a stationary point process, observed inside a known, bounded window. Marks are ignored.

The algorithm approximates the point pattern and its window by binary pixel images, introduces a Gaussian smoothing kernel and uses the Fast Fourier Transform fft to form a density estimate of κ. The calculation corresponds to the edge correction known as the "translation correction".

The Gaussian smoothing kernel may be specified by either of the arguments sigma or varcov. If sigma is a single number, this specifies an isotropic Gaussian kernel with standard deviation sigma on each coordinate axis. If sigma is a vector of two numbers, this specifies a Gaussian kernel with standard deviation sigma[1] on the x axis, standard deviation sigma[2] on the y axis, and zero correlation between the x and y axes. If varcov is given, this specifies the variance-covariance matrix of the Gaussian kernel. There do not seem to be any well-established rules for selecting the smoothing kernel in this context.

The density estimate of κ is returned in the form of a real-valued pixel image. Pixel values are estimates of the normalised second moment density at the centre of the pixel. (The uniform Poisson process would have values identically equal to 1.) The image x and y coordinates are on the same scale as vector displacements in the original point pattern window. The point x=0, y=0 corresponds to the 'typical point'. A peak in the image near (0,0) suggests clustering; a dip in the image near (0,0) suggests inhibition; peaks or dips at other positions suggest possible periodicity.

If desired, the value of $\kappa(A)$ for a region A can be estimated by computing the integral of the pixel image over the domain A, i.e.\ summing the pixel values and multiplying by pixel area, using integral.im. One possible application is to compute anisotropic counterparts of the K-function (in which the disc of radius t is replaced by another shape). See Examples.

Value

A real-valued pixel image (an object of class "im", see im.object) whose pixel values are estimates of the density of the reduced second moment measure at each location.

Warning

Some writers use the term *reduced second moment measure* when they mean the K-function. This has caused confusion.

As originally defined, the reduced second moment measure is a measure, obtained by modifying the second moment measure, while the K-function is a function obtained by evaluating this measure for discs of increasing radius. In **spatstat**, the K-function is computed by Kest and the reduced second moment measure is computed by Kmeasure.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

References

Stoyan, D, Kendall, W.S. and Mecke, J. (1995) *Stochastic geometry and its applications*. 2nd edition. Springer Verlag.

Stoyan, D. and Stoyan, H. (1994) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

See Also

Kest, fryplot, spatstat.options, integral.im, im.object

Examples

```
plot(Kmeasure(cells, 0.05))
# shows pronounced dip around origin consistent with strong inhibition
plot(Kmeasure(redwood, 0.03), col=grey(seq(1,0,length=32)))
# shows peaks at several places, reflecting clustering and ?periodicity
M <- Kmeasure(cells, 0.05)
# evaluate measure on a sector
W <- Window(M)
ang <- as.im(atan2, W)
rad <- as.im(function(x,y){sqrt(x^2+y^2)}, W)
sector <- solutionset(ang > 0 & ang < 1 & rad < 0.6)
integral.im(M[sector, drop=FALSE])
```

Kmulti *Marked K-Function*

Description

For a marked point pattern, estimate the multitype K function which counts the expected number of points of subset J within a given distance from a typical point in subset I.

Usage

```
Kmulti(X, I, J, r=NULL, breaks=NULL, correction, ..., rmax=NULL, ratio=FALSE)
```

Arguments

X	The observed point pattern, from which an estimate of the multitype K function $K_{IJ}(r)$ will be computed. It must be a marked point pattern. See under Details.
I	Subset index specifying the points of X from which distances are measured. See Details.
J	Subset index specifying the points in X to which distances are measured. See Details.
r	numeric vector. The values of the argument r at which the multitype K function $K_{IJ}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r. If necessary, specify rmax.
breaks	This argument is for internal use only.
correction	A character vector containing any selection of the options "border", "bord.modif", "isotropic", "Ripley", "translate", "translation", "periodic", "none" or "best". It specifies the edge correction(s) to be applied. Alternatively correction="all" selects all options.

...	Ignored.
rmax	Optional. Maximum desired value of the argument r.
ratio	Logical. If TRUE, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.

Details

The function Kmulti generalises Kest (for unmarked point patterns) and Kdot and Kcross (for multitype point patterns) to arbitrary marked point patterns.

Suppose X_I, X_J are subsets, possibly overlapping, of a marked point process. The multitype K function is defined so that $\lambda_J K_{IJ}(r)$ equals the expected number of additional random points of X_J within a distance r of a typical point of X_I. Here λ_J is the intensity of X_J i.e. the expected number of points of X_J per unit area. The function K_{IJ} is determined by the second order moment properties of X.

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to as.ppp.

The arguments I and J specify two subsets of the point pattern. They may be any type of subset indices, for example, logical vectors of length equal to npoints(X), or integer vectors with entries in the range 1 to npoints(X), or negative integer vectors.

Alternatively, I and J may be **functions** that will be applied to the point pattern X to obtain index vectors. If I is a function, then evaluating I(X) should yield a valid subset index. This option is useful when generating simulation envelopes using envelope.

The argument r is the vector of values for the distance r at which $K_{IJ}(r)$ should be evaluated. It is also used to determine the breakpoints (in the sense of hist) for the computation of histograms of distances.

First-time users would be strongly advised not to specify r. However, if it is specified, r must satisfy r[1] = 0, and max(r) must be larger than the radius of the largest disc contained in the window.

This algorithm assumes that X can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in X as Window(X)) may have arbitrary shape.

Biases due to edge effects are treated in the same manner as in Kest. The edge corrections implemented here are

border the border method or "reduced sample" estimator (see Ripley, 1988). This is the least efficient (statistically) and the fastest to compute. It can be computed for a window of arbitrary shape.

isotropic/Ripley Ripley's isotropic correction (see Ripley, 1988; Ohser, 1983). This is currently implemented only for rectangular and polygonal windows.

translate Translation correction (Ohser, 1983). Implemented for all window geometries.

The pair correlation function pcf can also be applied to the result of Kmulti.

Value

An object of class "fv" (see fv.object).

Essentially a data frame containing numeric columns

`r`	the values of the argument r at which the function $K_{IJ}(r)$ has been estimated
`theo`	the theoretical value of $K_{IJ}(r)$ for a marked Poisson process, namely πr^2

together with a column or columns named `"border"`, `"bord.modif"`, `"iso"` and/or `"trans"`, according to the selected edge corrections. These columns contain estimates of the function $K_{IJ}(r)$ obtained by the edge corrections named.

If `ratio=TRUE` then the return value also has two attributes called `"numerator"` and `"denominator"` which are `"fv"` objects containing the numerators and denominators of each estimate of $K(r)$.

Warnings

The function K_{IJ} is not necessarily differentiable.

The border correction (reduced sample) estimator of K_{IJ} used here is pointwise approximately unbiased, but need not be a nondecreasing function of r, while the true K_{IJ} must be nondecreasing.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.

Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.

Diggle, P. J. (1986). Displaced amacrine cells in the retina of a rabbit : analysis of a bivariate spatial point pattern. *J. Neurosci. Meth.* **18**, 115–125.

Harkness, R.D and Isham, V. (1983) A bivariate spatial point pattern of ants' nests. *Applied Statistics* **32**, 293–303

Lotwick, H. W. and Silverman, B. W. (1982). Methods for analysing spatial processes of several types of points. *J. Royal Statist. Soc. Ser. B* **44**, 406–413.

Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.

Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.

Van Lieshout, M.N.M. and Baddeley, A.J. (1999) Indices of dependence between types in multivariate point patterns. *Scandinavian Journal of Statistics* **26**, 511–532.

See Also

`Kcross`, `Kdot`, `Kest`, `pcf`

Examples

```
 # Longleaf Pine data: marks represent diameter
trees <- longleaf

K <- Kmulti(trees, marks(trees) <= 15, marks(trees) >= 25)
plot(K)
```

```
# functions determining subsets
f1 <- function(X) { marks(X) <= 15 }
f2 <- function(X) { marks(X) >= 15 }
K <- Kmulti(trees, f1, f2)
```

Kmulti.inhom *Inhomogeneous Marked K-Function*

Description

For a marked point pattern, estimate the inhomogeneous version of the multitype K function which counts the expected number of points of subset J within a given distance from a typical point in subset I, adjusted for spatially varying intensity.

Usage

```
Kmulti.inhom(X, I, J, lambdaI=NULL, lambdaJ=NULL,
        ...,
        r=NULL, breaks=NULL,
        correction=c("border", "isotropic", "Ripley", "translate"),
        lambdaIJ=NULL,
        sigma=NULL, varcov=NULL,
        lambdaX=NULL, update=TRUE, leaveoneout=TRUE)
```

Arguments

X	The observed point pattern, from which an estimate of the inhomogeneous multitype K function $K_{IJ}(r)$ will be computed. It must be a marked point pattern. See under Details.
I	Subset index specifying the points of X from which distances are measured. See Details.
J	Subset index specifying the points in X to which distances are measured. See Details.
lambdaI	Optional. Values of the estimated intensity of the sub-process X[I]. Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the points in X[I], a fitted point process model (object of class "ppm" or "kppm" or "dppm"), or a function(x,y) which can be evaluated to give the intensity value at any location.
lambdaJ	Optional. Values of the estimated intensity of the sub-process X[J]. Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the points in X[J], a fitted point process model (object of class "ppm" or "kppm" or "dppm"), or a function(x,y) which can be evaluated to give the intensity value at any location.
...	Ignored.

r	Optional. Numeric vector. The values of the argument r at which the multitype K function $K_{IJ}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
breaks	This argument is for internal use only.
correction	A character vector containing any selection of the options "border", "bord.modif", "isotropic", "Ripley", "translate", "none" or "best". It specifies the edge correction(s) to be applied. Alternatively correction="all" selects all options.
lambdaIJ	Optional. A matrix containing estimates of the product of the intensities lambdaI and lambdaJ for each pair of points, the first point belonging to subset I and the second point to subset J.
sigma, varcov	Optional arguments passed to density.ppp to control the smoothing bandwidth, when lambda is estimated by kernel smoothing.
lambdaX	Optional. Values of the intensity for all points of X. Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the points in X, a fitted point process model (object of class "ppm" or "kppm" or "dppm"), or a function(x,y) which can be evaluated to give the intensity value at any location. If present, this argument overrides both lambdaI and lambdaJ.
update	Logical value indicating what to do when lambdaI, lambdaJ or lambdaX is a fitted point process model (class "ppm", "kppm" or "dppm"). If update=TRUE (the default), the model will first be refitted to the data X (using update.ppm or update.kppm) before the fitted intensity is computed. If update=FALSE, the fitted intensity of the model will be computed without re-fitting it to X.
leaveoneout	Logical value (passed to density.ppp or fitted.ppm) specifying whether to use a leave-one-out rule when calculating the intensity.

Details

The function Kmulti.inhom is the counterpart, for spatially-inhomogeneous marked point patterns, of the multitype K function Kmulti.

Suppose X is a marked point process, with marks of any kind. Suppose X_I, X_J are two subprocesses, possibly overlapping. Typically X_I would consist of those points of X whose marks lie in a specified range of mark values, and similarly for X_J. Suppose that $\lambda_I(u)$, $\lambda_J(u)$ are the spatially-varying intensity functions of X_I and X_J respectively. Consider all the pairs of points (u, v) in the point process X such that the first point u belongs to X_I, the second point v belongs to X_J, and the distance between u and v is less than a specified distance r. Give this pair (u, v) the numerical weight $1/(\lambda_I(u)\lambda_J(u))$. Calculate the sum of these weights over all pairs of points as described. This sum (after appropriate edge-correction and normalisation) is the estimated inhomogeneous multitype K function.

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to as.ppp.

The arguments I and J specify two subsets of the point pattern. They may be any type of subset indices, for example, logical vectors of length equal to npoints(X), or integer vectors with entries in the range 1 to npoints(X), or negative integer vectors.

Alternatively, I and J may be **functions** that will be applied to the point pattern X to obtain index vectors. If I is a function, then evaluating I(X) should yield a valid subset index. This option is useful when generating simulation envelopes using envelope.

The argument lambdaI supplies the values of the intensity of the sub-process identified by index I. It may be either

a pixel image (object of class "im") which gives the values of the intensity of X[I] at all locations in the window containing X;

a numeric vector containing the values of the intensity of X[I] evaluated only at the data points of X[I]. The length of this vector must equal the number of points in X[I].

a function of the form function(x,y) which can be evaluated to give values of the intensity at any locations.

a fitted point process model (object of class "ppm", "kppm" or "dppm") whose fitted *trend* can be used as the fitted intensity. (If update=TRUE the model will first be refitted to the data X before the trend is computed.)

omitted: if lambdaI is omitted then it will be estimated using a leave-one-out kernel smoother.

If lambdaI is omitted, then it will be estimated using a 'leave-one-out' kernel smoother, as described in Baddeley, Møller and Waagepetersen (2000). The estimate of lambdaI for a given point is computed by removing the point from the point pattern, applying kernel smoothing to the remaining points using density.ppp, and evaluating the smoothed intensity at the point in question. The smoothing kernel bandwidth is controlled by the arguments sigma and varcov, which are passed to density.ppp along with any extra arguments.

Similarly lambdaJ supplies the values of the intensity of the sub-process identified by index J.

Alternatively if the argument lambdaX is given, then it specifies the intensity values for all points of X, and the arguments lambdaI, lambdaJ will be ignored.

The argument r is the vector of values for the distance r at which $K_{IJ}(r)$ should be evaluated. It is also used to determine the breakpoints (in the sense of hist) for the computation of histograms of distances.

First-time users would be strongly advised not to specify r. However, if it is specified, r must satisfy r[1] = 0, and max(r) must be larger than the radius of the largest disc contained in the window.

Biases due to edge effects are treated in the same manner as in Kinhom. The edge corrections implemented here are

border the border method or "reduced sample" estimator (see Ripley, 1988). This is the least efficient (statistically) and the fastest to compute. It can be computed for a window of arbitrary shape.

isotropic/Ripley Ripley's isotropic correction (see Ripley, 1988; Ohser, 1983). This is currently implemented only for rectangular windows.

translate Translation correction (Ohser, 1983). Implemented for all window geometries.

The pair correlation function pcf can also be applied to the result of Kmulti.inhom.

Value

An object of class "fv" (see fv.object).

Essentially a data frame containing numeric columns

r	the values of the argument r at which the function $K_{IJ}(r)$ has been estimated
theo	the theoretical value of $K_{IJ}(r)$ for a marked Poisson process, namely πr^2

together with a column or columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $K_{IJ}(r)$ obtained by the edge corrections named.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

References

Baddeley, A., Møller, J. and Waagepetersen, R. (2000) Non- and semiparametric estimation of interaction in inhomogeneous point patterns. *Statistica Neerlandica* **54**, 329–350.

See Also

Kmulti, Kdot.inhom, Kcross.inhom, pcf

Examples

```
# Finnish Pines data: marked by diameter and height
plot(finpines, which.marks="height")
II <- (marks(finpines)$height <= 2)
JJ <- (marks(finpines)$height > 3)
K <- Kmulti.inhom(finpines, II, JJ)
plot(K)
# functions determining subsets
f1 <- function(X) { marks(X)$height <= 2 }
f2 <- function(X) { marks(X)$height > 3 }
K <- Kmulti.inhom(finpines, f1, f2)
```

Kscaled *Locally Scaled K-function*

Description

Estimates the locally-rescaled K-function of a point process.

Usage

```
Kscaled(X, lambda=NULL, ..., r = NULL, breaks = NULL,
  rmax = 2.5,
  correction=c("border", "isotropic", "translate"),
  renormalise=FALSE, normpower=1,
  sigma=NULL, varcov=NULL)

Lscaled(...)
```

Arguments

X	The observed data point pattern, from which an estimate of the locally scaled K function will be computed. An object of class "ppp" or in a format recognised by as.ppp().
lambda	Optional. Values of the estimated intensity function. Either a vector giving the intensity values at the points of the pattern X, a pixel image (object of class "im") giving the intensity values at all locations, a function(x,y) which can be evaluated to give the intensity value at any location, or a fitted point process model (object of class "ppm").
...	Arguments passed from Lscaled to Kscaled and from Kscaled to density.ppp if lambda is omitted.
r	vector of values for the argument r at which the locally scaled K function should be evaluated. (These are rescaled distances.) Not normally given by the user; there is a sensible default.
breaks	This argument is for internal use only.
rmax	maximum value of the argument r that should be used. (This is the rescaled distance).
correction	A character vector containing any selection of the options "border", "isotropic", "Ripley", "translate", "translation", "none" or "best". It specifies the edge correction(s) to be applied. Alternatively correction="all" selects all options.
renormalise	Logical. Whether to renormalise the estimate. See Details.
normpower	Integer (usually either 1 or 2). Normalisation power. See Details.
sigma, varcov	Optional arguments passed to density.ppp to control the smoothing bandwidth, when lambda is estimated by kernel smoothing.

Details

Kscaled computes an estimate of the K function for a locally scaled point process. Lscaled computes the corresponding L function $L(r) = \sqrt{K(r)/\pi}$.

Locally scaled point processes are a class of models for inhomogeneous point patterns, introduced by Hahn et al (2003). They include inhomogeneous Poisson processes, and many other models.

The template K function of a locally-scaled process is a counterpart of the "ordinary" Ripley K function, in which the distances between points of the process are measured on a spatially-varying scale (such that the locally rescaled process has unit intensity).

The template K function is an indicator of interaction between the points. For an inhomogeneous Poisson process, the theoretical template K function is approximately equal to $K(r) = \pi r^2$. Values $K_{\mathrm{scaled}}(r) > \pi r^2$ are suggestive of clustering.

`Kscaled` computes an estimate of the template K function and `Lscaled` computes the corresponding L function $L(r) = \sqrt{K(r)/\pi}$.

The locally scaled interpoint distances are computed using an approximation proposed by Hahn (2007). The Euclidean distance between two points is multiplied by the average of the square roots of the intensity values at the two points.

The argument `lambda` should supply the (estimated) values of the intensity function λ. It may be either

a numeric vector containing the values of the intensity function at the points of the pattern `X`.

a pixel image (object of class `"im"`) assumed to contain the values of the intensity function at all locations in the window.

a function which can be evaluated to give values of the intensity at any locations.

omitted: if `lambda` is omitted, then it will be estimated using a 'leave-one-out' kernel smoother.

If `lambda` is a numeric vector, then its length should be equal to the number of points in the pattern `X`. The value `lambda[i]` is assumed to be the the (estimated) value of the intensity $\lambda(x_i)$ for the point x_i of the pattern X. Each value must be a positive number; `NA`'s are not allowed.

If `lambda` is a pixel image, the domain of the image should cover the entire window of the point pattern. If it does not (which may occur near the boundary because of discretisation error), then the missing pixel values will be obtained by applying a Gaussian blur to `lambda` using `blur`, then looking up the values of this blurred image for the missing locations. (A warning will be issued in this case.)

If `lambda` is a function, then it will be evaluated in the form `lambda(x,y)` where `x` and `y` are vectors of coordinates of the points of `X`. It should return a numeric vector with length equal to the number of points in `X`.

If `lambda` is omitted, then it will be estimated using a 'leave-one-out' kernel smoother, as described in Baddeley, Møller and Waagepetersen (2000). The estimate `lambda[i]` for the point `X[i]` is computed by removing `X[i]` from the point pattern, applying kernel smoothing to the remaining points using `density.ppp`, and evaluating the smoothed intensity at the point `X[i]`. The smoothing kernel bandwidth is controlled by the arguments `sigma` and `varcov`, which are passed to `density.ppp` along with any extra arguments.

If `renormalise=TRUE`, the estimated intensity `lambda` is multiplied by $c^(normpower/2)$ before performing other calculations, where $c = area(W)/sum[i](1/lambda(x[i]))$. This renormalisation has about the same effect as in `Kinhom`, reducing the variability and bias of the estimate in small samples and in cases of very strong inhomogeneity.

Edge corrections are used to correct bias in the estimation of K_{scaled}. First the interpoint distances are rescaled, and then edge corrections are applied as in `Kest`. See `Kest` for details of the edge corrections and the options for the argument `correction`.

The pair correlation function can also be applied to the result of `Kscaled`; see `pcf` and `pcf.fv`.

Value

An object of class "fv" (see fv.object).

Essentially a data frame containing at least the following columns,

r	the vector of values of the argument r at which the pair correlation function $g(r)$ has been estimated
theo	vector of values of πr^2, the theoretical value of $K_{\mathrm{scaled}}(r)$ for an inhomogeneous Poisson process

and containing additional columns according to the choice specified in the correction argument. The additional columns are named border, trans and iso and give the estimated values of $K_{\mathrm{scaled}}(r)$ using the border correction, translation correction, and Ripley isotropic correction, respectively.

Author(s)

Ute Hahn, Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

References

Baddeley, A., Møller, J. and Waagepetersen, R. (2000) Non- and semiparametric estimation of interaction in inhomogeneous point patterns. *Statistica Neerlandica* **54**, 329–350.

Hahn, U. (2007) *Global and Local Scaling in the Statistics of Spatial Point Processes*. Habilitationsschrift, Universitaet Augsburg.

Hahn, U., Jensen, E.B.V., van Lieshout, M.N.M. and Nielsen, L.S. (2003) Inhomogeneous spatial point processes by location-dependent scaling. *Advances in Applied Probability* **35**, 319–336.

Prokešová, M., Hahn, U. and Vedel Jensen, E.B. (2006) Statistics for locally scaled point patterns. In A. Baddeley, P. Gregori, J. Mateu, R. Stoica and D. Stoyan (eds.) *Case Studies in Spatial Point Pattern Modelling*. Lecture Notes in Statistics 185. New York: Springer Verlag. Pages 99–123.

See Also

Kest, pcf

Examples

```
X <- unmark(bronzefilter)
K <- Kscaled(X)
if(require("spatstat.model")) {
  fit <- ppm(X, ~x)
  lam <- predict(fit)
  K <- Kscaled(X, lam)
}
```

Ksector *Sector K-function*

Description

A directional counterpart of Ripley's K function, in which pairs of points are counted only when the vector joining the pair happens to lie in a particular range of angles.

Usage

```
Ksector(X, begin = 0, end = 360, ...,
        units = c("degrees", "radians"),
        r = NULL, breaks = NULL,
        correction = c("border", "isotropic", "Ripley", "translate"),
        domain=NULL, ratio = FALSE, verbose=TRUE)
```

Arguments

X	The observed point pattern, from which an estimate of $K(r)$ will be computed. An object of class "ppp", or data in any format acceptable to as.ppp().
begin, end	Numeric values giving the range of angles inside which points will be counted. Angles are measured in degrees (if units="degrees", the default) or radians (if units="radians") anti-clockwise from the positive x-axis.
...	Ignored.
units	Units in which the angles begin and end are expressed.
r	Optional. Vector of values for the argument r at which $K(r)$ should be evaluated. Users are advised *not* to specify this argument; there is a sensible default.
breaks	This argument is for internal use only.
correction	Optional. A character vector containing any selection of the options "none", "border", "bord.modif", "isotropic", "Ripley", "translate", "translation", "none", "good" or "best". It specifies the edge correction(s) to be applied. Alternatively correction="all" selects all options.
domain	Optional window. The first point x_i of each pair of points will be constrained to lie in domain.
ratio	Logical. If TRUE, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.
verbose	Logical value indicating whether to print progress reports and warnings.

Details

This is a directional counterpart of Ripley's K function (see Kest) in which, instead of counting all pairs of points within a specified distance r, we count only the pairs (x_i, x_j) for which the vector $x_j - x_i$ falls in a particular range of angles.

This can be used to evaluate evidence for anisotropy in the point pattern X.

Value

An object of class "fv" containing the estimated function.

Author(s)

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

Rolf Turner <rolfturner@posteo.net>

and Ege Rubak <rubak@math.aau.dk>

See Also

Kest

Examples

```
K <- Ksector(swedishpines, 0, 90)
plot(K)
```

---

laslett *Laslett's Transform*

Description

Apply Laslett's Transform to a spatial region, returning the original and transformed regions, and the original and transformed positions of the lower tangent points. This is a diagnostic for the Boolean model.

Usage

```
laslett(X, ..., verbose = FALSE, plotit = TRUE, discretise = FALSE,
        type=c("lower", "upper", "left", "right"))
```

Arguments

X	Spatial region to be transformed. A window (object of class "owin") or a logical-valued pixel image (object of class "im").
...	Graphics arguments to control the plot (passed to plot.laslett when plotit=TRUE) or arguments determining the pixel resolution (passed to as.mask).
verbose	Logical value indicating whether to print progress reports.
plotit	Logical value indicating whether to plot the result.
discretise	Logical value indicating whether polygonal windows should first be converted to pixel masks before the Laslett transform is computed. This should be set to TRUE for very complicated polygons.
type	Type of tangent points to be detected. This also determines the direction of contraction in the set transformation. Default is type="lower".

Details

This function finds the lower tangent points of the spatial region X, then applies Laslett's Transform to the space, and records the transformed positions of the lower tangent points.

Laslett's transform is a diagnostic for the Boolean Model. A test of the Boolean model can be performed by applying a test of CSR to the transformed tangent points. See the Examples.

The rationale is that, if the region X was generated by a Boolean model with convex grains, then the lower tangent points of X, when subjected to Laslett's transform, become a Poisson point process (Cressie, 1993, section 9.3.5; Molchanov, 1997; Barbour and Schmidt, 2001).

Intuitively, Laslett's transform is a way to account for the fact that tangent points of X cannot occur *inside* X. It treats the interior of X as empty space, and collapses this empty space so that only the *exterior* of X remains. In this collapsed space, the tangent points are completely random.

Formally, Laslett's transform is a random (i.e. data-dependent) spatial transformation which maps each spatial location (x, y) to a new location (x', y) at the same height y. The transformation is defined so that x' is the total *uncovered* length of the line segment from $(0, y)$ to (x, y), that is, the total length of the parts of this segment that fall outside the region X.

In more colourful terms, suppose we use an abacus to display a pixellated version of X. Each wire of the abacus represents one horizontal line in the pixel image. Each pixel lying *outside* the region X is represented by a bead of the abacus; pixels *inside* X are represented by the absence of a bead. Next we find any beads which are lower tangent points of X, and paint them green. Then Laslett's Transform is applied by pushing all beads to the left, as far as possible. The final locations of all the beads provide a new spatial region, inside which is the point pattern of tangent points (marked by the green-painted beads).

If `plotit=TRUE` (the default), a before-and-after plot is generated, showing the region X and the tangent points before and after the transformation. This plot can also be generated by calling `plot(a)` where a is the object returned by the function `laslett`.

If the argument `type` is given, then this determines the type of tangents that will be detected, and also the direction of contraction in Laslett's transform. The computation is performed by first rotating X, applying Laslett's transform for lower tangent points, then rotating back.

There are separate algorithms for polygonal windows and pixellated windows (binary masks). The polygonal algorithm may be slow for very complicated polygons. If this happens, setting `discretise=TRUE` will convert the polygonal window to a binary mask and invoke the pixel raster algorithm.

Value

A list, which also belongs to the class `"laslett"` so that it can immediately be printed and plotted.

The list elements are:

oldX: the original dataset X;

TanOld: a point pattern, whose window is `Frame(X)`, containing the lower tangent points of X;

TanNew: a point pattern, whose window is the Laslett transform of `Frame(X)`, and which contains the Laslett-transformed positions of the tangent points;

Rect: a rectangular window, which is the largest rectangle lying inside the transformed set;

df: a data frame giving the locations of the tangent points before and after transformation.

type: character string specifying the type of tangents.

Author(s)

Kassel Hingee and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

References

Barbour, A.D. and Schmidt, V. (2001) On Laslett's Transform for the Boolean Model. *Advances in Applied Probability* **33**(1), 1–5.

Cressie, N.A.C. (1993) *Statistics for spatial data*, second edition. John Wiley and Sons.

Molchanov, I. (1997) *Statistics of the Boolean Model for Practitioners and Mathematicians*. Wiley.

See Also

plot.laslett

Examples

```
a <- laslett(heather$coarse)
transformedHeather <- with(a, Window(TanNew))
plot(transformedHeather, invert=TRUE)

with(a, clarkevans.test(TanNew[Rect], correction="D", nsim=39))

X <- discs(runifrect(15) %mark% 0.2, npoly=16)
b <- laslett(X, type="left")
b
```

Lcross *Multitype L-function (cross-type)*

Description

Calculates an estimate of the cross-type L-function for a multitype point pattern.

Usage

```
Lcross(X, i, j, ..., from, to, correction)
```

Arguments

X	The observed point pattern, from which an estimate of the cross-type L function $L_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of marks(X).

j	The type (mark value) of the points in X to which distances are measured. A character string (or something that will be converted to a character string). Defaults to the second level of marks(X).
correction, ...	Arguments passed to Kcross.
from, to	An alternative way to specify i and j respectively.

Details

The cross-type L-function is a transformation of the cross-type K-function,

$$L_{ij}(r) = \sqrt{\frac{K_{ij}(r)}{\pi}}$$

where $K_{ij}(r)$ is the cross-type K-function from type i to type j. See Kcross for information about the cross-type K-function.

The command Lcross first calls Kcross to compute the estimate of the cross-type K-function, and then applies the square root transformation.

For a marked point pattern in which the points of type i are independent of the points of type j, the theoretical value of the L-function is $L_{ij}(r) = r$. The square root also has the effect of stabilising the variance of the estimator, so that L_{ij} is more appropriate for use in simulation envelopes and hypothesis tests.

Value

An object of class "fv", see fv.object, which can be plotted directly using plot.fv.

Essentially a data frame containing columns

r	the vector of values of the argument r at which the function L_{ij} has been estimated
theo	the theoretical value $L_{ij}(r) = r$ for a stationary Poisson process

together with columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function L_{ij} obtained by the edge corrections named.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

Kcross, Ldot, Lest

Examples

```
L <- Lcross(amacrine, "off", "on")
plot(L)
```

`Lcross.inhom` *Inhomogeneous Cross Type L Function*

Description

For a multitype point pattern, estimate the inhomogeneous version of the cross-type L function.

Usage

```
Lcross.inhom(X, i, j, ..., correction)
```

Arguments

`X`	The observed point pattern, from which an estimate of the inhomogeneous cross type L function $L_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
`i`	The type (mark value) of the points in `X` from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of `marks(X)`.
`j`	The type (mark value) of the points in `X` to which distances are measured. A character string (or something that will be converted to a character string). Defaults to the second level of `marks(X)`.
`correction, ...`	Other arguments passed to `Kcross.inhom`.

Details

This is a generalisation of the function `Lcross` to include an adjustment for spatially inhomogeneous intensity, in a manner similar to the function `Linhom`.

All the arguments are passed to `Kcross.inhom`, which estimates the inhomogeneous multitype K function $K_{ij}(r)$ for the point pattern. The resulting values are then transformed by taking $L(r) = \sqrt{K(r)/\pi}$.

Value

An object of class `"fv"` (see `fv.object`).

Essentially a data frame containing numeric columns

`r`	the values of the argument r at which the function $L_{ij}(r)$ has been estimated
`theo`	the theoretical value of $L_{ij}(r)$ for a marked Poisson process, identically equal to `r`

together with a column or columns named `"border"`, `"bord.modif"`, `"iso"` and/or `"trans"`, according to the selected edge corrections. These columns contain estimates of the function $L_{ij}(r)$ obtained by the edge corrections named.

Warnings

The arguments `i` and `j` are always interpreted as levels of the factor `X$marks`. They are converted to character strings if they are not already character strings. The value `i=1` does **not** refer to the first level of the factor.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

References

Møller, J. and Waagepetersen, R. Statistical Inference and Simulation for Spatial Point Processes Chapman and Hall/CRC Boca Raton, 2003.

See Also

Lcross, Linhom, Kcross.inhom

Examples

```
# Lansing Woods data
woods <- lansing

ma <- split(woods)$maple
wh <- split(woods)$whiteoak

# method (1): estimate intensities by nonparametric smoothing
lambdaM <- density.ppp(ma, sigma=0.15, at="points")
lambdaW <- density.ppp(wh, sigma=0.15, at="points")
L <- Lcross.inhom(woods, "whiteoak", "maple", lambdaW, lambdaM)

# method (2): fit parametric intensity model
if(require("spatstat.model"))  {
  fit <- ppm(woods ~marks * polynom(x,y,2))
  # evaluate fitted intensities at data points
  # (these are the intensities of the sub-processes of each type)
  inten <- fitted(fit, dataonly=TRUE)
  # split according to types of points
  lambda <- split(inten, marks(woods))
  L <- Lcross.inhom(woods, "whiteoak", "maple",
                    lambda$whiteoak, lambda$maple)
}

# synthetic example: type A points have intensity 50,
#                    type B points have intensity 100 * x
lamB <- as.im(function(x,y){50 + 100 * x}, owin())
X <- superimpose(A=runifpoispp(50), B=rpoispp(lamB))
L <- Lcross.inhom(X, "A", "B",
    lambdaI=as.im(50, Window(X)), lambdaJ=lamB)
```

`Ldot` *Multitype L-function (i-to-any)*

Description

Calculates an estimate of the multitype L-function (from type `i` to any type) for a multitype point pattern.

Usage

```
Ldot(X, i, ..., from, correction)
```

Arguments

`X`	The observed point pattern, from which an estimate of the dot-type L function $L_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
`i`	The type (mark value) of the points in `X` from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of `marks(X)`.
`correction, ...`	Arguments passed to `Kdot`.
`from`	An alternative way to specify `i`.

Details

This command computes

$$L_{i\bullet}(r) = \sqrt{\frac{K_{i\bullet}(r)}{\pi}}$$

where $K_{i\bullet}(r)$ is the multitype K-function from points of type `i` to points of any type. See `Kdot` for information about $K_{i\bullet}(r)$.

The command `Ldot` first calls `Kdot` to compute the estimate of the `i`-to-any K-function, and then applies the square root transformation.

For a marked Poisson point process, the theoretical value of the L-function is $L_{i\bullet}(r) = r$. The square root also has the effect of stabilising the variance of the estimator, so that $L_{i\bullet}$ is more appropriate for use in simulation envelopes and hypothesis tests.

Value

An object of class `"fv"`, see `fv.object`, which can be plotted directly using `plot.fv`.

Essentially a data frame containing columns

`r`	the vector of values of the argument r at which the function $L_{i\bullet}$ has been estimated
`theo`	the theoretical value $L_{i\bullet}(r) = r$ for a stationary Poisson process

together with columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $L_{i\bullet}$ obtained by the edge corrections named.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

Kdot, Lcross, Lest

Examples

```
L <- Ldot(amacrine, "off")
plot(L)
```

Ldot.inhom *Inhomogeneous Multitype L Dot Function*

Description

For a multitype point pattern, estimate the inhomogeneous version of the dot L function.

Usage

```
Ldot.inhom(X, i, ..., correction)
```

Arguments

X	The observed point pattern, from which an estimate of the inhomogeneous cross type L function $L_{i\bullet}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of marks(X).
correction, ...	Other arguments passed to Kdot.inhom.

Details

This a generalisation of the function Ldot to include an adjustment for spatially inhomogeneous intensity, in a manner similar to the function Linhom.

All the arguments are passed to Kdot.inhom, which estimates the inhomogeneous multitype K function $K_{i\bullet}(r)$ for the point pattern. The resulting values are then transformed by taking $L(r) = \sqrt{K(r)/\pi}$.

Value

An object of class "fv" (see fv.object).

Essentially a data frame containing numeric columns

r	the values of the argument r at which the function $L_{i\bullet}(r)$ has been estimated
theo	the theoretical value of $L_{i\bullet}(r)$ for a marked Poisson process, identical to r.

together with a column or columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $L_{i\bullet}(r)$ obtained by the edge corrections named.

Warnings

The argument i is interpreted as a level of the factor X$marks. It is converted to a character string if it is not already a character string. The value i=1 does **not** refer to the first level of the factor.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>

References

Møller, J. and Waagepetersen, R. Statistical Inference and Simulation for Spatial Point Processes Chapman and Hall/CRC Boca Raton, 2003.

See Also

Ldot, Linhom, Kdot.inhom, Lcross.inhom.

Examples

```
# Lansing Woods data
lan <- lansing
lan <- lan[seq(1,npoints(lan), by=10)]
ma <- split(lan)$maple
lg <- unmark(lan)

# Estimate intensities by nonparametric smoothing
lambdaM <- density(ma, sigma=0.15, at="points")
lambdadot <- density(lg, sigma=0.15, at="points")
L <- Ldot.inhom(lan, "maple", lambdaI=lambdaM,
                              lambdadot=lambdadot)


# synthetic example: type A points have intensity 50,
#                    type B points have intensity 50 + 100 * x
lamB <- as.im(function(x,y){50 + 100 * x}, owin())
lamdot <- as.im(function(x,y) { 100 + 100 * x}, owin())
X <- superimpose(A=runifpoispp(50), B=rpoispp(lamB))
L <- Ldot.inhom(X, "B",  lambdaI=lamB,     lambdadot=lamdot)
```

`Lest` *L-function*

Description

Calculates an estimate of the L-function (Besag's transformation of Ripley's K-function) for a spatial point pattern.

Usage

```
Lest(X, ..., correction)
```

Arguments

`X`	The observed point pattern, from which an estimate of $L(r)$ will be computed. An object of class `"ppp"`, or data in any format acceptable to `as.ppp()`.
`correction, ...`	Other arguments passed to `Kest` to control the estimation procedure.

Details

This command computes an estimate of the L-function for the spatial point pattern `X`. The L-function is a transformation of Ripley's K-function,

$$L(r) = \sqrt{\frac{K(r)}{\pi}}$$

where $K(r)$ is the K-function.

See `Kest` for information about Ripley's K-function. The transformation to L was proposed by Besag (1977).

The command `Lest` first calls `Kest` to compute the estimate of the K-function, and then applies the square root transformation.

For a completely random (uniform Poisson) point pattern, the theoretical value of the L-function is $L(r) = r$. The square root also has the effect of stabilising the variance of the estimator, so that $L(r)$ is more appropriate for use in simulation envelopes and hypothesis tests.

See `Kest` for the list of arguments.

Value

An object of class `"fv"`, see `fv.object`, which can be plotted directly using `plot.fv`.

Essentially a data frame containing columns

`r`	the vector of values of the argument r at which the function L has been estimated
`theo`	the theoretical value $L(r) = r$ for a stationary Poisson process

together with columns named `"border"`, `"bord.modif"`, `"iso"` and/or `"trans"`, according to the selected edge corrections. These columns contain estimates of the function $L(r)$ obtained by the edge corrections named.

Variance approximations

If the argument `var.approx=TRUE` is given, the return value includes columns `rip` and `ls` containing approximations to the variance of $\hat{L}(r)$ under CSR. These are obtained by the delta method from the variance approximations described in `Kest`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

References

Besag, J. (1977) Discussion of Dr Ripley's paper. *Journal of the Royal Statistical Society, Series B*, **39**, 193–195.

See Also

`Kest`, `pcf`

Examples

```
L <- Lest(cells)
plot(L, main="L function for cells")
```

Linhom *Inhomogeneous L-function*

Description

Calculates an estimate of the inhomogeneous version of the L-function (Besag's transformation of Ripley's K-function) for a spatial point pattern.

Usage

```
Linhom(X, ..., correction)
```

Arguments

`X`	The observed point pattern, from which an estimate of $L(r)$ will be computed. An object of class `"ppp"`, or data in any format acceptable to `as.ppp()`.
`correction, ...`	Other arguments passed to `Kinhom` to control the estimation procedure.

Details

This command computes an estimate of the inhomogeneous version of the L-function for a spatial point pattern.

The original L-function is a transformation (proposed by Besag) of Ripley's K-function,

$$L(r) = \sqrt{\frac{K(r)}{\pi}}$$

where $K(r)$ is the Ripley K-function of a spatially homogeneous point pattern, estimated by `Kest`.

The inhomogeneous L-function is the corresponding transformation of the inhomogeneous K-function, estimated by `Kinhom`. It is appropriate when the point pattern clearly does not have a homogeneous intensity of points. It was proposed by Baddeley, Møller and Waagepetersen (2000).

The command `Linhom` first calls `Kinhom` to compute the estimate of the inhomogeneous K-function, and then applies the square root transformation.

For a Poisson point pattern (homogeneous or inhomogeneous), the theoretical value of the inhomogeneous L-function is $L(r) = r$. The square root also has the effect of stabilising the variance of the estimator, so that L is more appropriate for use in simulation envelopes and hypothesis tests.

Value

An object of class `"fv"`, see `fv.object`, which can be plotted directly using `plot.fv`.

Essentially a data frame containing columns

`r`	the vector of values of the argument r at which the function L has been estimated
`theo`	the theoretical value $L(r) = r$ for a stationary Poisson process

together with columns named `"border"`, `"bord.modif"`, `"iso"` and/or `"trans"`, according to the selected edge corrections. These columns contain estimates of the function $L(r)$ obtained by the edge corrections named.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

References

Baddeley, A., Møller, J. and Waagepetersen, R. (2000) Non- and semiparametric estimation of interaction in inhomogeneous point patterns. *Statistica Neerlandica* **54**, 329–350.

See Also

`Kest`, `Lest`, `Kinhom`, `pcf`

Examples

```
X <- japanesepines
L <- Linhom(X, sigma=0.1)
plot(L, main="Inhomogeneous L function for Japanese Pines")
```

`localK` *Neighbourhood density function*

Description

Computes the neighbourhood density function, a local version of the K-function or L-function, defined by Getis and Franklin (1987).

Usage

```
localK(X, ..., rmax = NULL, correction = "Ripley", verbose = TRUE, rvalue=NULL)
localL(X, ..., rmax = NULL, correction = "Ripley", verbose = TRUE, rvalue=NULL)
```

Arguments

`X`	A point pattern (object of class `"ppp"`).
`...`	Ignored.
`rmax`	Optional. Maximum desired value of the argument r.
`correction`	String specifying the edge correction to be applied. Options are `"none"`, `"translate"`, `"translation"`, `"Ripley"`, `"isotropic"` or `"best"`. Only one correction may be specified.
`verbose`	Logical flag indicating whether to print progress reports during the calculation.
`rvalue`	Optional. A *single* value of the distance argument r at which the function L or K should be computed.

Details

The command `localL` computes the *neighbourhood density function*, a local version of the L-function (Besag's transformation of Ripley's K-function) that was proposed by Getis and Franklin (1987). The command `localK` computes the corresponding local analogue of the K-function.

Given a spatial point pattern `X`, the neighbourhood density function $L_i(r)$ associated with the ith point in `X` is computed by

$$L_i(r) = \sqrt{\frac{a}{(n-1)\pi} \sum_j e_{ij}}$$

where the sum is over all points $j \neq i$ that lie within a distance r of the ith point, a is the area of the observation window, n is the number of points in `X`, and e_{ij} is an edge correction term (as described in `Kest`). The value of $L_i(r)$ can also be interpreted as one of the summands that contributes to the global estimate of the L function.

By default, the function $L_i(r)$ or $K_i(r)$ is computed for a range of r values for each point i. The results are stored as a function value table (object of class `"fv"`) with a column of the table containing the function estimates for each point of the pattern `X`.

Alternatively, if the argument `rvalue` is given, and it is a single number, then the function will only be computed for this value of r, and the results will be returned as a numeric vector, with one entry of the vector for each point of the pattern `X`.

Inhomogeneous counterparts of `localK` and `localL` are computed by `localKinhom` and `localLinhom`.

Value

If `rvalue` is given, the result is a numeric vector of length equal to the number of points in the point pattern.

If `rvalue` is absent, the result is an object of class `"fv"`, see `fv.object`, which can be plotted directly using `plot.fv`. Essentially a data frame containing columns

`r`	the vector of values of the argument r at which the function K has been estimated
`theo`	the theoretical value $K(r) = \pi r^2$ or $L(r) = r$ for a stationary Poisson process

together with columns containing the values of the neighbourhood density function for each point in the pattern. Column `i` corresponds to the `i`th point. The last two columns contain the `r` and `theo` values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

References

Getis, A. and Franklin, J. (1987) Second-order neighbourhood analysis of mapped point patterns. *Ecology* **68**, 473–477.

See Also

`Kest`, `Lest`, `localKinhom`, `localLinhom`.

Examples

```
X <- ponderosa

# compute all the local L functions
L <- localL(X)

# plot all the local L functions against r
plot(L, main="local L functions for ponderosa", legend=FALSE)

# plot only the local L function for point number 7
plot(L, iso007 ~ r)

# compute the values of L(r) for r = 12 metres
L12 <- localL(X, rvalue=12)

# Spatially interpolate the values of L12
# Compare Figure 5(b) of Getis and Franklin (1987)
X12 <- X %mark% L12
Z <- Smooth(X12, sigma=5, dimyx=128)

plot(Z, col=topo.colors(128), main="smoothed neighbourhood density")
contour(Z, add=TRUE)
points(X, pch=16, cex=0.5)
```

`localKcross` *Local Multitype K Function (Cross-Type)*

Description

for a multitype point pattern, computes the cross-type version of the local K function.

Usage

```
localKcross(X, from, to, ..., rmax = NULL,
            correction = "Ripley", verbose = TRUE, rvalue=NULL)
localLcross(X, from, to, ..., rmax = NULL, correction = "Ripley")
```

Arguments

`X`	A multitype point pattern (object of class `"ppp"` with marks which are a factor).
`...`	Further arguments passed from `localLcross` to `localKcross`.
`rmax`	Optional. Maximum desired value of the argument r.
`from`	Type of points from which distances should be measured. A single value; one of the possible levels of `marks(X)`, or an integer indicating which level.
`to`	Type of points to which distances should be measured. A single value; one of the possible levels of `marks(X)`, or an integer indicating which level.
`correction`	String specifying the edge correction to be applied. Options are `"none"`, `"translate"`, `"translation"`, `"Ripley"`, `"isotropic"` or `"best"`. Only one correction may be specified.
`verbose`	Logical flag indicating whether to print progress reports during the calculation.
`rvalue`	Optional. A *single* value of the distance argument r at which the function L or K should be computed.

Details

Given a multitype spatial point pattern `X`, the local cross-type K function `localKcross` is the local version of the multitype K function `Kcross`. Recall that `Kcross(X, from, to)` is a sum of contributions from all pairs of points in `X` where the first point belongs to `from` and the second point belongs to type `to`. The *local* cross-type K function is defined for each point `X[i]` that belongs to type `from`, and it consists of all the contributions to the cross-type K function that originate from point `X[i]`:

$$K_{i,from,to}(r) = \sqrt{\frac{a}{(n-1)\pi} \sum_j e_{ij}}$$

where the sum is over all points $j \neq i$ belonging to type `to`, that lie within a distance r of the ith point, a is the area of the observation window, n is the number of points in `X`, and e_{ij} is an edge correction term (as described in `Kest`). The value of $K_{i,from,to}(r)$ can also be interpreted as one of the summands that contributes to the global estimate of the `Kcross` function.

By default, the function $K_{i,from,to}(r)$ is computed for a range of r values for each point i belonging to type `from`. The results are stored as a function value table (object of class `"fv"`) with a column of the table containing the function estimates for each point of the pattern `X` belonging to type `from`.

Alternatively, if the argument `rvalue` is given, and it is a single number, then the function will only be computed for this value of r, and the results will be returned as a numeric vector, with one entry of the vector for each point of the pattern `X` belonging to type `from`.

The local cross-type L function `localLcross` is computed by applying the transformation $L(r) = \sqrt{K(r)/(2\pi)}$.

Value

If `rvalue` is given, the result is a numeric vector of length equal to the number of points in the point pattern that belong to type `from`.

If `rvalue` is absent, the result is an object of class `"fv"`, see `fv.object`, which can be plotted directly using `plot.fv`. Essentially a data frame containing columns

`r`	the vector of values of the argument r at which the function K has been estimated
`theo`	the theoretical value $K(r) = \pi r^2$ or $L(r) = r$ for a stationary Poisson process

together with columns containing the values of the neighbourhood density function for each point in the pattern. Column `i` corresponds to the `i`th point of type `from`. The last two columns contain the `r` and `theo` values.

Author(s)

Ege Rubak <rubak@math.aau.dk> and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

See Also

`Kcross`, `Lcross`, `localK`, `localL`.

Inhomogeneous counterparts of `localK` and `localL` are computed by `localKcross.inhom` and `localLinhom`.

Examples

```
X <- amacrine

# compute all the local Lcross functions
L <- localLcross(X)

# plot all the local Lcross functions against r
plot(L, main="local Lcross functions for amacrine", legend=FALSE)

# plot only the local L function for point number 7
plot(L, iso007 ~ r)

# compute the values of L(r) for r = 0.1 metres
L12 <- localLcross(X, rvalue=0.1)
```

`localKcross.inhom` *Inhomogeneous Multitype K Function*

Description

Computes spatially-weighted versions of the the local multitype K-function or L-function.

Usage

```
localKcross.inhom(X, from, to,
            lambdaFrom=NULL, lambdaTo=NULL,
            ..., rmax = NULL,
            correction = "Ripley", sigma=NULL, varcov=NULL,
            lambdaX=NULL, update=TRUE, leaveoneout=TRUE)
localLcross.inhom(X, from, to,
            lambdaFrom=NULL, lambdaTo=NULL, ..., rmax = NULL)
```

Arguments

`X`	A point pattern (object of class `"ppp"`).
`from`	Type of points from which distances should be measured. A single value; one of the possible levels of `marks(X)`, or an integer indicating which level.
`to`	Type of points to which distances should be measured. A single value; one of the possible levels of `marks(X)`, or an integer indicating which level.
`lambdaFrom`, `lambdaTo`	Optional. Values of the estimated intensity function for the points of type `from` and `to`, respectively. Each argument should be either a vector giving the intensity values at the required points, a pixel image (object of class `"im"`) giving the intensity values at all locations, a fitted point process model (object of class `"ppm"`) or a `function(x,y)` which can be evaluated to give the intensity value at any location.
`...`	Extra arguments. Ignored if `lambda` is present. Passed to `density.ppp` if `lambda` is omitted.
`rmax`	Optional. Maximum desired value of the argument r.
`correction`	String specifying the edge correction to be applied. Options are `"none"`, `"translate"`, `"Ripley"`, `"translation"`, `"isotropic"` or `"best"`. Only one correction may be specified.
`sigma`, `varcov`	Optional arguments passed to `density.ppp` to control the kernel smoothing procedure for estimating `lambdaFrom` and `lambdaTo`, if they are missing.
`lambdaX`	Optional. Values of the estimated intensity function for all points of `X`. Either a vector giving the intensity values at each point of `X`, a pixel image (object of class `"im"`) giving the intensity values at all locations, a list of pixel images giving the intensity values at all locations for each type of point, or a fitted point process model (object of class `"ppm"`) or a `function(x,y)` or `function(x,y,m)` which can be evaluated to give the intensity value at any location.

update — Logical value indicating what to do when lambdaFrom, lambdaTo or lambdaX is a fitted model (class "ppm", "kppm" or "dppm"). If update=TRUE (the default), the model will first be refitted to the data X (using update.ppm or update.kppm) before the fitted intensity is computed. If update=FALSE, the fitted intensity of the model will be computed without re-fitting it to X.

leaveoneout — Logical value (passed to density.ppp or fitted.ppm) specifying whether to use a leave-one-out rule when calculating the intensity.

Details

The functions localKcross.inhom and localLcross.inhom are inhomogeneous or weighted versions of the local multitype K and L functions implemented in localKcross and localLcross.

Given a multitype spatial point pattern X, and two designated types from and to, the local multitype K function is defined for each point X[i] that belongs to type from, and is computed by

$$K_i(r) = \sqrt{\frac{1}{\pi} \sum_j \frac{e_{ij}}{\lambda_j}}$$

where the sum is over all points $j \neq i$ of type to that lie within a distance r of the ith point, λ_j is the estimated intensity of the point pattern at the point j, and e_{ij} is an edge correction term (as described in Kest).

The function $K_i(r)$ is computed for a range of r values for each point i. The results are stored as a function value table (object of class "fv") with a column of the table containing the function estimates for each point of the pattern X of type from.

The corresponding L function $L_i(r)$ is computed by applying the transformation $L(r) = \sqrt{K(r)/(2\pi)}$.

Value

An object of class "fv", see fv.object, which can be plotted directly using plot.fv. Essentially a data frame containing columns

r — the vector of values of the argument r at which the function K has been estimated

theo — the theoretical value $K(r) = \pi r^2$ or $L(r) = r$ for a stationary Poisson process

together with columns containing the values of the neighbourhood density function for each point in the pattern of type from. The last two columns contain the r and theo values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

Kinhom, Linhom, localK, localL.

Examples

```
X <- amacrine

# compute all the local L functions
L <- localLcross.inhom(X)

# plot all the local L functions against r
plot(L, main="local L functions for ponderosa", legend=FALSE)

# plot only the local L function for point number 7
plot(L, iso007 ~ r)
```

`localKdot` *Local Multitype K Function (Dot-Type)*

Description

for a multitype point pattern, computes the dot-type version of the local K function.

Usage

```
localKdot(X, from, ..., rmax = NULL,
            correction = "Ripley", verbose = TRUE, rvalue=NULL)
localLdot(X, from, ..., rmax = NULL, correction = "Ripley")
```

Arguments

`X`	A multitype point pattern (object of class `"ppp"` with marks which are a factor).
`...`	Further arguments passed from `localLdot` to `localKdot`.
`rmax`	Optional. Maximum desired value of the argument r.
`from`	Type of points from which distances should be measured. A single value; one of the possible levels of `marks(X)`, or an integer indicating which level.
`correction`	String specifying the edge correction to be applied. Options are `"none"`, `"translate"`, `"translation"`, `"Ripley"`, `"isotropic"` or `"best"`. Only one correction may be specified.
`verbose`	Logical flag indicating whether to print progress reports during the calculation.
`rvalue`	Optional. A *single* value of the distance argument r at which the function L or K should be computed.

Details

Given a multitype spatial point pattern `X`, the local dot-type K function `localKdot` is the local version of the multitype K function `Kdot`. Recall that `Kdot(X, from)` is a sum of contributions from all pairs of points in `X` where the first point belongs to `from`. The *local* dot-type K function is

defined for each point X[i] that belongs to type from, and it consists of all the contributions to the dot-type K function that originate from point X[i]:

$$K_{i,from,to}(r) = \sqrt{\frac{a}{(n-1)\pi} \sum_j e_{ij}}$$

where the sum is over all points $j \neq i$ that lie within a distance r of the ith point, a is the area of the observation window, n is the number of points in X, and e_{ij} is an edge correction term (as described in Kest). The value of $K_{i,from}(r)$ can also be interpreted as one of the summands that contributes to the global estimate of the Kdot function.

By default, the function $K_{i,from}(r)$ is computed for a range of r values for each point i belonging to type from. The results are stored as a function value table (object of class "fv") with a column of the table containing the function estimates for each point of the pattern X belonging to type from.

Alternatively, if the argument rvalue is given, and it is a single number, then the function will only be computed for this value of r, and the results will be returned as a numeric vector, with one entry of the vector for each point of the pattern X belonging to type from.

The local dot-type L function localLdot is computed by applying the transformation $L(r) = \sqrt{K(r)/(2\pi)}$.

Value

If rvalue is given, the result is a numeric vector of length equal to the number of points in the point pattern that belong to type from.

If rvalue is absent, the result is an object of class "fv", see fv.object, which can be plotted directly using plot.fv. Essentially a data frame containing columns

r	the vector of values of the argument r at which the function K has been estimated
theo	the theoretical value $K(r) = \pi r^2$ or $L(r) = r$ for a stationary Poisson process

together with columns containing the values of the neighbourhood density function for each point in the pattern. Column i corresponds to the ith point of type from. The last two columns contain the r and theo values.

Author(s)

Ege Rubak <rubak@math.aau.dk> and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

See Also

Kdot, Ldot, localK, localL.

Examples

```
X <- amacrine

# compute all the local Ldot functions
L <- localLdot(X)
```

```
# plot all the local Ldot functions against r
plot(L, main="local Ldot functions for amacrine", legend=FALSE)

# plot only the local L function for point number 7
plot(L, iso007 ~ r)

# compute the values of L(r) for r = 0.1 metres
L12 <- localLdot(X, rvalue=0.1)
```

`localKinhom` *Inhomogeneous Neighbourhood Density Function*

Description

Computes spatially-weighted versions of the the local K-function or L-function.

Usage

```
localKinhom(X, lambda, ..., rmax = NULL,
            correction = "Ripley", verbose = TRUE, rvalue=NULL,
            sigma = NULL, varcov = NULL, update=TRUE, leaveoneout=TRUE)
localLinhom(X, lambda, ..., rmax = NULL,
            correction = "Ripley", verbose = TRUE, rvalue=NULL,
            sigma = NULL, varcov = NULL, update=TRUE, leaveoneout=TRUE)
```

Arguments

`X`	A point pattern (object of class `"ppp"`).
`lambda`	Optional. Values of the estimated intensity function. Either a vector giving the intensity values at the points of the pattern `X`, a pixel image (object of class `"im"`) giving the intensity values at all locations, a fitted point process model (object of class `"ppm"` or `"kppm"` or `"dppm"`) or a `function(x,y)` which can be evaluated to give the intensity value at any location.
`...`	Extra arguments. Ignored if `lambda` is present. Passed to `density.ppp` if `lambda` is omitted.
`rmax`	Optional. Maximum desired value of the argument r.
`correction`	String specifying the edge correction to be applied. Options are `"none"`, `"translate"`, `"Ripley"`, `"translation"`, `"isotropic"` or `"best"`. Only one correction may be specified.
`verbose`	Logical flag indicating whether to print progress reports during the calculation.
`rvalue`	Optional. A *single* value of the distance argument r at which the function L or K should be computed.
`sigma, varcov`	Optional arguments passed to `density.ppp` to control the kernel smoothing procedure for estimating `lambda`, if `lambda` is missing.
`leaveoneout`	Logical value (passed to `density.ppp` or `fitted.ppm`) specifying whether to use a leave-one-out rule when calculating the intensity.

update — Logical value indicating what to do when lambda is a fitted model (class "ppm", "kppm" or "dppm"). If update=TRUE (the default), the model will first be refitted to the data X (using update.ppm or update.kppm) before the fitted intensity is computed. If update=FALSE, the fitted intensity of the model will be computed without re-fitting it to X.

Details

The functions localKinhom and localLinhom are inhomogeneous or weighted versions of the neighbourhood density function implemented in localK and localL.

Given a spatial point pattern X, the inhomogeneous neighbourhood density function $L_i(r)$ associated with the ith point in X is computed by

$$L_i(r) = \sqrt{\frac{1}{\pi} \sum_j \frac{e_{ij}}{\lambda_j}}$$

where the sum is over all points $j \neq i$ that lie within a distance r of the ith point, λ_j is the estimated intensity of the point pattern at the point j, and e_{ij} is an edge correction term (as described in Kest). The value of $L_i(r)$ can also be interpreted as one of the summands that contributes to the global estimate of the inhomogeneous L function (see Linhom).

By default, the function $L_i(r)$ or $K_i(r)$ is computed for a range of r values for each point i. The results are stored as a function value table (object of class "fv") with a column of the table containing the function estimates for each point of the pattern X.

Alternatively, if the argument rvalue is given, and it is a single number, then the function will only be computed for this value of r, and the results will be returned as a numeric vector, with one entry of the vector for each point of the pattern X.

Value

If rvalue is given, the result is a numeric vector of length equal to the number of points in the point pattern.

If rvalue is absent, the result is an object of class "fv", see fv.object, which can be plotted directly using plot.fv. Essentially a data frame containing columns

r — the vector of values of the argument r at which the function K has been estimated

theo — the theoretical value $K(r) = \pi r^2$ or $L(r) = r$ for a stationary Poisson process

together with columns containing the values of the neighbourhood density function for each point in the pattern. Column i corresponds to the ith point. The last two columns contain the r and theo values.

Author(s)

Mike Kuhn, Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

Kinhom, Linhom, localK, localL.

Examples

```
X <- ponderosa

# compute all the local L functions
L <- localLinhom(X)

# plot all the local L functions against r
plot(L, main="local L functions for ponderosa", legend=FALSE)

# plot only the local L function for point number 7
plot(L, iso007 ~ r)

# compute the values of L(r) for r = 12 metres
L12 <- localL(X, rvalue=12)
```

`localpcf` *Local pair correlation function*

Description

Computes individual contributions to the pair correlation function from each data point.

Usage

```
localpcf(X, ..., delta=NULL, rmax=NULL, nr=512, stoyan=0.15, rvalue=NULL)

localpcfinhom(X, ..., delta=NULL, rmax=NULL, nr=512, stoyan=0.15,
       lambda=NULL, sigma=NULL, varcov=NULL,
       update=TRUE, leaveoneout=TRUE, rvalue=NULL)
```

Arguments

`X`	A point pattern (object of class `"ppp"`).
`delta`	Smoothing bandwidth for pair correlation. The halfwidth of the Epanechnikov kernel.
`rmax`	Optional. Maximum value of distance r for which pair correlation values $g(r)$ should be computed.
`nr`	Optional. Number of values of distance r for which pair correlation $g(r)$ should be computed.
`stoyan`	Optional. The value of the constant c in Stoyan's rule of thumb for selecting the smoothing bandwidth `delta`.
`lambda`	Optional. Values of the estimated intensity function, for the inhomogeneous pair correlation. Either a vector giving the intensity values at the points of the pattern `X`, a pixel image (object of class `"im"`) giving the intensity values at all locations, a fitted point process model (object of class `"ppm"`, `"kppm"` or `"dppm"`) or a `function(x,y)` which can be evaluated to give the intensity value at any location.

sigma, varcov, ...
: These arguments are ignored by localpcf but are passed by localpcfinhom (when lambda=NULL) to the function density.ppp to control the kernel smoothing estimation of lambda.

leaveoneout
: Logical value (passed to density.ppp or fitted.ppm) specifying whether to use a leave-one-out rule when calculating the intensity.

update
: Logical value indicating what to do when lambda is a fitted model (class "ppm", "kppm" or "dppm"). If update=TRUE (the default), the model will first be refitted to the data X (using update.ppm or update.kppm) before the fitted intensity is computed. If update=FALSE, the fitted intensity of the model will be computed without re-fitting it to X.

rvalue
: Optional. A *single* value of the distance argument r at which the local pair correlation should be computed.

Details

localpcf computes the contribution, from each individual data point in a point pattern X, to the empirical pair correlation function of X. These contributions are sometimes known as LISA (local indicator of spatial association) functions based on pair correlation.

localpcfinhom computes the corresponding contribution to the *inhomogeneous* empirical pair correlation function of X.

Given a spatial point pattern X, the local pcf $g_i(r)$ associated with the ith point in X is computed by

$$g_i(r) = \frac{a}{2\pi n} \sum_j k(d_{i,j} - r)$$

where the sum is over all points $j \neq i$, a is the area of the observation window, n is the number of points in X, and d_{ij} is the distance between points i and j. Here k is the Epanechnikov kernel,

$$k(t) = \frac{3}{4\delta} \max(0, 1 - \frac{t^2}{\delta^2}).$$

Edge correction is performed using the border method (for the sake of computational efficiency): the estimate $g_i(r)$ is set to NA if $r > b_i$, where b_i is the distance from point i to the boundary of the observation window.

The smoothing bandwidth δ may be specified. If not, it is chosen by Stoyan's rule of thumb $\delta = c/\hat{\lambda}$ where $\hat{\lambda} = n/a$ is the estimated intensity and c is a constant, usually taken to be 0.15. The value of c is controlled by the argument stoyan.

For localpcfinhom, the optional argument lambda specifies the values of the estimated intensity function. If lambda is given, it should be either a numeric vector giving the intensity values at the points of the pattern X, a pixel image (object of class "im") giving the intensity values at all locations, a fitted point process model (object of class "ppm", "kppm" or "dppm") or a function(x,y) which can be evaluated to give the intensity value at any location. If lambda is not given, then it will be estimated using a leave-one-out kernel density smoother as described in pcfinhom.

Alternatively, if the argument rvalue is given, and it is a single number, then the function will only be computed for this value of r, and the results will be returned as a numeric vector, with one entry of the vector for each point of the pattern X.

Value

An object of class "fv", see fv.object, which can be plotted directly using plot.fv. Essentially a data frame containing columns

r	the vector of values of the argument r at which the function K has been estimated
theo	the theoretical value $K(r) = \pi r^2$ or $L(r) = r$ for a stationary Poisson process

together with columns containing the values of the local pair correlation function for each point in the pattern. Column i corresponds to the ith point. The last two columns contain the r and theo values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

localK, localKinhom, pcf, pcfinhom

Examples

```
X <- ponderosa

g <- localpcf(X, stoyan=0.5)
colo <- c(rep("grey", npoints(X)), "blue")
a <- plot(g, main=c("local pair correlation functions", "Ponderosa pines"),
        legend=FALSE, col=colo, lty=1)

# plot only the local pair correlation function for point number 7
plot(g, est007 ~ r)

# Extract the local pair correlation at distance 15 metres, for each point
g15 <- localpcf(X, rvalue=15, stoyan=0.5)
g15[1:10]
# Check that the value for point 7 agrees with the curve for point 7:
points(15, g15[7], col="red")

# Inhomogeneous
gi <- localpcfinhom(X, stoyan=0.5)
a <- plot(gi, main=c("inhomogeneous local pair correlation functions",
                     "Ponderosa pines"),
              legend=FALSE, col=colo, lty=1)
```

lohboot *Bootstrap Confidence Bands for Summary Function*

Description

Computes a bootstrap confidence band for a summary function of a point process.

Usage

```
lohboot(X,
        fun=c("pcf", "Kest", "Lest", "pcfinhom", "Kinhom", "Linhom",
              "Kcross", "Lcross", "Kdot", "Ldot",
              "Kcross.inhom", "Lcross.inhom"),
        ...,
         block=FALSE, global=FALSE, basicboot=FALSE, Vcorrection=FALSE,
         confidence=0.95, nx = 4, ny = nx, nsim=200, type=7)
```

Arguments

X	A point pattern (object of class "ppp").
fun	Name of the summary function for which confidence intervals are desired: one of the strings "pcf", "Kest", "Lest", "pcfinhom", "Kinhom" "Linhom", "Kcross", "Lcross", "Kdot", "Ldot", "Kcross.inhom" or "Lcross.inhom". Alternatively, the function itself; it must be one of the functions listed here.
...	Arguments passed to the corresponding local version of the summary function (see Details).
block	Logical value indicating whether to use Loh's block bootstrap as originally proposed. Default is FALSE for consistency with older code. See Details.
global	Logical. If FALSE (the default), pointwise confidence intervals are constructed. If TRUE, a global (simultaneous) confidence band is constructed.
basicboot	Logical value indicating whether to use the so-called basic bootstrap confidence interval. See Details.
Vcorrection	Logical value indicating whether to use a variance correction when fun="Kest" or fun="Kinhom". See Details.
confidence	Confidence level, as a fraction between 0 and 1.
nx, ny	Integers. If block=TRUE, divide the window into nx*ny rectangles.
nsim	Number of bootstrap simulations.
type	Integer. Type of quantiles. Argument passed to quantile.default controlling the way the quantiles are calculated.

Details

This algorithm computes confidence bands for the true value of the summary function `fun` using the bootstrap method of Loh (2008) and a modification described in Baddeley, Rubak, Turner (2015).

If `fun="pcf"`, for example, the algorithm computes a pointwise `(100 * confidence)`% confidence interval for the true value of the pair correlation function for the point process, normally estimated by `pcf`. It starts by computing the array of *local* pair correlation functions, `localpcf`, of the data pattern `X`. This array consists of the contributions to the estimate of the pair correlation function from each data point.

If `block=FALSE`, these contributions are resampled `nsim` times with replacement as described in Baddeley, Rubak, Turner (2015); from each resampled dataset the total contribution is computed, yielding `nsim` random pair correlation functions.

If `block=TRUE`, the calculation is performed as originally proposed by Loh (2008, 2010). The (bounding box of the) window is divided into $nx * ny$ rectangles (blocks). The average contribution of a block is obtained by averaging the contribution of each point included in the block. Then, the average contributions on each block are resampled `nsim` times with replacement as described in Loh (2008) and Loh (2010); from each resampled dataset the total contribution is computed, yielding `nsim` random pair correlation functions. Notice that for non-rectangular windows any blocks not fully contained in the window are discarded before doing the resampling, so the effective number of blocks may be substantially smaller than $nx * ny$ in this case.

The pointwise `alpha/2` and `1 - alpha/2` quantiles of these functions are computed, where `alpha = 1 - confidence`. The average of the local functions is also computed as an estimate of the pair correlation function.

There are several ways to define a bootstrap confidence interval. If `basicbootstrap=TRUE`, the so-called basic confidence bootstrap interval is used as described in Loh (2008).

It has been noticed in Loh (2010) that when the intensity of the point process is unknown, the bootstrap error estimate is larger than it should be. When the K function is used, an adjustment procedure has been proposed in Loh (2010) that is used if `Vcorrection=TRUE`. In this case, the basic confidence bootstrap interval is implicitly used.

To control the estimation algorithm, use the arguments `...`, which are passed to the local version of the summary function, as shown below:

fun	local version
pcf	localpcf
Kest	localK
Lest	localL
pcfinhom	localpcfinhom
Kinhom	localKinhom
Linhom	localLinhom
Kcross	localKcross
Lcross	localLcross
Kdot	localKdot
Ldot	localLdot
Kcross.inhom	localKcross.inhom
Lcross.inhom	localLcross.inhom

For `fun="Lest"`, the calculations are first performed as if `fun="Kest"`, and then the square-root

transformation is applied to obtain the L-function. Similarly for `fun="Linhom"`, `"Lcross"`, `"Ldot"`, `"Lcross.inhom"`.

Note that the confidence bands computed by `lohboot(fun="pcf")` may not contain the estimate of the pair correlation function computed by `pcf`, because of differences between the algorithm parameters (such as the choice of edge correction) in `localpcf` and `pcf`. If you are using `lohboot`, the appropriate point estimate of the pair correlation itself is the pointwise mean of the local estimates, which is provided in the result of `lohboot` and is shown in the default plot.

If the confidence bands seem unbelievably narrow, this may occur because the point pattern has a hard core (the true pair correlation function is zero for certain values of distance) or because of an optical illusion when the function is steeply sloping (remember the width of the confidence bands should be measured *vertically*).

An alternative to `lohboot` is `varblock`.

Value

A function value table (object of class `"fv"`) containing columns giving the estimate of the summary function, the upper and lower limits of the bootstrap confidence interval, and the theoretical value of the summary function for a Poisson process.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk> and Christophe Biscio.

References

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. Chapman and Hall/CRC Press.

Loh, J.M. (2008) A valid and fast spatial bootstrap for correlation functions. *The Astrophysical Journal*, **681**, 726–734.

Loh, J.M. (2010) Bootstrapping an inhomogeneous point process. *Journal of Statistical Planning and Inference*, **140**, 734–749.

See Also

Summary functions `Kest`, `pcf`, `Kinhom`, `pcfinhom`, `localK`, `localpcf`, `localKinhom`, `localpcfinhom`, `localKcross`, `localKdot`, `localLcross`, `localLdot`. `localKcross.inhom`, `localLcross.inhom`.

See `varblock` for an alternative bootstrap technique.

Examples

```
p <- lohboot(simdat, stoyan=0.5)
g <- lohboot(simdat, stoyan=0.5, block=TRUE)
g
plot(g)
```

`markconnect` *Mark Connection Function*

Description

Estimate the marked connection function of a multitype point pattern.

Usage

```
markconnect(X, i, j, r=NULL,
         correction=c("isotropic", "Ripley", "translate"),
         method="density", ..., normalise=FALSE)
```

Arguments

`X`	The observed point pattern. An object of class `"ppp"` or something acceptable to `as.ppp`.
`i`	Number or character string identifying the type (mark value) of the points in `X` from which distances are measured.
`j`	Number or character string identifying the type (mark value) of the points in `X` to which distances are measured.
`r`	numeric vector. The values of the argument r at which the mark connection function $p_{ij}(r)$ should be evaluated. There is a sensible default.
`correction`	A character vector containing any selection of the options `"isotropic"`, `"Ripley"` or `"translate"`. It specifies the edge correction(s) to be applied.
`method`	A character vector indicating the user's choice of density estimation technique to be used. Options are `"density"`, `"loess"`, `"sm"` and `"smrep"`.
`...`	Arguments passed to `markcorr`, or passed to the density estimation routine (`density`, `loess` or `sm.density`) selected by `method`.
`normalise`	If `TRUE`, normalise the pair connection function by dividing it by $p_i p_j$, the estimated probability that randomly-selected points will have marks i and j.

Details

The mark connection function $p_{ij}(r)$ of a multitype point process X is a measure of the dependence between the types of two points of the process a distance r apart.

Informally $p_{ij}(r)$ is defined as the conditional probability, given that there is a point of the process at a location u and another point of the process at a location v separated by a distance $||u - v|| = r$, that the first point is of type i and the second point is of type j. See Stoyan and Stoyan (1994).

If the marks attached to the points of `X` are independent and identically distributed, then $p_{ij}(r) \equiv p_i p_j$ where p_i denotes the probability that a point is of type i. Values larger than this, $p_{ij}(r) > p_i p_j$, indicate positive association between the two types, while smaller values indicate negative association.

The argument `X` must be a point pattern (object of class `"ppp"`) or any data that are acceptable to `as.ppp`. It must be a multitype point pattern (a marked point pattern with factor-valued marks).

The argument r is the vector of values for the distance r at which $p_{ij}(r)$ is estimated. There is a sensible default.

This algorithm assumes that X can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in X as Window(X)) may have arbitrary shape.

Biases due to edge effects are treated in the same manner as in Kest. The edge corrections implemented here are

isotropic/Ripley Ripley's isotropic correction (see Ripley, 1988; Ohser, 1983). This is implemented only for rectangular and polygonal windows (not for binary masks) and is slow for complicated polygons.

translate Translation correction (Ohser, 1983). Implemented for all window geometries.

none No edge correction.

The option correction="none" should only be used if the number of data points is extremely large (otherwise an edge correction is needed to correct bias).

Note that the estimator assumes the process is stationary (spatially homogeneous).

The mark connection function is estimated using density estimation techniques. The user can choose between

"density" which uses the standard kernel density estimation routine density, and works only for evenly-spaced r values;

"loess" which uses the function loess in the package **modreg**;

"sm" which uses the function sm.density in the package **sm** and is extremely slow;

"smrep" which uses the function sm.density in the package **sm** and is relatively fast, but may require manual control of the smoothing parameter hmult.

Value

An object of class "fv" (see fv.object).

Essentially a data frame containing numeric columns

r	the values of the argument r at which the mark connection function $p_{ij}(r)$ has been estimated
theo	the theoretical value of $p_{ij}(r)$ when the marks attached to different points are independent

together with a column or columns named "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $p_{ij}(r)$ obtained by the edge corrections named.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

References

Stoyan, D. and Stoyan, H. (1994) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

See Also

Multitype pair correlation pcfcross and multitype K-functions Kcross, Kdot.

Use alltypes to compute the mark connection functions between all pairs of types.

Mark correlation markcorr and mark variogram markvario for numeric-valued marks.

Examples

```
# Hughes' amacrine data
# Cells marked as 'on'/'off'
M <- markconnect(amacrine, "on", "off")
plot(M)

# Compute for all pairs of types at once
plot(alltypes(amacrine, markconnect))
```

---

markcorr *Mark Correlation Function*

Description

Estimate the marked correlation function of a marked point pattern.

Usage

```
markcorr(X, f = function(m1, m2) { m1 * m2}, r=NULL,
         correction=c("isotropic", "Ripley", "translate"),
         method="density", ..., weights=NULL,
         f1=NULL, normalise=TRUE, fargs=NULL, internal=NULL)
```

Arguments

X	The observed point pattern. An object of class "ppp" or something acceptable to as.ppp.
f	Optional. Test function f used in the definition of the mark correlation function. An R function with at least two arguments. There is a sensible default.
r	Optional. Numeric vector. The values of the argument r at which the mark correlation function $k_f(r)$ should be evaluated. There is a sensible default.
correction	A character vector containing any selection of the options "isotropic", "Ripley", "translate", "translation", "none" or "best". It specifies the edge correction(s) to be applied. Alternatively correction="all" selects all options.
method	A character vector indicating the user's choice of density estimation technique to be used. Options are "density", "loess", "sm" and "smrep".
...	Arguments passed to the density estimation routine (density, loess or sm.density) selected by method.

weights	Optional. Numeric weights for each data point in X. A numeric vector, a pixel image, or a function(x,y). Alternatively, an expression to be evaluated to yield the weights; the expression may involve the variables x,y,marks representing the coordinates and marks ofX.
f1	An alternative to f. If this argument is given, then f is assumed to take the form $f(u,v) = f_1(u)f_1(v)$.
normalise	If normalise=FALSE, compute only the numerator of the expression for the mark correlation.
fargs	Optional. A list of extra arguments to be passed to the function f or f1.
internal	Do not use this argument.

Details

By default, this command calculates an estimate of Stoyan's mark correlation $k_{mm}(r)$ for the point pattern.

Alternatively if the argument f or f1 is given, then it calculates Stoyan's generalised mark correlation $k_f(r)$ with test function f.

Theoretical definitions are as follows (see Stoyan and Stoyan (1994, p. 262)):

- For a point process X with numeric marks, Stoyan's mark correlation function $k_{mm}(r)$, is

$$k_{mm}(r) = \frac{E_{0u}[M(0)M(u)]}{E[M, M']}$$

  where E_{0u} denotes the conditional expectation given that there are points of the process at the locations 0 and u separated by a distance r, and where $M(0), M(u)$ denote the marks attached to these two points. On the denominator, M, M' are random marks drawn independently from the marginal distribution of marks, and E is the usual expectation.

- For a multitype point process X, the mark correlation is

$$k_{mm}(r) = \frac{P_{0u}[M(0)M(u)]}{P[M = M']}$$

  where P and P_{0u} denote the probability and conditional probability.

- The *generalised* mark correlation function $k_f(r)$ of a marked point process X, with test function f, is

$$k_f(r) = \frac{E_{0u}[f(M(0), M(u))]}{E[f(M, M')]}$$

The test function f is any function $f(m_1, m_2)$ with two arguments which are possible marks of the pattern, and which returns a nonnegative real value. Common choices of f are: for continuous nonnegative real-valued marks,

$$f(m_1, m_2) = m_1 m_2$$

for discrete marks (multitype point patterns),

$$f(m_1, m_2) = 1(m_1 = m_2)$$

and for marks taking values in $[0, 2\pi)$,

$$f(m_1, m_2) = \sin(m_1 - m_2)$$

.

Note that $k_f(r)$ is not a "correlation" in the usual statistical sense. It can take any nonnegative real value. The value 1 suggests "lack of correlation": if the marks attached to the points of `X` are independent and identically distributed, then $k_f(r) \equiv 1$. The interpretation of values larger or smaller than 1 depends on the choice of function f.

The argument `X` must be a point pattern (object of class `"ppp"`) or any data that are acceptable to `as.ppp`. It must be a marked point pattern.

The argument `f` determines the function to be applied to pairs of marks. It has a sensible default, which depends on the kind of marks in `X`. If the marks are numeric values, then `f <- function(m1, m2) { m1 * m2}` computes the product of two marks. If the marks are a factor (i.e. if `X` is a multitype point pattern) then `f <- function(m1, m2) { m1 == m2}` yields the value 1 when the two marks are equal, and 0 when they are unequal. These are the conventional definitions for numerical marks and multitype points respectively.

The argument `f` may be specified by the user. It must be an R function, accepting two arguments `m1` and `m2` which are vectors of equal length containing mark values (of the same type as the marks of `X`). (It may also take additional arguments, passed through `fargs`). It must return a vector of numeric values of the same length as `m1` and `m2`. The values must be non-negative, and `NA` values are not permitted.

Alternatively the user may specify the argument `f1` instead of `f`. This indicates that the test function f should take the form $f(u, v) = f_1(u) f_1(v)$ where $f_1(u)$ is given by the argument `f1`. The argument `f1` should be an R function with at least one argument. (It may also take additional arguments, passed through `fargs`).

The argument `r` is the vector of values for the distance r at which $k_f(r)$ is estimated.

This algorithm assumes that `X` can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in `X` as `Window(X)`) may have arbitrary shape.

Biases due to edge effects are treated in the same manner as in `Kest`. The edge corrections implemented here are

isotropic/Ripley Ripley's isotropic correction (see Ripley, 1988; Ohser, 1983). This is implemented only for rectangular and polygonal windows (not for binary masks).

translate Translation correction (Ohser, 1983). Implemented for all window geometries, but slow for complex windows.

Note that the estimator assumes the process is stationary (spatially homogeneous).

The numerator and denominator of the mark correlation function (in the expression above) are estimated using density estimation techniques. The user can choose between

`"density"` which uses the standard kernel density estimation routine `density`, and works only for evenly-spaced `r` values;

`"loess"` which uses the function `loess` in the package **modreg**;

`"sm"` which uses the function `sm.density` in the package **sm** and is extremely slow;

`"smrep"` which uses the function `sm.density` in the package **sm** and is relatively fast, but may require manual control of the smoothing parameter `hmult`.

If `normalise=FALSE` then the algorithm will compute only the numerator

$$c_f(r) = E_{0u} f(M(0), M(u))$$

of the expression for the mark correlation function. In this case, negative values of `f` are permitted.

Value

A function value table (object of class `"fv"`) or a list of function value tables, one for each column of marks.

An object of class `"fv"` (see `fv.object`) is essentially a data frame containing numeric columns

`r`	the values of the argument r at which the mark correlation function $k_f(r)$ has been estimated
`theo`	the theoretical value of $k_f(r)$ when the marks attached to different points are independent, namely 1

together with a column or columns named `"iso"` and/or `"trans"`, according to the selected edge corrections. These columns contain estimates of the mark correlation function $k_f(r)$ obtained by the edge corrections named.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

Stoyan, D. and Stoyan, H. (1994) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

See Also

Mark variogram `markvario` for numeric marks.

Mark connection function `markconnect` and multitype K-functions `Kcross`, `Kdot` for factor-valued marks.

Mark cross-correlation function `markcrosscorr` for point patterns with several columns of marks.

`Kmark` to estimate a cumulative function related to the mark correlation function.

Examples

```
# CONTINUOUS-VALUED MARKS:
# (1) Spruces
# marks represent tree diameter
# mark correlation function
ms <- markcorr(spruces)
plot(ms)
```

```
# (2) simulated data with independent marks

  X <- rpoispp(100)
  X <- X %mark% runif(npoints(X))
  Xc <- markcorr(X)
  plot(Xc)


# MULTITYPE DATA:
# Hughes' amacrine data
# Cells marked as 'on'/'off'
X <- if(interactive()) amacrine else amacrine[c(FALSE, TRUE)]
# (3) Kernel density estimate with Epanecnikov kernel
# (as proposed by Stoyan & Stoyan)
M <- markcorr(X, function(m1,m2) {m1==m2},
              correction="translate", method="density",
              kernel="epanechnikov")
# Note: kernel="epanechnikov" comes from help(density)

# (4) Same again with explicit control over bandwidth

  M <- markcorr(X,
              correction="translate", method="density",
              kernel="epanechnikov", bw=0.02)
  # see help(density) for correct interpretation of 'bw'



# weighted mark correlation
X <- if(interactive()) betacells else betacells[c(TRUE,FALSE)]
Y <- subset(X, select=type)
a <- marks(X)$area
v <- markcorr(Y, weights=a)
```

---

markcrosscorr *Mark Cross-Correlation Function*

Description

Given a spatial point pattern with several columns of marks, this function computes the mark correlation function between each pair of columns of marks.

Usage

```
markcrosscorr(X, r = NULL,
              correction = c("isotropic", "Ripley", "translate"),
              method = "density", ..., normalise = TRUE, Xname = NULL)
```

Arguments

X	The observed point pattern. An object of class "ppp" or something acceptable to as.ppp.
r	Optional. Numeric vector. The values of the argument r at which the mark correlation function $k_f(r)$ should be evaluated. There is a sensible default.
correction	A character vector containing any selection of the options "isotropic", "Ripley", "translate", "translation", "none" or "best". It specifies the edge correction(s) to be applied. Alternatively correction="all" selects all options.
method	A character vector indicating the user's choice of density estimation technique to be used. Options are "density", "loess", "sm" and "smrep".
...	Arguments passed to the density estimation routine (density, loess or sm.density) selected by method.
normalise	If normalise=FALSE, compute only the numerator of the expression for the mark correlation.
Xname	Optional character string name for the dataset X.

Details

First, all columns of marks are converted to numerical values. A factor with m possible levels is converted to m columns of dummy (indicator) values.

Next, each pair of columns is considered, and the mark cross-correlation is defined as

$$k_{mm}(r) = \frac{E_{0u}[M_i(0)M_j(u)]}{E[M_i, M_j]}$$

where E_{0u} denotes the conditional expectation given that there are points of the process at the locations 0 and u separated by a distance r. On the numerator, $M_i(0)$ and $M_j(u)$ are the marks attached to locations 0 and u respectively in the ith and jth columns of marks respectively. On the denominator, M_i and M_j are independent random values drawn from the ith and jth columns of marks, respectively, and E is the usual expectation.

Note that $k_{mm}(r)$ is not a "correlation" in the usual statistical sense. It can take any nonnegative real value. The value 1 suggests "lack of correlation": if the marks attached to the points of X are independent and identically distributed, then $k_{mm}(r) \equiv 1$.

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to as.ppp. It must be a marked point pattern.

The cross-correlations are estimated in the same manner as for markcorr.

Value

A function array (object of class "fasp") containing the mark cross-correlation functions for each possible pair of columns of marks.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

markcorr

Examples

```
# The dataset 'betacells' has two columns of marks:
#       'type' (factor)
#       'area' (numeric)
if(interactive()) plot(betacells)
plot(markcrosscorr(betacells))
```

---

markmarkscatter *Mark-Mark Scatter Plot*

Description

Generates the mark-mark scatter plot of a point pattern.

Usage

```
markmarkscatter(X, rmax, ..., col = NULL, symap = NULL, transform=I, jit=FALSE)
```

Arguments

X	A point pattern (object of class "ppp", "pp3", "lpp" or "ppx") with numeric marks.
rmax	Maximum distance between pairs of points which contribute to the plot.
...	Additional arguments passed to plot.ppp to control the scatterplot.
transform	Optional. A function which should be applied to the mark values.
jit	Logical value indicating whether mark values should be randomly perturbed using jitter.
col	Optional. A vector of colour values, or a colourmap to be used to portray the pairwise distance values. Ignored if symap is given.
symap	Optional. A symbolmap to be used to portray the pairwise distance values. Overrides col.

Details

The mark-mark scatter plot (Ballani et al, 2019) is a scatterplot of the mark values of all pairs of distinct points in X which are closer than the distance rmax. The dots in the scatterplot are coloured according to the pairwise distance between the two spatial points. The plot is augmented by three curves explained by Ballani et al (2019).

If the marks only take a few different values, then it is usually appropriate to apply random perturbation (jitter) to the mark values, by setting jit=TRUE.

Value

Null.

Author(s)

Adrian Baddeley (coded from the description in Ballani et al.)

References

Ballani, F., Pommerening, A. and Stoyan, D. (2019) Mark-mark scatterplots improve pattern analysis in spatial plant ecology. *Ecological Informatics* **49**, 13–21.

Examples

```
markmarkscatter(longleaf, 10)

markmarkscatter(spruces, 10, jit=TRUE)
```

---

marktable *Tabulate Marks in Neighbourhood of Every Point in a Point Pattern*

Description

Visit each point in a multitype point pattern, find the neighbouring points, and compile a frequency table of the marks of these neighbour points.

Usage

```
marktable(X, R, N, exclude=TRUE, collapse=FALSE)
```

Arguments

X	A multitype point pattern. An object of class "ppp", "lpp", "pp3" or "ppx", with marks which are a factor.
R	Neighbourhood radius. Incompatible with N.
N	Number of neighbours of each point. Incompatible with R.
exclude	Logical. If exclude=TRUE, the neighbours of a point do not include the point itself. If exclude=FALSE, a point belongs to its own neighbourhood.
collapse	Logical. If collapse=FALSE (the default) the results for each point are returned as separate rows of a table. If collapse=TRUE, the results are aggregated according to the type of point.

Details

This algorithm visits each point in the point pattern X, inspects all the neighbouring points within a radius R of the current point (or the N nearest neighbours of the current point), and compiles a frequency table of the marks attached to the neighbours.

The dataset X must be a multitype point pattern, that is, marks(X) must be a factor.

If collapse=FALSE (the default), the result is a two-dimensional contingency table with one row for each point in the pattern, and one column for each possible mark value. The [i,j] entry in the table gives the number of neighbours of point i that have mark j.

If collapse=TRUE, this contingency table is aggregated according to the type of point, so that the result is a contingency table with one row and one column for each possible mark value. The [i,j] entry in the table gives the number of neighbours of a point with mark i that have mark j.

To perform more complicated calculations on the neighbours of every point, use markstat or applynbd.

Value

A contingency table (object of class "table"). If collapse=FALSE, the table has one row for each point in X, and one column for each possible mark value. If collapse=TRUE, the table has one row and one column for each possible mark value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

markstat, applynbd, Kcross, ppp.object, table

Examples

```
head(marktable(amacrine, 0.1))
head(marktable(amacrine, 0.1, exclude=FALSE))
marktable(amacrine, N=1, collapse=TRUE)
```

---

markvario *Mark Variogram*

Description

Estimate the mark variogram of a marked point pattern.

Usage

```
markvario(X, correction = c("isotropic", "Ripley", "translate"),
r = NULL, method = "density", ..., normalise=FALSE)
```

Arguments

X	The observed point pattern. An object of class "ppp" or something acceptable to as.ppp. It must have marks which are numeric.
correction	A character vector containing any selection of the options "isotropic", "Ripley" or "translate". It specifies the edge correction(s) to be applied.
r	numeric vector. The values of the argument r at which the mark variogram $\gamma(r)$ should be evaluated. There is a sensible default.
method	A character vector indicating the user's choice of density estimation technique to be used. Options are "density", "loess", "sm" and "smrep".
...	Other arguments passed to markcorr, or passed to the density estimation routine (density, loess or sm.density) selected by method.
normalise	If TRUE, normalise the variogram by dividing it by the estimated mark variance.

Details

The mark variogram $\gamma(r)$ of a marked point process X is a measure of the dependence between the marks of two points of the process a distance r apart. It is informally defined as

$$\gamma(r) = E[\frac{1}{2}(M_1 - M_2)^2]$$

where $E[]$ denotes expectation and M_1, M_2 are the marks attached to two points of the process a distance r apart.

The mark variogram of a marked point process is analogous, but **not equivalent**, to the variogram of a random field in geostatistics. See Waelder and Stoyan (1996).

Value

An object of class "fv" (see fv.object).

Essentially a data frame containing numeric columns

r	the values of the argument r at which the mark variogram $\gamma(r)$ has been estimated
theo	the theoretical value of $\gamma(r)$ when the marks attached to different points are independent; equal to the sample variance of the marks

together with a column or columns named "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $\gamma(r)$ obtained by the edge corrections named.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

References

Cressie, N.A.C. (1991) *Statistics for spatial data*. John Wiley and Sons, 1991.

Mase, S. (1996) The threshold method for estimating annual rainfall. *Annals of the Institute of Statistical Mathematics* **48** (1996) 201-213.

Waelder, O. and Stoyan, D. (1996) On variograms in point process statistics. *Biometrical Journal* **38** (1996) 895-905.

See Also

Mark correlation function `markcorr` for numeric marks.

Mark connection function `markconnect` and multitype K-functions `Kcross`, `Kdot` for factor-valued marks.

Examples

```
# Longleaf Pine data
# marks represent tree diameter
# Subset of this large pattern
swcorner <- owin(c(0,100),c(0,100))
sub <- longleaf[ , swcorner]
# mark correlation function
mv <- markvario(sub)
plot(mv)
```

`Math.fasp` *S3 Group Generic Methods for Function Arrays*

Description

These are group generic methods for objects of class `"fasp"`, which allows for usual mathematical functions and operators to be applied directly to function arrays. See Details for a list of implemented functions.

Usage

```
## S3 methods for group generics have prototypes:
Math(x, ...)
Ops(e1, e2)
Complex(z)
Summary(..., na.rm=FALSE, drop=TRUE)
```

Arguments

`x, z, e1, e2`	objects of class `"fasp"`.
`...`	further arguments passed to methods.
`na.rm`	Logical value specifying whether missing values should be removed.

Details

Below is a list of mathematical functions and operators which are defined for objects of class "fasp". The methods are implemented using eval.fasp, which tries to harmonise the functions via harmonise.fv if they aren't compatible to begin with.

1. Group "Math":
   - abs, sign, sqrt,
     floor, ceiling, trunc,
     round, signif
   - exp, log, expm1, log1p,
     cos, sin, tan,
     cospi, sinpi, tanpi,
     acos, asin, atan
     cosh, sinh, tanh,
     acosh, asinh, atanh
   - lgamma, gamma, digamma, trigamma
   - cumsum, cumprod, cummax, cummin
2. Group "Ops":
   - "+", "-", "*", "/", "^", "%%", "%/%"
   - "&", "|", "!"
   - "==", "!=", "<", "<=", ">=", ">"
3. Group "Summary":
   - all, any
   - sum, prod
   - min, max
   - range
4. Group "Complex":
   - Arg, Conj, Im, Mod, Re

For the Ops group, one of the arguments is permitted to be a single atomic value, or a function table, instead of a function array.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

eval.fasp for evaluating expressions involving function arrays.

Examples

```
## convert array of K functions to array of L functions
K <- alltypes(amacrine, "K")
L <- sqrt(K/pi)
```

Math.fv *S3 Group Generic Methods for Function Tables*

Description

These are group generic methods for objects of class "fv", which allows for usual mathematical functions and operators to be applied directly to function tables. See Details for a list of implemented functions.

Usage

```
## S3 methods for group generics have prototypes:
Math(x, ...)
Ops(e1, e2)
Complex(z)
Summary(..., na.rm=FALSE, drop=TRUE)
```

Arguments

x, z, e1, e2	objects of class "fv".
...	further arguments passed to methods.
na.rm	Logical value specifying whether missing values should be removed.

Details

Below is a list of mathematical functions and operators which are defined for objects of class "fv". The methods are implemented using eval.fv, which tries to harmonise the functions via harmonise.fv if they aren't compatible to begin with.

1. Group "Math":
   - abs, sign, sqrt,
     floor, ceiling, trunc,
     round, signif
   - exp, log, expm1, log1p,
     cos, sin, tan,
     cospi, sinpi, tanpi,
     acos, asin, atan
     cosh, sinh, tanh,
     acosh, asinh, atanh
   - lgamma, gamma, digamma, trigamma
   - cumsum, cumprod, cummax, cummin
2. Group "Ops":
   - "+", "-", "*", "/", "^", "%%", "%/%"
   - "&", "|", "!"
   - "==", "!=", "<", "<=", ">=", ">"

3. Group "Summary":
    - all, any
    - sum, prod
    - min, max
    - range
4. Group "Complex":
    - Arg, Conj, Im, Mod, Re

For the Ops group, one of the arguments is permitted to be a single atomic value instead of a function table.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

eval.fv for evaluating expressions involving function tables.

Examples

```
## Convert K function to L function
K <- Kest(cells)
L <- sqrt(K/pi)
## Manually calculate J function
FR <- Fest(redwood)
GR <- Gest(redwood)
suppressWarnings(JR <- (1-GR)/(1-FR))
```

---

methods.rho2hat *Methods for Intensity Functions of Two Spatial Covariates*

Description

These are methods for the class "rho2hat".

Usage

```
## S3 method for class 'rho2hat'
plot(x, ..., do.points=FALSE)

## S3 method for class 'rho2hat'
print(x, ...)

## S3 method for class 'rho2hat'
predict(object, ..., relative=FALSE)
```

Arguments

x, object	An object of class "rho2hat".
...	Arguments passed to other methods.
do.points	Logical value indicating whether to plot the observed values of the covariates at the data points.
relative	Logical value indicating whether to compute the estimated point process intensity (relative=FALSE) or the relative risk (relative=TRUE) in the case of a relative risk estimate.

Details

These functions are methods for the generic commands print, predict and plot for the class "rho2hat".

An object of class "rho2hat" is an estimate of the intensity of a point process, as a function of two given spatial covariates. See rho2hat.

The method plot.rho2hat displays the estimated function ρ using plot.fv, and optionally adds a rug plot of the observed values of the covariate. In this plot the two axes represent possible values of the two covariates.

The method predict.rho2hat computes a pixel image of the intensity $\rho(Z_1(u), Z_2(u))$ at each spatial location u, where $Z_1(u)$ and $Z_2(u)$ are the two spatial covariates.

Value

For predict.rho2hat the value is a pixel image (object of class "im"). For other functions, the value is NULL.

Author(s)

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

See Also

rho2hat

Examples

```
r2 <- with(bei.extra, rho2hat(bei, elev, grad))
r2
plot(r2)
plot(predict(r2))
```

`methods.rhohat` *Methods for Intensity Functions of Spatial Covariate*

Description

These are methods for the class `"rhohat"`.

Usage

```
## S3 method for class 'rhohat'
print(x, ...)

## S3 method for class 'rhohat'
plot(x, ..., do.rug=TRUE)

## S3 method for class 'rhohat'
predict(object, ..., relative=FALSE,
             what=c("rho", "lo", "hi", "se"))

## S3 method for class 'rhohat'
simulate(object, nsim=1, ..., drop=TRUE)
```

Arguments

`x, object`	An object of class `"rhohat"` representing a smoothed estimate of the intensity function of a point process.
`...`	Arguments passed to other methods.
`do.rug`	Logical value indicating whether to plot the observed values of the covariate as a rug plot along the horizontal axis.
`relative`	Logical value indicating whether to compute the estimated point process intensity (`relative=FALSE`) or the relative risk (`relative=TRUE`) in the case of a relative risk estimate.
`nsim`	Number of simulations to be generated.
`drop`	Logical value indicating what to do when `nsim=1`. If `drop=TRUE` (the default), a point pattern is returned. If `drop=FALSE`, a list of length 1 containing a point pattern is returned.
`what`	Optional character string (partially matched) specifying which value should be calculated: either the function estimate (`what="rho"`, the default), the lower or upper end of the confidence interval (`what="lo"` or `what="hi"`) or the standard error (`what="se"`).

Details

These functions are methods for the generic commands `print`, `plot`, `predict` and `simulate` for the class `"rhohat"`.

An object of class "rhohat" is an estimate of the intensity of a point process, as a function of a given spatial covariate. See rhohat.

The method plot.rhohat displays the estimated function ρ using plot.fv, and optionally adds a rug plot of the observed values of the covariate.

The method predict.rhohat computes a pixel image of the intensity $\rho(Z(u))$ at each spatial location u, where Z is the spatial covariate.

The method simulate.rhohat invokes predict.rhohat to determine the predicted intensity, and then simulates a Poisson point process with this intensity.

Value

For predict.rhohat the value is a pixel image (object of class "im" or "linim"). For simulate.rhohat the value is a point pattern (object of class "ppp" or "lpp"). For other functions, the value is NULL.

Author(s)

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

See Also

rhohat

Examples

```
X <-  rpoispp(function(x,y){exp(3+3*x)})
rho <- rhohat(X, function(x,y){x})
rho
plot(rho)
Y <- predict(rho)
plot(Y)
plot(simulate(rho), add=TRUE)
#
if(require("spatstat.model")) {
  fit <- ppm(X, ~x)
  rho <- rhohat(fit, "y")
  opa <- par(mfrow=c(1,2))
  plot(predict(rho))
  plot(predict(rho, relative=TRUE))
  par(opa)
  plot(predict(rho, what="se"))
}
```

---

methods.ssf *Methods for Spatially Sampled Functions*

Description

Methods for various generic commands, for the class "ssf" of spatially sampled functions.

Usage

```
  ## S3 method for class 'ssf'
marks(x, ...)

  ## S3 replacement method for class 'ssf'
marks(x, ...) <- value

  ## S3 method for class 'ssf'
unmark(X)

  ## S3 method for class 'ssf'
as.im(X, ...)

  ## S3 method for class 'ssf'
as.function(x, ...)

  ## S3 method for class 'ssf'
as.ppp(X, ...)

  ## S3 method for class 'ssf'
print(x, ..., brief=FALSE)

  ## S3 method for class 'ssf'
summary(object, ...)

  ## S3 method for class 'ssf'
range(x, ...)

  ## S3 method for class 'ssf'
min(x, ...)

  ## S3 method for class 'ssf'
max(x, ...)

  ## S3 method for class 'ssf'
integral(f, domain=NULL, ..., weights=attr(f, "weights"))
```

Arguments

x, X, f, object	A spatially sampled function (object of class "ssf").
...	Arguments passed to the default method.
brief	Logical value controlling the amount of detail printed.
value	Matrix of replacement values for the function.
domain	Optional. Domain of integration. An object of class"owin" or "tess".
weights	Optional. Numeric vector of *quadrature weights* associated with the sample points.

Details

An object of class "ssf" represents a function (real- or vector-valued) that has been sampled at a finite set of points.

The commands documented here are methods for this class, for the generic commands marks, marks<-, unmark, as.im, as.function, as.ppp, print, summary, range, min, max and integral.

Value

marks returns a matrix.

marks(x) <- value returns an object of class "ssf".

as.owin returns a window (object of class "owin").

as.ppp and unmark return a point pattern (object of class "ppp").

as.function returns a function(x,y) of class "funxy".

print returns NULL.

summary returns an object of class "summary.ssf" which has a print method.

range returns a numeric vector of length 2. min and max return a single numeric value.

integral returns a numeric or complex value, vector, or matrix. integral(f) returns a numeric or complex value (if f had numeric or complex values) or a numeric vector (if f had vector values). If domain is a tessellation then integral(f, domain) returns a numeric or complex vector with one entry for each tile (if f had numeric or complex values) or a numeric matrix with one row for each tile (if f had vector values).

Author(s)

Adrian Baddeley

See Also

ssf

Examples

```
g <- distfun(cells[1:4])
X <- rsyst(Window(cells), 10)
f <- ssf(X, g(X))
f
summary(f)
marks(f)
as.ppp(f)
as.im(f)
integral(f)
integral(f, quadrats(Window(f), 3))
```

`miplot` *Morisita Index Plot*

Description

Displays the Morisita Index Plot of a spatial point pattern.

Usage

```
miplot(X, ...)
```

Arguments

`X`	A point pattern (object of class `"ppp"`) or something acceptable to `as.ppp`.
`...`	Optional arguments to control the appearance of the plot.

Details

Morisita (1959) defined an index of spatial aggregation for a spatial point pattern based on quadrat counts. The spatial domain of the point pattern is first divided into Q subsets (quadrats) of equal size and shape. The numbers of points falling in each quadrat are counted. Then the Morisita Index is computed as

$$\mathrm{MI} = Q \frac{\sum_{i=1}^{Q} n_i(n_i - 1)}{N(N-1)}$$

where n_i is the number of points falling in the i-th quadrat, and N is the total number of points. If the pattern is completely random, `MI` should be approximately equal to 1. Values of `MI` greater than 1 suggest clustering.

The *Morisita Index plot* is a plot of the Morisita Index `MI` against the linear dimension of the quadrats. The point pattern dataset is divided into 2×2 quadrats, then 3×3 quadrats, etc, and the Morisita Index is computed each time. This plot is an attempt to discern different scales of dependence in the point pattern data.

Value

None.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

References

M. Morisita (1959) Measuring of the dispersion of individuals and analysis of the distributional patterns. Memoir of the Faculty of Science, Kyushu University, Series E: Biology. **2**: 215–235.

See Also

quadratcount

Examples

```
miplot(longleaf)
opa <- par(mfrow=c(2,3))
plot(cells)
plot(japanesepines)
plot(redwood)
miplot(cells)
miplot(japanesepines)
miplot(redwood)
par(opa)
```

MISE.envelope *Mean Integrated Squared Error on an Envelope Object*

Description

Compute the mean integrated squared error, or integrated squared bias, or integrated variance, of the simulated function estimates in an envelope object.

Usage

```
ISB.envelope(object, theo, domain, dimension=2)
IV.envelope(object, domain, dimension=2)
MISE.envelope(object, theo, domain, dimension=2)
```

Arguments

object	Object of class "envelope" generated by the function envelope
theo	Function in the R language that evaluates the true (theoretically expected) value of the spatial summary function.
domain	Numeric vector of length 2 specifying the limits of the domain of integration for the integrated squared error.
dimension	Integer (either 1 or 2) specifying whether to calculate the one-dimensional or two-dimensional integral of squared error.

Details

The first argument should be an object of class "envelope" and should contain the simulated function estimates (i.e. it should have been computed using envelope with savefuns=TRUE).

MISE.envelope computes the mean integrated squared error. ISB.envelope computes the integrated squared bias. IV.envelope computes the integrated sample variance.

The simulated function estimates are extracted from `object` and their deviation from the true function `theo` is computed pointwise. The squared deviations are integrated over the interval specified by `domain`, giving one value of integrated squared error for each simulated function estimate. Then `MISE.envelope` returns the average of these values, that is, the estimated mean squared error. Similarly for the other computations.

Value

A single numeric value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Martin Hazelton <Martin.Hazelton@otago.ac.nz> and Tilman Davies <Tilman.Davies@otago.ac.nz>.

See Also

`bias.envelope`, `ISE.envelope`.

Examples

```
E <- envelope(cells, Kest, nsim=20, savefuns=TRUE)
theoK <- function(r) { pi * r^2 }
dom <- c(0, 0.1)
MISE.envelope(E, theoK, dom)
ISB.envelope(E, theoK, dom)
```

---

nnclean *Nearest Neighbour Clutter Removal*

Description

Detect features in a 2D or 3D spatial point pattern using nearest neighbour clutter removal.

Usage

```
  nnclean(X, k, ...)

  ## S3 method for class 'ppp'
nnclean(X, k, ...,
                     edge.correct = FALSE, wrap = 0.1,
                     convergence = 0.001, plothist = FALSE,
                     verbose = TRUE, maxit = 50)

  ## S3 method for class 'pp3'
nnclean(X, k, ...,
                     convergence = 0.001, plothist = FALSE,
                     verbose = TRUE, maxit = 50)
```

Arguments

X	A two-dimensional spatial point pattern (object of class "ppp") or a three-dimensional point pattern (object of class "pp3").
k	Degree of neighbour: k=1 means nearest neighbour, k=2 means second nearest, etc.
...	Arguments passed to hist.default to control the appearance of the histogram, if plothist=TRUE.
edge.correct	Logical flag specifying whether periodic edge correction should be performed (only implemented in 2 dimensions).
wrap	Numeric value specifying the relative size of the margin in which data will be replicated for the periodic edge correction (if edge.correct=TRUE). A fraction of window width and window height.
convergence	Relative tolerance threshold for testing convergence of EM algorithm.
maxit	Maximum number of iterations for EM algorithm.
plothist	Logical flag specifying whether to plot a diagnostic histogram of the nearest neighbour distances and the fitted distribution.
verbose	Logical flag specifying whether to print progress reports.

Details

Byers and Raftery (1998) developed a technique for recognising features in a spatial point pattern in the presence of random clutter.

For each point in the pattern, the distance to the kth nearest neighbour is computed. Then the E-M algorithm is used to fit a mixture distribution to the kth nearest neighbour distances. The mixture components represent the feature and the clutter. The mixture model can be used to classify each point as belong to one or other component.

The function nnclean is generic, with methods for two-dimensional point patterns (class "ppp") and three-dimensional point patterns (class "pp3") currently implemented.

The result is a point pattern (2D or 3D) with two additional columns of marks:

class A factor, with levels "noise" and "feature", indicating the maximum likelihood classification of each point.

prob Numeric vector giving the estimated probabilities that each point belongs to a feature.

The object also has extra information stored in attributes: "theta" contains the fitted parameters of the mixture model, "info" contains information about the fitting procedure, and "hist" contains the histogram structure returned from hist.default if plothist = TRUE.

Value

An object of the same kind as X, obtained by attaching marks to the points of X.

The object also has attributes, as described under Details.

Author(s)

Original by Simon Byers and Adrian Raftery. Adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au

References

Byers, S. and Raftery, A.E. (1998) Nearest-neighbour clutter removal for estimating features in spatial point processes. *Journal of the American Statistical Association* **93**, 577–584.

See Also

nndist, split.ppp, cut.ppp

Examples

```
# shapley galaxy cluster
X <- nnclean(shapley, k=17, plothist=TRUE)
plot(X, which.marks=1, chars=c(".", "+"), cols=1:2,
     main="Shapley data, cluster and noise")
plot(X, which.marks=2, cols=function(x)hsv(0.2+0.8*(1-x),1,1),
     main="Shapley data, probability of cluster")
Y <- split(X, un=TRUE)
plot(Y, chars="+", cex=0.5)
marks(X) <- marks(X)$prob
plot(cut(X, breaks=3), chars=c(".", "+", "+"), cols=1:3)
```

---

nncorr *Nearest-Neighbour Correlation Indices of Marked Point Pattern*

Description

Computes nearest-neighbour correlation indices of a marked point pattern, including the nearest-neighbour mark product index (default case of nncorr), the nearest-neighbour mark index (nnmean), and the nearest-neighbour variogram index (nnvario).

Usage

```
nncorr(X,
       f = function(m1, m2) { m1 * m2 },
       k = 1,
       ...,
       use = "all.obs", method = c("pearson", "kendall", "spearman"),
       denominator=NULL, na.action="warn")

nnmean(X, k=1, na.action="warn")

nnvario(X, k=1, na.action="warn")
```

Arguments

X	The observed point pattern. An object of class "ppp".
f	Function f used in the definition of the nearest neighbour correlation. There is a sensible default that depends on the type of marks of X.
k	Integer. The k-th nearest neighbour of each point will be used.
...	Extra arguments passed to f.
use, method	Arguments passed to the standard correlation function cor.
denominator	Internal use only.
na.action	Character string (passed to is.marked.ppp) specifying what to do if the marks contain NA values.

Details

The nearest neighbour correlation index $\bar{n}_f$ of a marked point process X is a number measuring the dependence between the mark of a typical point and the mark of its nearest neighbour.

The command nncorr computes the nearest neighbour correlation index based on any test function f provided by the user. The default behaviour of nncorr is to compute the nearest neighbour mark product index. The commands nnmean and nnvario are convenient abbreviations for other special choices of f.

In the default case, nncorr(X) computes three different versions of the nearest-neighbour correlation index: the unnormalised, normalised, and classical correlations.

unnormalised: The **unnormalised** nearest neighbour correlation (Stoyan and Stoyan, 1994, section 14.7) is defined as

$$\bar{n}_f = E[f(M, M^*)]$$

where $E[]$ denotes mean value, M is the mark attached to a typical point of the point process, and M^* is the mark attached to its nearest neighbour (i.e. the nearest other point of the point process).

Here f is any function $f(m_1, m_2)$ with two arguments which are possible marks of the pattern, and which returns a nonnegative real value. Common choices of f are: for continuous real-valued marks,

$$f(m_1, m_2) = m_1 m_2$$

for discrete marks (multitype point patterns),

$$f(m_1, m_2) = 1(m_1 = m_2)$$

and for marks taking values in $[0, 2\pi)$,

$$f(m_1, m_2) = \sin(m_1 - m_2)$$

For example, in the second case, the unnormalised nearest neighbour correlation $\bar{n}_f$ equals the proportion of points in the pattern which have the same mark as their nearest neighbour.

Note that $\bar{n}_f$ is not a "correlation" in the usual statistical sense. It can take values greater than 1.

normalised: We can define a **normalised** nearest neighbour correlation by

$$\bar{m}_f = \frac{E[f(M, M^*)]}{E[f(M, M')]}$$

where again M is the mark attached to a typical point, M^* is the mark attached to its nearest neighbour, and M' is an independent copy of M with the same distribution. This normalisation is also not a "correlation" in the usual statistical sense, but is normalised so that the value 1 suggests "lack of correlation": if the marks attached to the points of `X` are independent and identically distributed, then $\bar{m}_f = 1$. The interpretation of values larger or smaller than 1 depends on the choice of function f.

classical: Finally if the marks of `X` are real numbers, we can also compute the **classical** correlation, that is, the correlation coefficient of the two random variables M and M^*. The classical correlation has a value between -1 and 1. Values close to -1 or 1 indicate strong dependence between the marks.

In the default case where `f` is not given, `nncorr(X)` computes

- If the marks of `X` are real numbers, the unnormalised and normalised versions of the nearest-neighbour product index $E[M\, M^*]$, and the classical correlation between M and M^*.
- If the marks of `X` are factor valued, the unnormalised and normalised versions of the nearest-neighbour equality index $P[M = M^*]$.

The wrapper functions `nnmean` and `nnvario` compute the correlation indices for two special choices of the function $f(m_1, m_2)$. They are defined only when the marks are numeric.

- `nnmean` computes the correlation indices for $f(m_1, m_2) = m_1$. The unnormalised index is simply the mean value of the mark of the neighbour of a typical point, $E[M^*]$, while the normalised index is $E[M^*]/E[M]$, the ratio of the mean mark of the neighbour of a typical point to the mean mark of a typical point.
- `nnvario` computes the correlation indices for $f(m_1, m_2) = (1/2)(m_1 - m_2)^2$.

The argument `X` must be a point pattern (object of class `"ppp"`) and must be a marked point pattern. (The marks may be a data frame, containing several columns of mark variables; each column is treated separately.)

If the argument `f` is given, it must be a function, accepting two arguments `m1` and `m2` which are vectors of equal length containing mark values (of the same type as the marks of `X`). It must return a vector of numeric values of the same length as `m1` and `m2`. The values must be non-negative.

The arguments `use` and `method` control the calculation of the classical correlation using `cor`, as explained in the help file for `cor`.

Other arguments may be passed to `f` through the `...` argument.

This algorithm assumes that `X` can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in `X` as `Window(X)`) may have arbitrary shape. Biases due to edge effects are treated using the 'border method' edge correction.

Value

Labelled vector of length 2 or 3 containing the unnormalised and normalised nearest neighbour correlations, and the classical correlation if appropriate. Alternatively a matrix with 2 or 3 rows, containing this information for each mark variable.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

References

Stoyan, D. and Stoyan, H. (1994) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

Examples

```
nnmean(finpines)
nnvario(finpines)
nncorr(finpines)
# heights of neighbouring trees are slightly negatively correlated

nncorr(amacrine)
# neighbouring cells are usually of different type
```

---

nndensity.ppp *Estimate Intensity of Point Pattern Using Nearest Neighbour Distances*

Description

Estimates the intensity of a point pattern using the distance from each spatial location to the kth nearest data point.

Usage

```
nndensity(x, ...)

## S3 method for class 'ppp'
nndensity(x, k, ..., verbose = TRUE)
```

Arguments

x	A point pattern (object of class "ppp") or some other spatial object.
k	Integer. The distance to the kth nearest data point will be computed. There is a sensible default.
...	Arguments passed to nnmap and as.mask controlling the pixel resolution.
verbose	Logical. If TRUE, print the value of k when it is automatically selected. If FALSE, remain silent.

Details

This function computes a quick estimate of the intensity of the point process that generated the point pattern x.

For each spatial location s, let $d(s)$ be the distance from s to the k-th nearest point in the dataset x. If the data came from a homogeneous Poisson process with intensity λ, then $\pi d(s)^2$ would follow a negative exponential distribution with mean $1/\lambda$, and the maximum likelihood estimate of λ would be $1/(\pi d(s)^2)$. This is the estimate computed by nndensity, apart from an edge effect correction.

See Cressie (1991, equation (8.5.14), p. 654) and Silverman (1986, p. 96).

This estimator of intensity is relatively fast to compute, and is spatially adaptive (so that it can handle wide variation in the intensity function). However, it implicitly assumes the points are independent, so it does not perform well if the pattern is strongly clustered or strongly inhibited.

In normal use, the value of k should be at least 3. (Theoretically the estimator has infinite expected value if $k = 1$, and infinite variance if $k = 2$. The computed intensity estimate will have infinite peaks around each data point if k = 1.) The default value of k is the square root of the number of points in x, which seems to work well in many cases.

The window of x is digitised using as.mask and the values $d(s)$ are computed using nnmap. To control the pixel resolution, see as.mask.

Value

A pixel image (object of class "im") giving the estimated intensity of the point process at each spatial location. Pixel values are intensities (number of points per unit area).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>.

References

Cressie, N.A.C. (1991) *Statistics for spatial data*. John Wiley and Sons, New York.

Silverman, B.W. (1986) *Density Estimation*. Chapman and Hall, New York.

See Also

density.ppp, intensity.ppp for alternative estimates of point process intensity.

Examples

```
plot(nndensity(swedishpines))
```

nnorient *Nearest Neighbour Orientation Distribution*

Description

Computes the distribution of the orientation of the vectors from each point to its nearest neighbour.

Usage

```
nnorient(X, ..., cumulative = FALSE, correction, k = 1,
                  unit = c("degree", "radian"),
                  domain = NULL, ratio = FALSE)
```

Arguments

X	Point pattern (object of class "ppp").
...	Arguments passed to circdensity to control the kernel smoothing, if cumulative=FALSE.
cumulative	Logical value specifying whether to estimate the probability density (cumulative=FALSE, the default) or the cumulative distribution function (cumulative=TRUE).
correction	Character vector specifying edge correction or corrections. Options are "none", "bord.modif", "good" and "best". Alternatively correction="all" selects all options.
k	Integer. The kth nearest neighbour will be used.
ratio	Logical. If TRUE, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.
unit	Unit in which the angles should be expressed. Either "degree" or "radian".
domain	Optional window. The first point x_i of each pair of points will be constrained to lie in domain.

Details

This algorithm considers each point in the pattern X and finds its nearest neighbour (or kth nearest neighour). The *direction* of the arrow joining the data point to its neighbour is measured, as an angle in degrees or radians, anticlockwise from the x axis.

If cumulative=FALSE (the default), a kernel estimate of the probability density of the angles is calculated using circdensity. This is the function $\vartheta(\phi)$ defined in Illian et al (2008), equation (4.5.3), page 253.

If cumulative=TRUE, then the cumulative distribution function of these angles is calculated.

In either case the result can be plotted as a rose diagram by rose, or as a function plot by plot.fv.

The algorithm gives each observed direction a weight, determined by an edge correction, to adjust for the fact that some interpoint distances are more likely to be observed than others. The choice of edge correction or corrections is determined by the argument correction.

It is also possible to calculate an estimate of the probability density from the cumulative distribution function, by numerical differentiation. Use deriv.fv with the argument Dperiodic=TRUE.

Value

A function value table (object of class "fv") containing the estimates of the probability density or the cumulative distribution function of angles, in degrees (if unit="degree") or radians (if unit="radian").

Author(s)

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

Rolf Turner <rolfturner@posteo.net>

and Ege Rubak <rubak@math.aau.dk>

References

Illian, J., Penttinen, A., Stoyan, H. and Stoyan, D. (2008) *Statistical Analysis and Modelling of Spatial Point Patterns.* Wiley.

See Also

pairorient

Examples

```
rose(nnorient(redwood, adjust=0.6), col="grey")
plot(CDF <- nnorient(redwood, cumulative=TRUE))
```

---

pairMean *Mean of a Function of Interpoint Distance*

Description

Computes the mean value, or the double integral, of a specified function of the distance between two independent random points in a given window or windows.

Usage

```
pairMean(fun, W, V = NULL, ..., normalise = TRUE)
```

Arguments

fun	A function in the R language which takes one argument.
W	A window (object of class "owin") containing the first random point.
V	Optional. Another window containing the second random point. Defaults to W.
...	Further optional arguments passed to distcdf to determine the pixel resolution for the calculation and the probability distributions of the random points.
normalise	Logical value specifying whether to calculate the mean value (normalise=TRUE, the default) or the double integral (normalise=FALSE).

Details

This command computes the mean value of `fun(T)` where `T` is the Euclidean distance $T = \|X_1 - X_2\|$ between two independent random points X_1 and X_2.

In the simplest case, the command `pairMean(fun, W)`, the random points are assumed to be uniformly distributed in the same window `W`. Alternatively the two random points may be uniformly distributed in two different windows `W` and `V`. Other options are described in `distcdf`.

The algorithm uses `distcdf` to compute the cumulative distribution function of `T`, and `stieltjes` to compute the mean value of `fun(T)`.

If `normalise=TRUE` (the default) the result is the mean value of `fun(T)`. If `normalise=FALSE` the result is the double integral.

Value

A single numeric value.

Author(s)

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

See Also

`distcdf`

Examples

```
pairMean(function(d) { d^2 }, disc())
```

---

pairorient *Point Pair Orientation Distribution*

Description

Computes the distribution of the orientation of vectors joining pairs of points at a particular range of distances.

Usage

```
pairorient(X, r1, r2, ..., cumulative=FALSE,
           correction, ratio = FALSE,
           unit=c("degree", "radian"), domain=NULL)
```

Arguments

X	Point pattern (object of class "ppp").
r1, r2	Minimum and maximum values of distance to be considered.
...	Arguments passed to circdensity to control the kernel smoothing, if cumulative=FALSE.
cumulative	Logical value specifying whether to estimate the probability density (cumulative=FALSE, the default) or the cumulative distribution function (cumulative=TRUE).
correction	Character vector specifying edge correction or corrections. Options are "none", "isotropic", "translate", "border", "bord.modif", "good" and "best". Alternatively correction="all" selects all options. The default is to compute all edge corrections except "none".
ratio	Logical. If TRUE, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.
unit	Unit in which the angles should be expressed. Either "degree" or "radian".
domain	Optional window. The first point x_i of each pair of points will be constrained to lie in domain.

Details

This algorithm considers all pairs of points in the pattern X that lie more than r1 and less than r2 units apart. The *direction* of the arrow joining the points is measured, as an angle in degrees or radians, anticlockwise from the x axis.

If cumulative=FALSE (the default), a kernel estimate of the probability density of the orientations is calculated using circdensity.

If cumulative=TRUE, then the cumulative distribution function of these directions is calculated. This is the function $O_{r1,r2}(\phi)$ defined in Stoyan and Stoyan (1994), equation (14.53), page 271.

In either case the result can be plotted as a rose diagram by rose, or as a function plot by plot.fv.

The algorithm gives each observed direction a weight, determined by an edge correction, to adjust for the fact that some interpoint distances are more likely to be observed than others. The choice of edge correction or corrections is determined by the argument correction. See the help for Kest for details of edge corrections, and explanation of the options available. The choice correction="none" is not recommended; it is included for demonstration purposes only. The default is to compute all corrections except "none".

It is also possible to calculate an estimate of the probability density from the cumulative distribution function, by numerical differentiation. Use deriv.fv with the argument Dperiodic=TRUE.

Value

A function value table (object of class "fv") containing the estimates of the probability density or the cumulative distribution function of angles, in degrees (if unit="degree") or radians (if unit="radian").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

Stoyan, D. and Stoyan, H. (1994) Fractals, Random Shapes and Point Fields: Methods of Geometrical Statistics. John Wiley and Sons.

See Also

Kest, Ksector, nnorient

Examples

```
rose(pairorient(redwood, 0.05, 0.15, sigma=8), col="grey")
plot(CDF <- pairorient(redwood, 0.05, 0.15, cumulative=TRUE))
plot(f <- deriv(CDF, spar=0.6, Dperiodic=TRUE))
```

---

pairs.im *Scatterplot Matrix for Pixel Images*

Description

Produces a scatterplot matrix of the pixel values in two or more pixel images.

Usage

```
## S3 method for class 'im'
pairs(..., plot=TRUE, drop=TRUE)
```

Arguments

...	Any number of arguments, each of which is either a pixel image (object of class "im") or a named argument to be passed to pairs.default. Alternatively, a single argument which is a list of pixel images.
plot	Logical. If TRUE, the scatterplot matrix is plotted.
drop	Logical value specifying whether pixel values that are NA should be removed from the data frame that is returned by the function. This does not affect the plot.

Details

This is a method for the generic function pairs for the class of pixel images.

It produces a square array of plot panels, in which each panel shows a scatterplot of the pixel values of one image against the corresponding pixel values of another image.

At least two of the arguments ... should be pixel images (objects of class "im"). Their spatial domains must overlap, but need not have the same pixel dimensions.

First the pixel image domains are intersected, and converted to a common pixel resolution. Then the corresponding pixel values of each image are extracted. Then pairs.default is called to plot the scatterplot matrix.

Any arguments in ... which are not pixel images will be passed to pairs.default to control the plot.

The return value of pairs.im is a data frame, returned invisibly. The data frame has one column for each image. Each row contains the pixel values of the different images for one pixel in the raster. If drop=TRUE (the default), any row which contains NA is deleted. The plot is not affected by the value of drop.

Value

Invisible. A data.frame containing the corresponding pixel values for each image. The return value also belongs to the class plotpairsim which has a plot method, so that it can be re-plotted.

Image or Contour Plots

Since the scatterplots may show very dense concentrations of points, it may be useful to set panel=panel.image or panel=panel.contour to draw a colour image or contour plot of the kernel-smoothed density of the scatterplot in each panel. The argument panel is passed to pairs.default. See the help for panel.image and panel.contour.

Low Level Control of Graphics

To control the appearance of the individual scatterplot panels, see pairs.default, points or par. To control the plotting symbol for the points in the scatterplot, use the arguments pch, col, bg as described under points (because the default panel plotter is the function points). To suppress the tick marks on the plot axes, type par(xaxt="n", yaxt="n") before calling pairs.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

pairs, pairs.default, panel.contour, panel.image, plot.im, cov.im, im, par

Examples

```
X <- density(rpoispp(30))
Y <- density(rpoispp(40))
Z <- density(rpoispp(30))
p <- pairs(X,Y,Z)
p
plot(p)
```

panel.contour *Panel Plots using Colour Image or Contour Lines*

Description

These functions can be passed to pairs or coplot to determine what kind of plotting is done in each panel of a multi-panel graphical display.

Usage

```
panel.contour(x, y, ..., sigma = NULL)

panel.image(x, y, ..., sigma = NULL)

panel.histogram(x, ...)
```

Arguments

x, y	Coordinates of points in a scatterplot.
...	Extra graphics arguments, passed to contour.im, plot.im or rect, respectively, to control the appearance of the panel.
sigma	Bandwidth of kernel smoother, on a scale where x and y range between 0 and 1.

Details

These functions can serve as one of the arguments panel, lower.panel, upper.panel, diag.panel passed to graphics commands like pairs or coplot, to determine what kind of plotting is done in each panel of a multi-panel graphical display. In particular they work with pairs.im.

The functions panel.contour and panel.contour are suitable for the off-diagonal plots which involve two datasets x and y. They first rescale x and y to the unit square, then apply kernel smoothing with bandwidth sigma using density.ppp. Then panel.contour draws a contour plot while panel.image draws a colour image.

The function panel.histogram is suitable for the diagonal plots which involve a single dataset x. It displays a histogram of the data.

Value

Null.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

pairs.im, pairs.default, panel.smooth

Examples

```
pairs(bei.extra,
      panel      = panel.contour,
      diag.panel = panel.histogram)
with(bei.extra,
      pairs(grad, elev,
            panel      = panel.image,
            diag.panel = panel.histogram))
pairs(marks(finpines), panel=panel.contour, diag.panel=panel.histogram)
```

---

pcf *Pair Correlation Function*

Description

Estimate the pair correlation function.

Usage

```
pcf(X, ...)
```

Arguments

X	Either the observed data point pattern, or an estimate of its K function, or an array of multitype K functions (see Details).
...	Other arguments passed to the appropriate method.

Details

The pair correlation function of a stationary point process is

$$g(r) = \frac{K'(r)}{2\pi r}$$

where $K'(r)$ is the derivative of $K(r)$, the reduced second moment function (aka "Ripley's K function") of the point process. See Kest for information about $K(r)$. For a stationary Poisson process, the pair correlation function is identically equal to 1. Values $g(r) < 1$ suggest inhibition between points; values greater than 1 suggest clustering.

We also apply the same definition to other variants of the classical K function, such as the multitype K functions (see Kcross, Kdot) and the inhomogeneous K function (see Kinhom). For all these variants, the benchmark value of $K(r) = \pi r^2$ corresponds to $g(r) = 1$.

This routine computes an estimate of $g(r)$ either directly from a point pattern, or indirectly from an estimate of $K(r)$ or one of its variants.

This function is generic, with methods for the classes "ppp", "fv" and "fasp".

If X is a point pattern (object of class "ppp") then the pair correlation function is estimated using a traditional kernel smoothing method (Stoyan and Stoyan, 1994). See pcf.ppp for details.

If X is a function value table (object of class "fv"), then it is assumed to contain estimates of the K function or one of its variants (typically obtained from Kest or Kinhom). This routine computes an estimate of $g(r)$ using smoothing splines to approximate the derivative. See pcf.fv for details.

If X is a function value array (object of class "fasp"), then it is assumed to contain estimates of several K functions (typically obtained from Kmulti or alltypes). This routine computes an estimate of $g(r)$ for each cell in the array, using smoothing splines to approximate the derivatives. See pcf.fasp for details.

Value

Either a function value table (object of class "fv", see fv.object) representing a pair correlation function, or a function array (object of class "fasp", see fasp.object) representing an array of pair correlation functions.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>

References

Stoyan, D. and Stoyan, H. (1994) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

See Also

pcf.ppp, pcf.fv, pcf.fasp, Kest, Kinhom, Kcross, Kdot, Kmulti, alltypes

Examples

```
# ppp object
X <- simdat

p <- pcf(X)
plot(p)

# fv object
K <- Kest(X)
p2 <- pcf(K, spar=0.8, method="b")
plot(p2)

# multitype pattern; fasp object
amaK <- alltypes(amacrine, "K")
amap <- pcf(amaK, spar=1, method="b")
plot(amap)
```

pcf.fasp *Pair Correlation Function obtained from array of K functions*

Description

Estimates the (bivariate) pair correlation functions of a point pattern, given an array of (bivariate) K functions.

Usage

```
 ## S3 method for class 'fasp'
pcf(X, ..., method="c")
```

Arguments

X	An array of multitype K functions (object of class "fasp").
...	Arguments controlling the smoothing spline function smooth.spline.
method	Letter "a", "b", "c" or "d" indicating the method for deriving the pair correlation function from the K function.

Details

The pair correlation function of a stationary point process is

$$g(r) = \frac{K'(r)}{2\pi r}$$

where $K'(r)$ is the derivative of $K(r)$, the reduced second moment function (aka "Ripley's K function") of the point process. See Kest for information about $K(r)$. For a stationary Poisson process, the pair correlation function is identically equal to 1. Values $g(r) < 1$ suggest inhibition between points; values greater than 1 suggest clustering.

We also apply the same definition to other variants of the classical K function, such as the multitype K functions (see Kcross, Kdot) and the inhomogeneous K function (see Kinhom). For all these variants, the benchmark value of $K(r) = \pi r^2$ corresponds to $g(r) = 1$.

This routine computes an estimate of $g(r)$ from an array of estimates of $K(r)$ or its variants, using smoothing splines to approximate the derivatives. It is a method for the generic function pcf.

The argument X should be a function array (object of class "fasp", see fasp.object) containing several estimates of K functions. This should have been obtained from alltypes with the argument fun="K".

The smoothing spline operations are performed by smooth.spline and predict.smooth.spline from the modreg library. Four numerical methods are available:

- **"a"** apply smoothing to $K(r)$, estimate its derivative, and plug in to the formula above;
- **"b"** apply smoothing to $Y(r) = \frac{K(r)}{2\pi r}$ constraining $Y(0) = 0$, estimate the derivative of Y, and solve;

- **"c"** apply smoothing to $Z(r) = \frac{K(r)}{\pi r^2}$ constraining $Z(0) = 1$, estimate its derivative, and solve.
- **"d"** apply smoothing to $V(r) = \sqrt{K(r)}$, estimate its derivative, and solve.

Method `"c"` seems to be the best at suppressing variability for small values of r. However it effectively constrains $g(0) = 1$. If the point pattern seems to have inhibition at small distances, you may wish to experiment with method `"b"` which effectively constrains $g(0) = 0$. Method `"a"` seems comparatively unreliable.

Useful arguments to control the splines include the smoothing tradeoff parameter `spar` and the degrees of freedom `df`. See `smooth.spline` for details.

Value

A function array (object of class `"fasp"`, see `fasp.object`) representing an array of pair correlation functions. This can be thought of as a matrix `Y` each of whose entries `Y[i,j]` is a function value table (class `"fv"`) representing the pair correlation function between points of type `i` and points of type `j`.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

References

Stoyan, D, Kendall, W.S. and Mecke, J. (1995) *Stochastic geometry and its applications*. 2nd edition. Springer Verlag.

Stoyan, D. and Stoyan, H. (1994) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

See Also

`Kest`, `Kinhom`, `Kcross`, `Kdot`, `Kmulti`, `alltypes`, `smooth.spline`, `predict.smooth.spline`

Examples

```
# multitype point pattern
KK <- alltypes(amacrine, "K")
p <- pcf.fasp(KK, spar=0.5, method="b")
plot(p)
# strong inhibition between points of the same type
```

pcf.fv *Pair Correlation Function obtained from K Function*

Description

Estimates the pair correlation function of a point pattern, given an estimate of the K function.

Usage

```
  ## S3 method for class 'fv'
pcf(X, ..., method="c")
```

Arguments

X	An estimate of the K function or one of its variants. An object of class "fv".
...	Arguments controlling the smoothing spline function smooth.spline.
method	Letter "a", "b", "c" or "d" indicating the method for deriving the pair correlation function from the K function.

Details

The pair correlation function of a stationary point process is

$$g(r) = \frac{K'(r)}{2\pi r}$$

where $K'(r)$ is the derivative of $K(r)$, the reduced second moment function (aka "Ripley's K function") of the point process. See Kest for information about $K(r)$. For a stationary Poisson process, the pair correlation function is identically equal to 1. Values $g(r) < 1$ suggest inhibition between points; values greater than 1 suggest clustering.

We also apply the same definition to other variants of the classical K function, such as the multitype K functions (see Kcross, Kdot) and the inhomogeneous K function (see Kinhom). For all these variants, the benchmark value of $K(r) = \pi r^2$ corresponds to $g(r) = 1$.

This routine computes an estimate of $g(r)$ from an estimate of $K(r)$ or its variants, using smoothing splines to approximate the derivative. It is a method for the generic function pcf for the class "fv".

The argument X should be an estimated K function, given as a function value table (object of class "fv", see fv.object). This object should be the value returned by Kest, Kcross, Kmulti or Kinhom.

The smoothing spline operations are performed by smooth.spline and predict.smooth.spline from the modreg library. Four numerical methods are available:

- **"a"** apply smoothing to $K(r)$, estimate its derivative, and plug in to the formula above;
- **"b"** apply smoothing to $Y(r) = \frac{K(r)}{2\pi r}$ constraining $Y(0) = 0$, estimate the derivative of Y, and solve;
- **"c"** apply smoothing to $Z(r) = \frac{K(r)}{\pi r^2}$ constraining $Z(0) = 1$, estimate its derivative, and solve.

- **"d"** apply smoothing to $V(r) = \sqrt{K(r)}$, estimate its derivative, and solve.

Method "c" seems to be the best at suppressing variability for small values of r. However it effectively constrains $g(0) = 1$. If the point pattern seems to have inhibition at small distances, you may wish to experiment with method "b" which effectively constrains $g(0) = 0$. Method "a" seems comparatively unreliable.

Useful arguments to control the splines include the smoothing tradeoff parameter spar and the degrees of freedom df. See smooth.spline for details.

Value

A function value table (object of class "fv", see fv.object) representing a pair correlation function.

Essentially a data frame containing (at least) the variables

r	the vector of values of the argument r at which the pair correlation function $g(r)$ has been estimated
pcf	vector of values of $g(r)$

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

Stoyan, D, Kendall, W.S. and Mecke, J. (1995) *Stochastic geometry and its applications*. 2nd edition. Springer Verlag.

Stoyan, D. and Stoyan, H. (1994) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

See Also

pcf, pcf.ppp, Kest, Kinhom, Kcross, Kdot, Kmulti, alltypes, smooth.spline, predict.smooth.spline

Examples

```
# univariate point pattern
X <- simdat

K <- Kest(X)
p <- pcf.fv(K, spar=0.5, method="b")
plot(p, main="pair correlation function for simdat")
# indicates inhibition at distances r < 0.3
```

`pcf.ppp` *Pair Correlation Function of Point Pattern*

Description

Estimates the pair correlation function of a point pattern using kernel methods.

Usage

```
  ## S3 method for class 'ppp'
pcf(X, ..., r = NULL,
                    adaptive=FALSE,
                    kernel="epanechnikov", bw=NULL, h=NULL,
                    bw.args=list(), stoyan=0.15, adjust=1,
                    correction=c("translate", "Ripley"),
                    divisor = c("r", "d", "a", "t"),
                    zerocor=c("weighted", "reflection", "convolution",
                              "bdrykern", "JonesFoster", "none"),
                    gref = NULL,
                    tau = 0,
                    fast = TRUE,
                    var.approx = FALSE,
                    domain=NULL,
                    ratio=FALSE, close=NULL)
```

Arguments

`X`	A point pattern (object of class `"ppp"`).
`...`	Arguments passed to `density.default` or to `densityBC` controlling the kernel smoothing of the pair correlation estimate.
`r`	Optional. Vector of values for the argument r at which $g(r)$ should be evaluated. There is a sensible default.
`adaptive`	Logical value specifying whether to use adaptive kernel smoothing (`adaptive=TRUE`) or fixed-bandwidth kernel smoothing (`adaptive=FALSE`, the default).
`kernel`	Choice of smoothing kernel, passed to `density.default`.
`bw`	Bandwidth for smoothing kernel. Either a single numeric value giving the standard deviation of the kernel, or a character string specifying a bandwidth selection rule, or a function that computes the selected bandwidth. See Details.
`h`	Kernel halfwidth h (incompatible with argument `bw`). A numerical value. The parameter `h` is defined as the half-width of the support of the kernel, except for the Gaussian kernel where `h` is the standard deviation.
`bw.args`	Optional. List of additional arguments to be passed to `bw` when `bw` is a function. Alternatively, `bw` may be a function that should be applied to `X` to produce a list of additional arguments.
`stoyan`	Coefficient for Stoyan's bandwidth selection rule; see Details.

adjust	Numerical adjustment factor for the bandwidth. The bandwidth actually used is adjust * bw. This makes it easy to specify choices like 'half the selected bandwidth'.
correction	Edge correction. A character vector specifying the choice (or choices) of edge correction. See Details.
divisor	Choice of divisor in the estimation formula: either "r" (the default) or "d", or the new alternatives "a" or "t". See Details.
zerocor	String (partially matched) specifying a correction for the boundary effect bias at $r = 0$. Possible values are "none", "weighted", "convolution", "reflection", "bdrykern" and "JonesFoster". See Details, or help file for densityBC.
gref	Optional. A pair correlation function that will be used as the reference for the transformation to uniformity, when divisor="t". Either a function in the R language giving the pair correlation function, or a fitted model (object of class "kppm", "dppm", "ppm" or "slrm") or a theoretical point process model (object of class "zclustermodel" or "detpointprocfamily") for which the pair correlation function can be computed.
tau	Optional shrinkage coefficient. A single numeric value.
fast	Logical value specifying whether to compute the kernel smoothing using a Fast Fourier Transform algorithm (fast=TRUE) or an exact analytic kernel sum (fast=FALSE).
var.approx	Logical value indicating whether to compute an analytic approximation to the variance of the estimated pair correlation.
domain	Optional. Calculations will be restricted to this subset of the window. See Details.
ratio	Logical. If TRUE, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.
close	Advanced use only. Precomputed data. See section on Advanced Use.

Details

The pair correlation function $g(r)$ is a summary of the dependence between points in a spatial point process. The best intuitive interpretation is the following: the probability $p(r)$ of finding two points at locations x and y separated by a distance r is equal to

$$p(r) = \lambda^2 g(r)\,\mathrm{d}x\,\mathrm{d}y$$

where λ is the intensity of the point process. For a completely random (uniform Poisson) process, $p(r) = \lambda^2\,\mathrm{d}x\,\mathrm{d}y$ so $g(r) = 1$. Formally, the pair correlation function of a stationary point process is defined by

$$g(r) = \frac{K'(r)}{2\pi r}$$

where $K'(r)$ is the derivative of $K(r)$, the reduced second moment function (aka "Ripley's K function") of the point process. See Kest for information about $K(r)$.

For a stationary Poisson process, the pair correlation function is identically equal to 1. Values $g(r) < 1$ suggest inhibition between points; values greater than 1 suggest clustering.

This routine computes an estimate of $g(r)$ by kernel smoothing.

- If divisor="r" (the default), then the standard kernel estimator (Stoyan and Stoyan, 1994, pages 284–285) is used. By default, the recommendations of Stoyan and Stoyan (1994) are followed exactly.
- If divisor="d" then a modified estimator is used (Guan, 2007): the contribution from an interpoint distance d_{ij} to the estimate of $g(r)$ is divided by d_{ij} instead of dividing by r. This usually improves the bias of the estimator when r is close to zero.
- If divisor="a" then improved method of Baddeley, Davies and Hazelton (2025) is used. The distances d_{ij} are first converted to disc areas $a_{ij} = \pi d_{ij}^2$, and smoothing is performed on the area scale, then the result is back-transformed to the original scale.
- If divisor="t" then the distances d_{ij} are transformed to uniformity using the reference pair correlation function gref as described in Baddeley, Davies and Hazelton (2025).
- If divisor is a function in the R language, then it will be applied to the point pattern X and should return one of the strings "r", "d", "a" or "t" listed above. This option makes it possible to specify a rule which decides which estimator to use, based on the data.

There is also a choice of spatial edge corrections (which are needed to avoid bias due to edge effects associated with the boundary of the spatial window):

- If correction="translate" or correction="translation" then the translation correction is used. For divisor="r" the translation-corrected estimate is given in equation (15.15), page 284 of Stoyan and Stoyan (1994).
- If correction="Ripley" or correction="isotropic" then Ripley's isotropic edge correction is used. For divisor="r" the isotropic-corrected estimate is given in equation (15.18), page 285 of Stoyan and Stoyan (1994).
- If correction="none" then no edge correction is used, that is, an uncorrected estimate is computed.

Multiple corrections can be selected. The default is correction=c("translate", "Ripley").

Alternatively correction="all" selects all options; correction="best" selects the option which has the best statistical performance; correction="good" selects the option which is the best compromise between statistical performance and speed of computation.

Argument zerocor determines the correction to the one-dimensional kernel-smoothed estimate on the real number line, to correct bias close to the boundary $r = 0$. The argument zerocor is passed to densityBC. Options include:

- zerocor="none": no correction.
- zerocor="convolution": the convolution, uniform or renormalization kernel.
- zerocor="weighted": the cut-and-normalization method.
- zerocor="reflection": the reflection method.
- zerocor="bdrykern": the linear boundary kernel.
- zerocor="JonesFoster": the Jones-Foster modification of the linear boundary kernel.

The choice of smoothing kernel is controlled by the argument kernel which is passed to density.default. The default is the Epanechnikov kernel, recommended by Stoyan and Stoyan (1994, page 285).

The bandwidth of the smoothing kernel can be controlled by the argument bw. Bandwidth is defined as the standard deviation of the kernel; see the documentation for density.default. For the Epanechnikov kernel with half-width h, the argument bw is equivalent to $h/\sqrt{5}$.

Stoyan and Stoyan (1994, page 285) recommend using the Epanechnikov kernel with support $[-h, h]$ chosen by the rule of thumn $h = c/\sqrt{\lambda}$, where λ is the (estimated) intensity of the point process, and c is a constant in the range from 0.1 to 0.2. See equation (15.16). If bw is missing or NULL, then this rule of thumb will be applied. The argument stoyan determines the value of c. The smoothing bandwidth that was used in the calculation is returned as an attribute of the final result.

The argument bw can be

- missing or null. In this case, the default value for bw is "stoyan" when adaptive=FALSE and "bw.abram" when adaptive=TRUE.
- a single numeric value giving the bandwidth.
- a character string specifying a bandwidth selection rule. String names of rules applicable when adaptive=FALSE include "stoyan", "fiksel" and any rules recognised by density.default. String names applicable when adaptive=TRUE include "bw.abram" and "bw.pow".
- a function that computes the selected bandwidth.
  - If adaptive=FALSE, the function bw will be applied to the point pattern X to determine the bandwidth. Examples include bw.pcf and bw.stoyan. The function bw should accept the point pattern X as its first argument. Additional arguments to bw may be specified in the list bw.args. If bw recognises any of the arguments kernel, correction, divisor, zerocor and adaptive, then these arguments will be passed to bw as well. The function bw should return a single numeric value.
  - If adaptive=TRUE, the function bw will be applied to the vector of pairwise distances between data points (or the transformed distances if divisor="a" or divisor="t"). Examples include bw.abram.default and bw.pow. The function bw should accept the vector of pairwise distances as its first argument. Additional arguments to bw may be specified in the list bw.args.

Note that if bw.args is a function, it will be applied to the point pattern X to determine the list of arguments (whether adaptive is TRUE or FALSE).

The argument r is the vector of values for the distance r at which $g(r)$ should be evaluated. There is a sensible default. If it is specified, r must be a vector of increasing numbers starting from r[1] = 0, and max(r) must not exceed half the diameter of the window.

If the argument domain is given, estimation will be restricted to this region. That is, the estimate of $g(r)$ will be based on pairs of points in which the first point lies inside domain and the second point is unrestricted. The argument domain should be a window (object of class "owin") or something acceptable to as.owin. It must be a subset of the window of the point pattern X.

To compute a confidence band for the true value of the pair correlation function, use lohboot.

If var.approx = TRUE, the variance of the estimate of the pair correlation will also be calculated using an analytic approximation (Illian et al, 2008, page 234) which is valid for stationary point processes which are not too clustered. This calculation is not yet implemented when the argument domain is given.

If fast=TRUE, the calculation uses the Fast Fourier Transform to the maximum extent possible for the chosen boundary correction. If fast=FALSE (the default), the entire calculation uses analytic formulas written in C code.

Value

A function value table (object of class "fv"). Essentially a data frame containing the variables

r	the vector of values of the argument r at which the pair correlation function $g(r)$ has been estimated
theo	vector of values equal to 1, the theoretical value of $g(r)$ for the Poisson process
trans	vector of values of $g(r)$ estimated by translation correction
iso	vector of values of $g(r)$ estimated by Ripley isotropic correction
v	vector of approximate values of the variance of the estimate of $g(r)$

as required.

If ratio=TRUE then the return value also has two attributes called "numerator" and "denominator" which are "fv" objects containing the numerators and denominators of each estimate of $g(r)$.

The return value also has an attribute "bw" giving the smoothing bandwidth that was used, and an attribute "info" containing details of the algorithm parameters.

Advanced Use

To perform the same computation using several different bandwidths bw, it is efficient to use the argument close. This should be the result of closepairs(X, rmax) for a suitably large value of rmax, namely rmax >= max(r) + 3 * bw.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net>, Ege Rubak <rubak@math.aau.dk>, Martin Hazelton <Martin.Hazelton@otago.ac.nz> and Tilman Davies <Tilman.Davies@otago.ac.nz>.

References

Baddeley, A., Davies, T.M. and Hazelton, M.L. (2025) An improved estimator of the pair correlation function of a spatial point process. *Biometrika*, to appear.

Guan, Y. (2007) A least-squares cross-validation bandwidth selection approach in pair correlation function estimation. *Statistics and Probability Letters* **77** (18) 1722–1729.

Illian, J., Penttinen, A., Stoyan, H. and Stoyan, D. (2008) *Statistical Analysis and Modelling of Spatial Point Patterns.* Wiley.

Stoyan, D. and Stoyan, H. (1994) *Fractals, random shapes and point fields: methods of geometrical statistics.* John Wiley and Sons.

See Also

densityBC,

Kest, pcf, density.default, bw.stoyan, bw.pcf, lohboot.

Examples

```
pr <- pcf(redwood, divisor="r")
plot(pr, main="pair correlation function for redwoods")

# compare estimates
pd <- pcf(redwood, divisor="d")
pa <- pcf(redwood, divisor="a")

plot(pr, cbind(iso, theo) ~ r, col=c("red", "black"),
     ylim.covers=0, main="Estimates of PCF",
     lwd=c(2,1), lty=c(1,3), legend=FALSE)
plot(pd, iso ~ r, col="blue", lwd=2, add=TRUE)
plot(pa, iso ~ r, col="green", lwd=2, add=TRUE)
legend("center", col=c("red", "blue", "green"), lty=1, lwd=2,
       legend=c("divisor=r","divisor=d", "divisor=a"))
```

---

pcf3est *Pair Correlation Function of a Three-Dimensional Point Pattern*

Description

Estimates the pair correlation function from a three-dimensional point pattern.

Usage

```
pcf3est(X, ..., rmax = NULL, nrval = 128,
        correction = c("translation", "isotropic"),
        delta=NULL, adjust=1, biascorrect=TRUE)
```

Arguments

X	Three-dimensional point pattern (object of class "pp3").
...	Ignored.
rmax	Optional. Maximum value of argument r for which $g_3(r)$ will be estimated.
nrval	Optional. Number of values of r for which $g_3(r)$ will be estimated.
correction	Optional. Character vector specifying the edge correction(s) to be applied. See Details.
delta	Optional. Half-width of the Epanechnikov smoothing kernel.
adjust	Optional. Adjustment factor for the default value of delta.
biascorrect	Logical value. Whether to correct for underestimation due to truncation of the kernel near $r = 0$.

Details

For a stationary point process Φ in three-dimensional space, the pair correlation function is

$$g_3(r) = \frac{K_3'(r)}{4\pi r^2}$$

where K_3' is the derivative of the three-dimensional K-function (see `K3est`).

The three-dimensional point pattern `X` is assumed to be a partial realisation of a stationary point process Φ. The distance between each pair of distinct points is computed. Kernel smoothing is applied to these distance values (weighted by an edge correction factor) and the result is renormalised to give the estimate of $g_3(r)$.

The available edge corrections are:

`"translation"`: the Ohser translation correction estimator (Ohser, 1983; Baddeley et al, 1993)

`"isotropic"`: the three-dimensional counterpart of Ripley's isotropic edge correction (Ripley, 1977; Baddeley et al, 1993).

Kernel smoothing is performed using the Epanechnikov kernel with half-width `delta`. If `delta` is missing, the default is to use the rule-of-thumb $\delta = 0.26/\lambda^{1/3}$ where $\lambda = n/v$ is the estimated intensity, computed from the number n of data points and the volume v of the enclosing box. This default value of `delta` is multiplied by the factor `adjust`.

The smoothing estimate of the pair correlation $g_3(r)$ is typically an underestimate when r is small, due to truncation of the kernel at $r = 0$. If `biascorrect=TRUE`, the smoothed estimate is approximately adjusted for this bias. This is advisable whenever the dataset contains a sufficiently large number of points.

Value

A function value table (object of class `"fv"`) that can be plotted, printed or coerced to a data frame containing the function values.

Additionally the value of `delta` is returned as an attribute of this object.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rana Moyeed.

References

Baddeley, A.J, Moyeed, R.A., Howard, C.V. and Boyde, A. (1993) Analysis of a three-dimensional point pattern with replication. *Applied Statistics* **42**, 641–668.

Ohser, J. (1983) On estimators for the reduced second moment measure of point processes. *Mathematische Operationsforschung und Statistik, series Statistics*, **14**, 63 – 71.

Ripley, B.D. (1977) Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, Series B*, **39**, 172 – 212.

See Also

pp3 to create a three-dimensional point pattern (object of class "pp3").

F3est, G3est, K3est for other summary functions of a three-dimensional point pattern.

pcf to estimate the pair correlation function of point patterns in two dimensions or other spaces.

Examples

```
X <- rpoispp3(250)
Z <- pcf3est(X)
Zbias <- pcf3est(X, biascorrect=FALSE)
if(interactive()) {
  opa <- par(mfrow=c(1,2))
  plot(Z,     ylim.covers=c(0, 1.2))
  plot(Zbias, ylim.covers=c(0, 1.2))
  par(opa)
}
attr(Z, "delta")
```

---

pcfcross *Multitype pair correlation function (cross-type)*

Description

Calculates an estimate of the cross-type pair correlation function for a multitype point pattern.

Usage

```
pcfcross(X, i, j, ...,
         r = NULL,
         kernel = "epanechnikov", bw = NULL, stoyan = 0.15,
         correction = c("isotropic", "Ripley", "translate"),
         divisor = c("r", "d"),
         ratio = FALSE)
```

Arguments

X	The observed point pattern, from which an estimate of the cross-type pair correlation function $g_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor).
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of marks(X).
j	The type (mark value) of the points in X to which distances are measured. A character string (or something that will be converted to a character string). Defaults to the second level of marks(X).
...	Ignored.

r	Vector of values for the argument r at which $g(r)$ should be evaluated. There is a sensible default.
kernel	Choice of smoothing kernel, passed to density.default.
bw	Bandwidth for smoothing kernel, passed to density.default.
stoyan	Coefficient for default bandwidth rule; see Details.
correction	Choice of edge correction.
divisor	Choice of divisor in the estimation formula: either "r" (the default) or "d". See Details.
ratio	Logical. If TRUE, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.

Details

The cross-type pair correlation function is a generalisation of the pair correlation function pcf to multitype point patterns.

For two locations x and y separated by a distance r, the probability $p(r)$ of finding a point of type i at location x and a point of type j at location y is

$$p(r) = \lambda_i \lambda_j g_{i,j}(r)\, \mathrm{d}x\, \mathrm{d}y$$

where λ_i is the intensity of the points of type i. For a completely random Poisson marked point process, $p(r) = \lambda_i \lambda_j$ so $g_{i,j}(r) = 1$. Indeed for any marked point pattern in which the points of type i are independent of the points of type j, the theoretical value of the cross-type pair correlation is $g_{i,j}(r) = 1$.

For a stationary multitype point process, the cross-type pair correlation function between marks i and j is formally defined as

$$g_{i,j}(r) = \frac{K'_{i,j}(r)}{2\pi r}$$

where $K'_{i,j}$ is the derivative of the cross-type K function $K_{i,j}(r)$. of the point process. See Kest for information about $K(r)$.

The command pcfcross computes a kernel estimate of the cross-type pair correlation function between marks i and j.

- If divisor="r" (the default), then the multitype counterpart of the standard kernel estimator (Stoyan and Stoyan, 1994, pages 284–285) is used. By default, the recommendations of Stoyan and Stoyan (1994) are followed exactly.
- If divisor="d" then a modified estimator is used: the contribution from an interpoint distance d_{ij} to the estimate of $g(r)$ is divided by d_{ij} instead of dividing by r. This usually improves the bias of the estimator when r is close to zero.

There is also a choice of spatial edge corrections (which are needed to avoid bias due to edge effects associated with the boundary of the spatial window): correction="translate" is the Ohser-Stoyan translation correction, and correction="isotropic" or "Ripley" is Ripley's isotropic correction.

The choice of smoothing kernel is controlled by the argument kernel which is passed to density. The default is the Epanechnikov kernel.

The bandwidth of the smoothing kernel can be controlled by the argument bw. Its precise interpretation is explained in the documentation for density.default. For the Epanechnikov kernel with support $[-h, h]$, the argument bw is equivalent to $h/\sqrt{5}$.

If bw is not specified, the default bandwidth is determined by Stoyan's rule of thumb (Stoyan and Stoyan, 1994, page 285) applied to the points of type j. That is, $h = c/\sqrt{\lambda}$, where λ is the (estimated) intensity of the point process of type j, and c is a constant in the range from 0.1 to 0.2. The argument stoyan determines the value of c.

The companion function pcfdot computes the corresponding analogue of Kdot.

Value

An object of class "fv", see fv.object, which can be plotted directly using plot.fv.

Essentially a data frame containing columns

r	the vector of values of the argument r at which the function $g_{i,j}$ has been estimated
theo	the theoretical value $g_{i,j}(r) = 1$ for independent marks.

together with columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $g_{i,j}$ obtained by the edge corrections named.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

Mark connection function markconnect.

Multitype pair correlation pcfdot, pcfmulti.

Pair correlation pcf,pcf.ppp.

Kcross

Examples

```
p <- pcfcross(amacrine, "off", "on")
p <- pcfcross(amacrine, "off", "on", stoyan=0.1)
plot(p)
```

pcfcross.inhom *Inhomogeneous Multitype Pair Correlation Function (Cross-Type)*

Description

Estimates the inhomogeneous cross-type pair correlation function for a multitype point pattern.

Usage

```
pcfcross.inhom(X, i, j, lambdaI = NULL, lambdaJ = NULL, ...,
               r = NULL, breaks = NULL,
               kernel="epanechnikov", bw=NULL, adjust.bw = 1, stoyan=0.15,
               correction = c("isotropic", "Ripley", "translate"),
               sigma = NULL, adjust.sigma = 1, varcov = NULL)
```

Arguments

X	The observed point pattern, from which an estimate of the inhomogeneous cross-type pair correlation function $g_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor).
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of marks(X).
j	The type (mark value) of the points in X to which distances are measured. A character string (or something that will be converted to a character string). Defaults to the second level of marks(X).
lambdaI	Optional. Values of the estimated intensity function of the points of type i. Either a vector giving the intensity values at the points of type i, a pixel image (object of class "im") giving the intensity values at all locations, or a function(x,y) which can be evaluated to give the intensity value at any location.
lambdaJ	Optional. Values of the estimated intensity function of the points of type j. A numeric vector, pixel image or function(x,y).
r	Vector of values for the argument r at which $g_{ij}(r)$ should be evaluated. There is a sensible default.
breaks	This argument is for internal use only.
kernel	Choice of one-dimensional smoothing kernel, passed to density.default.
bw	Bandwidth for one-dimensional smoothing kernel, passed to density.default.
adjust.bw	Numeric value. bw will be multiplied by this value.
...	Other arguments passed to the one-dimensional kernel density estimation function density.default.
stoyan	Bandwidth coefficient; see Details.
correction	Choice of edge correction.
sigma, varcov	Optional arguments passed to density.ppp to control the smoothing bandwidth, when lambdaI or lambdaJ is estimated by spatial kernel smoothing.
adjust.sigma	Numeric value. sigma will be multiplied by this value.

Details

The inhomogeneous cross-type pair correlation function $g_{ij}(r)$ is a summary of the dependence between two types of points in a multitype spatial point process that does not have a uniform density of points.

The best intuitive interpretation is the following: the probability $p(r)$ of finding two points, of types i and j respectively, at locations x and y separated by a distance r is equal to

$$p(r) = \lambda_i(x) lambda_j(y) g(r)\, \mathrm{d}x\, \mathrm{d}y$$

where λ_i is the intensity function of the process of points of type i. For a multitype Poisson point process, this probability is $p(r) = \lambda_i(x)\lambda_j(y)$ so $g_{ij}(r) = 1$.

The command `pcfcross.inhom` estimates the inhomogeneous pair correlation using a modified version of the algorithm in `pcf.ppp`. The arguments `bw` and `adjust.bw` control the degree of one-dimensional smoothing of the estimate of pair correlation.

If the arguments `lambdaI` and/or `lambdaJ` are missing or null, they will be estimated from `X` by spatial kernel smoothing using a leave-one-out estimator, computed by `density.ppp`. The arguments `sigma`, `varcov` and `adjust.sigma` control the degree of spatial smoothing.

Value

A function value table (object of class `"fv"`). Essentially a data frame containing the variables

`r`	the vector of values of the argument r at which the inhomogeneous cross-type pair correlation function $g_{ij}(r)$ has been estimated
`theo`	vector of values equal to 1, the theoretical value of $g_{ij}(r)$ for the Poisson process
`trans`	vector of values of $g_{ij}(r)$ estimated by translation correction
`iso`	vector of values of $g_{ij}(r)$ estimated by Ripley isotropic correction

as required.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

`pcf.ppp`, `pcfinhom`, `pcfcross`, `pcfdot.inhom`

Examples

```
plot(pcfcross.inhom(amacrine, "on", "off", stoyan=0.1),
     legendpos="bottom")
```

pcfdot *Multitype pair correlation function (i-to-any)*

Description

Calculates an estimate of the multitype pair correlation function (from points of type `i` to points of any type) for a multitype point pattern.

Usage

```
pcfdot(X, i, ..., r = NULL,
       kernel = "epanechnikov", bw = NULL, stoyan = 0.15,
       correction = c("isotropic", "Ripley", "translate"),
       divisor = c("r", "d"),
       ratio=FALSE)
```

Arguments

`X`	The observed point pattern, from which an estimate of the dot-type pair correlation function $g_{i\bullet}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor).
`i`	The type (mark value) of the points in `X` from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of `marks(X)`.
`...`	Ignored.
`r`	Vector of values for the argument r at which $g(r)$ should be evaluated. There is a sensible default.
`kernel`	Choice of smoothing kernel, passed to `density.default`.
`bw`	Bandwidth for smoothing kernel, passed to `density.default`.
`stoyan`	Coefficient for default bandwidth rule; see Details.
`correction`	Choice of edge correction.
`divisor`	Choice of divisor in the estimation formula: either `"r"` (the default) or `"d"`. See Details.
`ratio`	Logical. If `TRUE`, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.

Details

This is a generalisation of the pair correlation function `pcf` to multitype point patterns.

For two locations x and y separated by a nonzero distance r, the probability $p(r)$ of finding a point of type i at location x and a point of any type at location y is

$$p(r) = \lambda_i \lambda g_{i\bullet}(r)\, \mathrm{d}x\, \mathrm{d}y$$

where λ is the intensity of all points, and λ_i is the intensity of the points of type i. For a completely random Poisson marked point process, $p(r) = \lambda_i \lambda$ so $g_{i\bullet}(r) = 1$.

For a stationary multitype point process, the type-i-to-any-type pair correlation function between marks i and j is formally defined as

$$g_{i\bullet}(r) = \frac{K'_{i\bullet}(r)}{2\pi r}$$

where $K'_{i\bullet}$ is the derivative of the type-i-to-any-type K function $K_{i\bullet}(r)$. of the point process. See Kdot for information about $K_{i\bullet}(r)$.

The command pcfdot computes a kernel estimate of the multitype pair correlation function from points of type i to points of any type.

- If divisor="r" (the default), then the multitype counterpart of the standard kernel estimator (Stoyan and Stoyan, 1994, pages 284–285) is used. By default, the recommendations of Stoyan and Stoyan (1994) are followed exactly.
- If divisor="d" then a modified estimator is used: the contribution from an interpoint distance d_{ij} to the estimate of $g(r)$ is divided by d_{ij} instead of dividing by r. This usually improves the bias of the estimator when r is close to zero.

There is also a choice of spatial edge corrections (which are needed to avoid bias due to edge effects associated with the boundary of the spatial window): correction="translate" is the Ohser-Stoyan translation correction, and correction="isotropic" or "Ripley" is Ripley's isotropic correction.

The choice of smoothing kernel is controlled by the argument kernel which is passed to density. The default is the Epanechnikov kernel.

The bandwidth of the smoothing kernel can be controlled by the argument bw. Its precise interpretation is explained in the documentation for density.default. For the Epanechnikov kernel with support $[-h, h]$, the argument bw is equivalent to $h/\sqrt{5}$.

If bw is not specified, the default bandwidth is determined by Stoyan's rule of thumb (Stoyan and Stoyan, 1994, page 285). That is, $h = c/\sqrt{\lambda}$, where λ is the (estimated) intensity of the unmarked point process, and c is a constant in the range from 0.1 to 0.2. The argument stoyan determines the value of c.

The companion function pcfcross computes the corresponding analogue of Kcross.

Value

An object of class "fv", see fv.object, which can be plotted directly using plot.fv.

Essentially a data frame containing columns

r	the vector of values of the argument r at which the function $g_{i\bullet}$ has been estimated
theo	the theoretical value $g_{i\bullet}(r) = 1$ for independent marks.

together with columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $g_{i,j}$ obtained by the edge corrections named.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

Mark connection function markconnect.

Multitype pair correlation pcfcross, pcfmulti.

Pair correlation pcf,pcf.ppp.

Kdot

Examples

```
p <- pcfdot(amacrine, "on")
p <- pcfdot(amacrine, "on", stoyan=0.1)
plot(p)
```

---

pcfdot.inhom *Inhomogeneous Multitype Pair Correlation Function (Type-i-To-Any-Type)*

Description

Estimates the inhomogeneous multitype pair correlation function (from type i to any type) for a multitype point pattern.

Usage

```
pcfdot.inhom(X, i, lambdaI = NULL, lambdadot = NULL, ...,
                r = NULL, breaks = NULL,
                kernel="epanechnikov", bw=NULL, adjust.bw=1, stoyan=0.15,
                correction = c("isotropic", "Ripley", "translate"),
                sigma = NULL, adjust.sigma = 1, varcov = NULL)
```

Arguments

X	The observed point pattern, from which an estimate of the inhomogeneous multitype pair correlation function $g_{i\bullet}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor).
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of marks(X).
lambdaI	Optional. Values of the estimated intensity function of the points of type i. Either a vector giving the intensity values at the points of type i, a pixel image (object of class "im") giving the intensity values at all locations, or a function(x,y) which can be evaluated to give the intensity value at any location.

lambdadot	Optional. Values of the estimated intensity function of the point pattern X. A numeric vector, pixel image or function(x,y).
r	Vector of values for the argument r at which $g_{i\bullet}(r)$ should be evaluated. There is a sensible default.
breaks	This argument is for internal use only.
kernel	Choice of one-dimensional smoothing kernel, passed to density.default.
bw	Bandwidth for one-dimensional smoothing kernel, passed to density.default.
adjust.bw	Numeric value. bw will be multiplied by this value.
...	Other arguments passed to the one-dimensional kernel density estimation function density.default.
stoyan	Bandwidth coefficient; see Details.
correction	Choice of edge correction.
sigma, varcov	Optional arguments passed to density.ppp to control the smoothing bandwidth, when lambdaI and/or lambdadot is estimated by spatial kernel smoothing.
adjust.sigma	Numeric value. sigma will be multiplied by this value.

Details

The inhomogeneous multitype (type i to any type) pair correlation function $g_{i\bullet}(r)$ is a summary of the dependence between different types of points in a multitype spatial point process that does not have a uniform density of points.

The best intuitive interpretation is the following: the probability $p(r)$ of finding a point of type i at location x and another point of any type at location y, where x and y are separated by a distance r, is equal to

$$p(r) = \lambda_i(x) lambda(y) g(r)\,\mathrm{d}x\,\mathrm{d}y$$

where λ_i is the intensity function of the process of points of type i, and where λ is the intensity function of the points of all types. For a multitype Poisson point process, this probability is $p(r) = \lambda_i(x)\lambda(y)$ so $g_{i\bullet}(r) = 1$.

The command pcfdot.inhom estimates the inhomogeneous multitype pair correlation using a modified version of the algorithm in pcf.ppp. The arguments bw and adjust.bw control the degree of one-dimensional smoothing of the estimate of pair correlation.

If the arguments lambdaI and/or lambdadot are missing or null, they will be estimated from X by spatial kernel smoothing using a leave-one-out estimator, computed by density.ppp. The arguments sigma, varcov and adjust.sigma control the degree of spatial smoothing.

Value

A function value table (object of class "fv"). Essentially a data frame containing the variables

r	the vector of values of the argument r at which the inhomogeneous multitype pair correlation function $g_{i\bullet}(r)$ has been estimated
theo	vector of values equal to 1, the theoretical value of $g_{i\bullet}(r)$ for the Poisson process
trans	vector of values of $g_{i\bullet}(r)$ estimated by translation correction
iso	vector of values of $g_{i\bullet}(r)$ estimated by Ripley isotropic correction

as required.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

pcf.ppp, pcfinhom, pcfdot, pcfcross.inhom

Examples

```
plot(pcfdot.inhom(amacrine, "on", stoyan=0.1), legendpos="bottom")
```

---

pcfinhom *Inhomogeneous Pair Correlation Function*

Description

Estimates the inhomogeneous pair correlation function of a point pattern using kernel methods.

Usage

```
pcfinhom(X, lambda = NULL, ...,
            r = NULL, adaptive = FALSE,
            kernel = "epanechnikov", bw = NULL, h = NULL,
            bw.args = list(), stoyan = 0.15, adjust = 1,
            correction = c("translate", "Ripley"),
            divisor = c("r", "d", "a", "t"),
            zerocor=c("weighted", "reflection", "convolution",
                      "bdrykern", "JonesFoster", "none"),
            renormalise = TRUE, normpower = 1,
            update = TRUE, leaveoneout = TRUE,
            reciplambda = NULL, sigma = NULL, adjust.sigma = 1, varcov = NULL,
            gref = NULL, tau = 0, fast = TRUE, var.approx = FALSE,
            domain = NULL, ratio = FALSE, close = NULL)
```

Arguments

X	A point pattern (object of class "ppp").
lambda	Optional. Values of the estimated intensity function. Either a vector giving the intensity values at the points of the pattern X, a pixel image (object of class "im") giving the intensity values at all locations, a fitted point process model (object of class "ppm", "kppm" or "dppm") or a function(x,y) which can be evaluated to give the intensity value at any location.
...	Arguments passed to density.default or to densityBC controlling the kernel smoothing.
r	Vector of values for the argument r at which $g(r)$ should be evaluated. There is a sensible default.

adaptive	Logical value specifying whether to use adaptive kernel smoothing (adaptive=TRUE) or fixed-bandwidth kernel smoothing (adaptive=FALSE, the default).
kernel	Choice of smoothing kernel, passed to density.default.
bw	Bandwidth for smoothing kernel. Either a single numeric value giving the standard deviation of the kernel, or a character string specifying a bandwidth selection rule, or a function that computes the selected bandwidth. See Details.
h	Kernel halfwidth h (incompatible with argument bw). A numerical value. The parameter h is defined as the half-width of the support of the kernel, except for the Gaussian kernel where h is the standard deviation.
bw.args	Optional. List of additional arguments to be passed to bw when bw is a function. Alternatively, bw may be a function that should be applied to X to produce a list of additional arguments.
stoyan	Coefficient for Stoyan's bandwidth selection rule; see Details.
adjust	Numerical adjustment factor for the bandwidth. The bandwidth actually used is adjust * bw. This makes it easy to specify choices like 'half the selected bandwidth'.
correction	Edge correction. A character vector specifying the choice (or choices) of edge correction. See Details.
divisor	Choice of divisor in the estimation formula: either "r" (the default) or "d", or the new alternatives "a" or "t". See Details.
zerocor	String (partially matched) specifying a correction for the boundary effect bias at $r = 0$. Possible values are "none", "weighted", "convolution", "reflection", "bdrykern" and "JonesFoster". See Details, or help file for densityBC.
renormalise	Logical. Whether to renormalise the estimate. See Details.
normpower	Integer (usually either 1 or 2). Normalisation power. See Details.
update	Logical. If lambda is a fitted model (class "ppm", "kppm" or "dppm") and update=TRUE (the default), the model will first be refitted to the data X (using update.ppm or update.kppm) before the fitted intensity is computed. If update=FALSE, the fitted intensity of the model will be computed without refitting it to X.
leaveoneout	Logical value (passed to density.ppp or fitted.ppm) specifying whether to use a leave-one-out rule when calculating the intensity.
reciplambda	Alternative to lambda. Values of the estimated *reciprocal* $1/\lambda$ of the intensity function. Either a vector giving the reciprocal intensity values at the points of the pattern X, a pixel image (object of class "im") giving the reciprocal intensity values at all locations, or a function(x,y) which can be evaluated to give the reciprocal intensity value at any location.
sigma, varcov	Optional arguments passed to density.ppp to control the smoothing bandwidth, when lambda is estimated by kernel smoothing.
adjust.sigma	Numeric value. sigma will be multiplied by this value.
gref	Optional. A pair correlation function that will be used as the reference for the transformation to uniformity, when divisor="t". Either a function in the R language giving the pair correlation function, or a fitted model (object of

	class "kppm", "dppm", "ppm" or "slrm") or a theoretical point process model (object of class "zclustermodel" or "detpointprocfamily") for which the pair correlation function can be computed.
tau	Optional shrinkage coefficient. A single numeric value.
fast	Logical value specifying whether to compute the kernel smoothing using a Fast Fourier Transform algorithm (fast=TRUE) or an exact analytic kernel sum (fast=FALSE).
var.approx	Logical value indicating whether to compute an analytic approximation to the variance of the estimated pair correlation.
domain	Optional. Calculations will be restricted to this subset of the window. See Details.
ratio	Logical. If TRUE, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.
close	Advanced use only. Precomputed data. See section on Advanced Use.

Details

The inhomogeneous pair correlation function $g_{\mathrm{inhom}}(r)$ is a summary of the dependence between points in a spatial point process that does not have a uniform density of points.

The best intuitive interpretation is the following: the probability $p(r)$ of finding two points at locations x and y separated by a distance r is equal to

$$p(r) = \lambda(x) lambda(y) g(r)\, \mathrm{d}x\, \mathrm{d}y$$

where λ is the intensity function of the point process. For a Poisson point process with intensity function λ, this probability is $p(r) = \lambda(x)\lambda(y)$ so $g_{\mathrm{inhom}}(r) = 1$.

The inhomogeneous pair correlation function is related to the inhomogeneous K function through

$$g_{\mathrm{inhom}}(r) = \frac{K'_{\mathrm{inhom}}(r)}{2\pi r}$$

where $K'_{\mathrm{inhom}}(r)$ is the derivative of $K_{\mathrm{inhom}}(r)$, the inhomogeneous K function. See Kinhom for information about $K_{\mathrm{inhom}}(r)$.

The command pcfinhom estimates the inhomogeneous pair correlation using a modified version of the algorithm in pcf. In this modified version, the contribution from each pair of points $X[i], X[j]$ is weighted by $1/(\lambda(X[i])\lambda(X[j]))$. The arguments divisor, correction and zerocor are interpreted as described in the help file for pcf.

If renormalise=TRUE (the default), then the estimates are multiplied by $c^{\mathrm{normpower}}$ where $c = \mathrm{area}(W)/\sum(1/\lambda(x_i))$. This rescaling reduces the variability and bias of the estimate in small samples and in cases of very strong inhomogeneity. The default value of normpower is 1 but the most sensible value is 2, which would correspond to rescaling the lambda values so that $\sum(1/\lambda(x_i)) = \mathrm{area}(W)$.

Value

A function value table (object of class "fv"). Essentially a data frame containing the variables

r	the vector of values of the argument r at which the pair correlation function $g(r)$ has been estimated

theo	vector of values equal to 1, the theoretical value of $g(r)$ for the Poisson process
trans	vector of values of $g(r)$ estimated by translation correction
iso	vector of values of $g(r)$ estimated by Ripley isotropic correction
v	vector of approximate values of the variance of the estimate of $g(r)$

as required.

If ratio=TRUE then the return value also has two attributes called "numerator" and "denominator" which are "fv" objects containing the numerators and denominators of each estimate of $g(r)$.

The return value also has an attribute "bw" giving the smoothing bandwidth that was used, and an attribute "info" containing details of the algorithm parameters.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net>, Ege Rubak <rubak@math.aau.dk>, Martin Hazelton <Martin.Hazelton@otago.ac.nz> and Tilman Davies <Tilman.Davies@otago.ac.nz>.

References

Baddeley, A., Davies, T.M. and Hazelton, M.L. (2025) An improved estimator of the pair correlation function of a spatial point process. *Biometrika*, to appear.

See Also

pcf, bw.bdh, bw.pcfinhom

Examples

```
g <- pcfinhom(japanesepines, divisor="a")
```

---

pcfmulti *Marked pair correlation function*

Description

For a marked point pattern, estimate the multitype pair correlation function using kernel methods.

Usage

```
pcfmulti(X, I, J, ..., r = NULL,
         kernel = "epanechnikov", bw = NULL, stoyan = 0.15,
         correction = c("translate", "Ripley"),
         divisor = c("r", "d"),
         Iname = "points satisfying condition I",
         Jname = "points satisfying condition J",
         ratio = FALSE)
```

Arguments

X	The observed point pattern, from which an estimate of the cross-type pair correlation function $g_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor).
I	Subset index specifying the points of X from which distances are measured.
J	Subset index specifying the points in X to which distances are measured.
...	Ignored.
r	Vector of values for the argument r at which $g(r)$ should be evaluated. There is a sensible default.
kernel	Choice of smoothing kernel, passed to density.default.
bw	Bandwidth for smoothing kernel, passed to density.default.
stoyan	Coefficient for default bandwidth rule.
correction	Choice of edge correction.
divisor	Choice of divisor in the estimation formula: either "r" (the default) or "d".
Iname, Jname	Optional. Character strings describing the members of the subsets I and J.
ratio	Logical. If TRUE, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.

Details

This is a generalisation of pcfcross to arbitrary collections of points.

The algorithm measures the distance from each data point in subset I to each data point in subset J, excluding identical pairs of points. The distances are kernel-smoothed and renormalised to form a pair correlation function.

- If divisor="r" (the default), then the multitype counterpart of the standard kernel estimator (Stoyan and Stoyan, 1994, pages 284–285) is used. By default, the recommendations of Stoyan and Stoyan (1994) are followed exactly.
- If divisor="d" then a modified estimator is used: the contribution from an interpoint distance d_{ij} to the estimate of $g(r)$ is divided by d_{ij} instead of dividing by r. This usually improves the bias of the estimator when r is close to zero.

There is also a choice of spatial edge corrections (which are needed to avoid bias due to edge effects associated with the boundary of the spatial window): correction="translate" is the Ohser-Stoyan translation correction, and correction="isotropic" or "Ripley" is Ripley's isotropic correction.

The arguments I and J specify two subsets of the point pattern X. They may be any type of subset indices, for example, logical vectors of length equal to npoints(X), or integer vectors with entries in the range 1 to npoints(X), or negative integer vectors.

Alternatively, I and J may be **functions** that will be applied to the point pattern X to obtain index vectors. If I is a function, then evaluating I(X) should yield a valid subset index. This option is useful when generating simulation envelopes using envelope.

The choice of smoothing kernel is controlled by the argument kernel which is passed to density. The default is the Epanechnikov kernel.

The bandwidth of the smoothing kernel can be controlled by the argument bw. Its precise interpretation is explained in the documentation for density.default. For the Epanechnikov kernel with support $[-h, h]$, the argument bw is equivalent to $h/\sqrt{5}$.

If bw is not specified, the default bandwidth is determined by Stoyan's rule of thumb (Stoyan and Stoyan, 1994, page 285) applied to the points of type j. That is, $h = c/\sqrt{\lambda}$, where λ is the (estimated) intensity of the point process of type j, and c is a constant in the range from 0.1 to 0.2. The argument stoyan determines the value of c.

Value

An object of class "fv".

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

pcfcross, pcfdot, pcf.ppp.

Examples

```
adult <- (marks(longleaf) >= 30)
juvenile <- !adult
p <- pcfmulti(longleaf, adult, juvenile)
```

---

plot.bermantest *Plot Result of Berman Test*

Description

Plot the result of Berman's test of goodness-of-fit

Usage

```
## S3 method for class 'bermantest'
plot(x, ...,
                  lwd=par("lwd"), col=par("col"), lty=par("lty"),
                  lwd0=lwd, col0=2, lty0=2)
```

Arguments

x	Object to be plotted. An object of class "bermantest" produced by berman.test.
...	extra arguments that will be passed to the plotting function plot.ecdf.
col, lwd, lty	The width, colour and type of lines used to plot the empirical distribution curve.
col0, lwd0, lty0	The width, colour and type of lines used to plot the predicted (null) distribution curve.

Details

This is the plot method for the class "bermantest". An object of this class represents the outcome of Berman's test of goodness-of-fit of a spatial Poisson point process model, computed by berman.test.

For the *Z1* test (i.e. if x was computed using berman.test(,which="Z1")), the plot displays the two cumulative distribution functions that are compared by the test: namely the empirical cumulative distribution function of the covariate at the data points, $\hat{F}$, and the predicted cumulative distribution function of the covariate under the model, F_0, both plotted against the value of the covariate. Two vertical lines show the mean values of these two distributions. If the model is correct, the two curves should be close; the test is based on comparing the two vertical lines.

For the *Z2* test (i.e. if x was computed using berman.test(,which="Z2")), the plot displays the empirical cumulative distribution function of the values $U_i = F_0(Y_i)$ where Y_i is the value of the covariate at the i-th data point. The diagonal line with equation $y = x$ is also shown. Two vertical lines show the mean of the values U_i and the value $1/2$. If the model is correct, the two curves should be close. The test is based on comparing the two vertical lines.

Value

NULL.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

berman.test

Examples

```
plot(berman.test(cells, "x"))

if(require("spatstat.model")) {
  # synthetic data: nonuniform Poisson process
  X <- rpoispp(function(x,y) { 100 * exp(-x) }, win=square(1))

  # fit uniform Poisson process
  fit0 <- ppm(X ~1)

  # test covariate = x coordinate
  xcoord <- function(x,y) { x }

  # test wrong model
  k <- berman.test(fit0, xcoord, "Z1")

  # plot result of test
  plot(k, col="red", col0="green")

  # Z2 test
```

```
  k2 <- berman.test(fit0, xcoord, "Z2")
  plot(k2, col="red", col0="green")
}
```

`plot.cdftest` *Plot a Spatial Distribution Test*

Description

Plot the result of a spatial distribution test computed by `cdf.test`.

Usage

```
## S3 method for class 'cdftest'
plot(x, ...,
                   style=c("cdf", "PP", "QQ"),
                   lwd=par("lwd"), col=par("col"), lty=par("lty"),
                   lwd0=lwd, col0=2, lty0=2,
                   do.legend)
```

Arguments

`x`	Object to be plotted. An object of class `"cdftest"` produced by a method for `cdf.test`.
`...`	extra arguments that will be passed to the plotting function `plot.default`.
`style`	Style of plot. See Details.
`col`, `lwd`, `lty`	The width, colour and type of lines used to plot the empirical curve (the empirical distribution, or PP plot or QQ plot).
`col0`, `lwd0`, `lty0`	The width, colour and type of lines used to plot the reference curve (the predicted distribution, or the diagonal).
`do.legend`	Logical value indicating whether to add an explanatory legend. Applies only when `style="cdf"`.

Details

This is the `plot` method for the class `"cdftest"`. An object of this class represents the outcome of a spatial distribution test, computed by `cdf.test`, and based on either the Kolmogorov-Smirnov, Cramér-von Mises or Anderson-Darling test.

If `style="cdf"` (the default), the plot displays the two cumulative distribution functions that are compared by the test: namely the empirical cumulative distribution function of the covariate at the data points, and the predicted cumulative distribution function of the covariate under the model, both plotted against the value of the covariate. The Kolmogorov-Smirnov test statistic (for example) is the maximum vertical separation between the two curves.

If `style="PP"` then the P-P plot is drawn. The x coordinates of the plot are cumulative probabilities for the covariate under the model. The y coordinates are cumulative probabilities for the covariate

at the data points. The diagonal line $y = x$ is also drawn for reference. The Kolmogorov-Smirnov test statistic is the maximum vertical separation between the P-P plot and the diagonal reference line.

If `style="QQ"` then the Q-Q plot is drawn. The x coordinates of the plot are quantiles of the covariate under the model. The y coordinates are quantiles of the covariate at the data points. The diagonal line $y = x$ is also drawn for reference. The Kolmogorov-Smirnov test statistic cannot be read off the Q-Q plot.

Value

NULL.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

cdf.test

Examples

```
op <- options(useFancyQuotes=FALSE)

plot(cdf.test(cells, "x"))

if(require("spatstat.model")) {
  # synthetic data: nonuniform Poisson process
  X <- rpoispp(function(x,y) { 100 * exp(x) }, win=square(1))

  # fit uniform Poisson process
  fit0 <- ppm(X ~1)

  # test covariate = x coordinate
  xcoord <- function(x,y) { x }

  # test wrong model
  k <- cdf.test(fit0, xcoord)

  # plot result of test
  plot(k, lwd0=3)

  plot(k, style="PP")

  plot(k, style="QQ")
}

options(op)
```

plot.envelope *Plot a Simulation Envelope*

Description

Plot method for the class "envelope".

Usage

```
 ## S3 method for class 'envelope'
plot(x, ..., main)
```

Arguments

x	An object of class "envelope", containing the variables to be plotted or variables from which the plotting coordinates can be computed.
main	Main title for plot.
...	Extra arguments passed to plot.fv.

Details

This is the plot method for the class "envelope" of simulation envelopes. Objects of this class are created by the command envelope.

This plot method is currently identical to plot.fv.

Its default behaviour is to shade the region between the upper and lower envelopes in a light grey colour. To suppress the shading and plot the upper and lower envelopes as curves, set shade=NULL. To change the colour of the shading, use the argument shadecol which is passed to plot.fv.

See plot.fv for further information on how to control the plot.

Value

Either NULL, or a data frame giving the meaning of the different line types and colours.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

envelope, plot.fv

Examples

```
E <- envelope(cells, Kest, nsim=19)
plot(E)
plot(E, sqrt(./pi) ~ r)
```

plot.fasp *Plot a Function Array*

Description

Plots an array of summary functions, usually associated with a point pattern, stored in an object of class "fasp". A method for plot.

Usage

```
   ## S3 method for class 'fasp'
plot(x,formule=NULL, ...,
                        subset=NULL, title=NULL, banner=TRUE,
                        transpose=FALSE,
                        samex=FALSE, samey=FALSE,
                        mar.panel=NULL,
                        outerlabels=TRUE, cex.outerlabels=1.25,
                        legend=FALSE)
```

Arguments

x	An object of class "fasp" representing a function array.
formule	A formula or list of formulae indicating what variables are to be plotted against what variable. Each formula is either an R language formula object, or a string that can be parsed as a formula. If formule is a list, its k^{th} component should be applicable to the $(i,j)^{th}$ plot where x$which[i,j]=k. If the formula is left as NULL, then plot.fasp attempts to use the component default.formula of x. If that component is NULL as well, it gives up.
...	Arguments passed to plot.fv to control the individual plot panels.
subset	A logical vector, or a vector of indices, or an expression or a character string, or a **list** of such, indicating a subset of the data to be included in each plot. If subset is a list, its k^{th} component should be applicable to the $(i,j)^{th}$ plot where x$which[i,j]=k.
title	Overall title for the plot.
banner	Logical. If TRUE, the overall title is plotted. If FALSE, the overall title is not plotted and no space is allocated for it.
transpose	Logical. If TRUE, rows and columns will be exchanged.
samex, samey	Logical values indicating whether all individual plot panels should have the same x axis limits and the same y axis limits, respectively. This makes it easier to compare the plots.
mar.panel	Vector of length 4 giving the value of the graphics parameter mar controlling the size of plot margins for each individual plot panel. See par.
outerlabels	Logical. If TRUE, the row and column names of the array of functions are plotted in the margins of the array of plot panels. If FALSE, each individual plot panel is labelled by its row and column name.

`cex.outerlabels` Character expansion factor for row and column labels of array.

`legend` Logical flag determining whether to plot a legend in each panel.

Details

An object of class `"fasp"` represents an array of summary functions, usually associated with a point pattern. See `fasp.object` for details. Such an object is created, for example, by `alltypes`.

The function `plot.fasp` is a method for `plot`. It calls `plot.fv` to plot the individual panels.

For information about the interpretation of the arguments `formule` and `subset`, see `plot.fv`.

Arguments that are often passed through `...` include `col` to control the colours of the different lines in a panel, and `lty` and `lwd` to control the line type and line width of the different lines in a panel. The argument `shade` can also be used to display confidence intervals or significance bands as filled grey shading. See `plot.fv`.

The argument `title`, if present, will determine the overall title of the plot. If it is absent, it defaults to `x$title`. Titles for the individual plot panels will be taken from `x$titles`.

Value

None.

Warnings

(Each component of) the `subset` argument may be a logical vector (of the same length as the vectors of data which are extracted from `x`), or a vector of indices, or an **expression** such as `expression(r<=0.2)`, or a text string, such as `"r<=0.2"`.

Attempting a syntax such as `subset = r<=0.2` (without wrapping `r<=0.2` either in quote marks or in `expression()`) will cause this function to fall over.

Variables referred to in any formula must exist in the data frames stored in `x`. What the names of these variables are will of course depend upon the nature of `x`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

`alltypes`, `plot.fv`, `fasp.object`

Examples

```
if(interactive()) {
X.G <- alltypes(amacrine,"G")
plot(X.G)
plot(X.G,subset="r<=0.2")
plot(X.G,formule=asin(sqrt(cbind(km,theo))) ~ asin(sqrt(theo)))
plot(X.G,fo=cbind(km,theo) - theo~r, subset="theo<=0.9")
}
```

plot.fv *Plot Function Values*

Description

Plot method for the class "fv".

Usage

```
 ## S3 method for class 'fv'
plot(x, fmla, ..., subset=NULL, lty=NULL, col=NULL, lwd=NULL,
           xlim=NULL, ylim=NULL, xlab=NULL, ylab=NULL,
           clip.xlim=TRUE, ylim.covers=NULL,
           legend=!add, legendpos="topleft", legendavoid=missing(legendpos),
           legendmath=TRUE, legendargs=list(),
           shade=fvnames(x, ".s"), shadecol="grey",
           add=FALSE, log="",
           mathfont=c("italic", "plain", "bold", "bolditalic"),
           limitsonly=FALSE, do.plot=TRUE)
```

Arguments

x	An object of class "fv", containing the variables to be plotted or variables from which the plotting coordinates can be computed.
fmla	an R language formula determining which variables or expressions are plotted. Either a formula object, or a string that can be parsed as a formula. See Details.
subset	(optional) subset of rows of the data frame that will be plotted.
lty	(optional) numeric vector of values of the graphical parameter lty controlling the line style of each plot.
col	(optional) numeric vector of values of the graphical parameter col controlling the colour of each plot.
lwd	(optional) numeric vector of values of the graphical parameter lwd controlling the line width of each plot.
xlim	(optional) range of x axis
ylim	(optional) range of y axis
xlab	(optional) label for x axis
ylab	(optional) label for y axis
...	Extra arguments passed to plot.default.
clip.xlim	Logical value specifying whether the range of the horizontal axis xlim should be automatically restricted to a subset of the range of the available data. See the section on **Controlling the horizontal axis limits** below.
ylim.covers	Optional vector of y values that must be included in the y axis. For example ylim.covers=0 will ensure that the y axis includes the origin.

legend	Logical flag or NULL. If legend=TRUE, the algorithm plots a legend in the top left corner of the plot, explaining the meaning of the different line types and colours.
legendpos	The position of the legend. Either a character string keyword (see legend for keyword options) or a pair of coordinates in the format list(x,y). Alternatively if legendpos="float", a location will be selected inside the plot region, avoiding the graphics.
legendavoid	Whether to avoid collisions between the legend and the graphics. Logical value. If TRUE, the code will check for collisions between the legend box and the graphics, and will override legendpos if a collision occurs. If FALSE, the value of legendpos is always respected.
legendmath	Logical. If TRUE, the legend will display the mathematical notation for each curve. If FALSE, the legend text is the identifier (column name) for each curve.
legendargs	Named list containing additional arguments to be passed to legend controlling the appearance of the legend.
shade	A character vector giving the names of two columns of x, or another type of index that identifies two columns. When the corresponding curves are plotted, the region between the curves will be shaded in light grey. The object x may or may not contain two columns which are designated as boundaries for shading; they are identified by fvnames(x, ".s"). The default is to shade between these two curves if they exist. To suppress this behaviour, set shade=NULL.
shadecol	The colour to be used in the shade plot. A character string or an integer specifying a colour.
add	Logical. Whether the plot should be added to an existing plot
log	A character string which contains "x" if the x axis is to be logarithmic, "y" if the y axis is to be logarithmic and "xy" or "yx" if both axes are to be logarithmic.
mathfont	Character string. The font to be used for mathematical expressions in the axis labels and the legend.
limitsonly	Logical. If FALSE, plotting is performed normally. If TRUE, no plotting is performed at all; just the x and y limits of the plot are computed and returned.
do.plot	Logical value indicating whether to actually plot the graph of x. Setting do.plot=FALSE will simply return the data frame giving the meaning of the different line types and colours, without plotting them.

Details

This is the plot method for the class "fv".

An object of class "fv" is a convenient way of storing several different statistical estimates of a summary function; see fv.object. The default behaviour, executed by plot(x), displays these different estimates as curves with different colours and line styles, and plots a legend explaining them.

The use of the argument fmla is like plot.formula, but offers some extra functionality.

The left and right hand sides of fmla are evaluated, and the results are plotted against each other (the left side on the y axis against the right side on the x axis).

The left and right hand sides of fmla may be the names of columns of the data frame x, or expressions involving these names. If a variable in fmla is not the name of a column of x, the algorithm

will search for an object of this name in the environment where `plot.fv` was called, and then in the enclosing environment, and so on.

Multiple curves may be specified by a single formula of the form `cbind(y1,y2,...,yn) ~ x`, where `x,y1,y2,...,yn` are expressions involving the variables in the data frame. Each of the variables `y1,y2,...,yn` in turn will be plotted against `x`. See the examples.

Convenient abbreviations which can be used in the formula are

- the symbol `.` which represents all the columns in the data frame that will be plotted by default;
- the symbol `.x` which represents the function argument;
- the symbol `.y` which represents the recommended value of the function.

For further information, see `fvnames`.

The value returned by this plot function indicates the meaning of the line types and colours in the plot. It can be used to make a suitable legend for the plot if you want to do this by hand. See the examples.

The argument `shade` can be used to display critical bands or confidence intervals. If it is not `NULL`, then it should be a subset index for the columns of `x`, that identifies exactly 2 columns. When the corresponding curves are plotted, the region between the curves will be shaded in light grey. See the Examples.

The default values of `lty`, `col` and `lwd` can be changed using `spatstat.options("plot.fv")`.

Use `type = "n"` to create the plot region and draw the axes without plotting any curves.

Use `do.plot=FALSE` to suppress all plotting. The return value is a data frame giving the meaning of the different line types and colours which would have been plotted.

Use `limitsonly=TRUE` to suppress all plotting and just compute the x and y limits. This can be used to calculate common x and y scales for several plots.

To change the kind of parenthesis enclosing the explanatory text about the unit of length, use `spatstat.options('units.paren')`

Value

Invisible: either `NULL`, or a data frame giving the meaning of the different line types and colours.

Controlling the horizontal axis limits

The plot generated by `plot(x)` does not necessarily display all the data that is contained in the object. The range of values of the function argument r displayed in the plot may be narrower than the range of values actually contained in the data frame.

To override this behaviour and display all the available data, set `clip.xlim=FALSE`.

Statistical literature for summary functions of spatial data recommends that, when the function is plotted, the values of the function argument on the horizontal axis should be restricted to a limited range of values. For example, Ripley recommends that the K-function $K(r)$ should be plotted only for values of distance r between 0 and $b/4$ where b is the shortest side of the enclosing rectangle of the data.

This may be desirable so that the interesting detail is clearly visible in the plot. It may be necessary because values outside the recommended range are theoretically invalid, or unreliable due to high variance or large bias.

To support this standard practice, each object of class "fv" may include data specifying a "recommended range" of values of the function argument. The object produced by Kest includes a recommended range following Ripley's recommendation above. Similarly for Gest, Fest and many other commands.

When plot(x) is executed, the horizontal axis is restricted to the recommended range of values. This recommendation can be overridden by setting clip.xlim=FALSE or by specifying the numerical limits xlim.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

fv.object, Kest

Examples

```
K <- Kest(cells)
# K is an object of class "fv"

plot(K, iso ~ r)               # plots iso against r

plot(K, sqrt(iso/pi) ~ r)   # plots sqrt(iso/r)  against r

plot(K, cbind(iso,theo) ~ r)   # plots iso against r  AND theo against r

plot(K, .  ~ r)            # plots all available estimates of K against r

plot(K, sqrt(./pi) ~ r)   # plots all estimates of L-function
                          # L(r) = sqrt(K(r)/pi)

plot(K, cbind(iso,theo) ~ r, col=c(2,3))
                                # plots iso against r  in colour 2
                                # and theo against r in colour 3

plot(K, iso ~ r, subset=quote(r < 0.2))
                                # plots iso against r for r < 10

# Can't remember the names of the columns? No problem..
plot(K, sqrt(./pi) ~ .x)

# making a legend by hand
v <- plot(K, . ~ r, legend=FALSE)
legend("topleft", legend=v$meaning, lty=v$lty, col=v$col)

# significance bands
KE <- envelope(cells, Kest, nsim=19)
plot(KE, shade=c("hi", "lo"))

# how to display two functions on a common scale
Kr <- Kest(redwood)
```

```
a <- plot(K, limitsonly=TRUE)
b <- plot(Kr, limitsonly=TRUE)
xlim <- range(a$xlim, b$xlim)
ylim <- range(a$ylim, b$ylim)
opa <- par(mfrow=c(1,2))
plot(K, xlim=xlim, ylim=ylim)
plot(Kr, xlim=xlim, ylim=ylim)
par(opa)
# For a shortcut, try plot(anylist(K, Kr), equal.scales=TRUE)
```

`plot.laslett` *Plot Laslett Transform*

Description

Plot the result of Laslett's Transform.

Usage

```
## S3 method for class 'laslett'
plot(x, ...,
          Xpars = list(box = TRUE, col = "grey"),
          pointpars = list(pch = 3, cols = "blue"),
          rectpars = list(lty = 3, border = "green"))
```

Arguments

`x`	Object of class `"laslett"` produced by `laslett` representing the result of Laslett's transform.
`...`	Additional plot arguments passed to `plot.solist`.
`Xpars`	A list of plot arguments passed to `plot.owin` or `plot.im` to display the original region `X` before transformation.
`pointpars`	A list of plot arguments passed to `plot.ppp` to display the tangent points.
`rectpars`	A list of plot arguments passed to `plot.owin` to display the maximal rectangle.

Details

This is the `plot` method for the class `"laslett"`.

The function `laslett` applies Laslett's Transform to a spatial region `X` and returns an object of class `"laslett"` representing the result of the transformation. The result is plotted by this method.

The plot function `plot.solist` is used to align the before-and-after pictures. See `plot.solist` for further options to control the plot.

Value

None.

Author(s)

Kassel Hingee and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

See Also

laslett

Examples

```
b <- laslett(heather$coarse, plotit=FALSE)
plot(b, main="Heather Data")
```

---

plot.quadrattest *Display the result of a quadrat counting test.*

Description

Given the result of a quadrat counting test, graphically display the quadrats that were used, the observed and expected counts, and the residual in each quadrat.

Usage

```
 ## S3 method for class 'quadrattest'
plot(x, ..., textargs=list())
```

Arguments

x	Object of class "quadrattest" containing the result of quadrat.test.
...	Additional arguments passed to plot.tess to control the display of the quadrats.
textargs	List of additional arguments passed to text.default to control the appearance of the text.

Details

This is the plot method for objects of class "quadrattest". Such an object is produced by quadrat.test and represents the result of a χ^2 test for a spatial point pattern.

The quadrats are first plotted using plot.tess. Then in each quadrat, the observed and expected counts and the Pearson residual are displayed as text using text.default. Observed count is displayed at top left; expected count at top right; and Pearson residual at bottom.

Value

Null.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

quadrat.test, plot.tess, text.default, plot.quadratcount

Examples

```
plot(quadrat.test(swedishpines, 3))
```

---

plot.scan.test *Plot Result of Scan Test*

Description

Computes or plots an image showing the likelihood ratio test statistic for the scan test, or the optimal circle radius.

Usage

```
 ## S3 method for class 'scan.test'
plot(x, ..., what=c("statistic", "radius"),
               do.window = TRUE)

 ## S3 method for class 'scan.test'
as.im(X, ..., what=c("statistic", "radius"))
```

Arguments

x, X	Result of a scan test. An object of class "scan.test" produced by scan.test.
...	Arguments passed to plot.im to control the appearance of the plot.
what	Character string indicating whether to produce an image of the (profile) likelihood ratio test statistic (what="statistic", the default) or an image of the optimal value of circle radius (what="radius").
do.window	Logical value indicating whether to plot the original window of the data as well.

Details

These functions extract, and plot, the spatially-varying value of the likelihood ratio test statistic which forms the basis of the scan test.

If the test result X was based on circles of the same radius r, then as.im(X) is a pixel image of the likelihood ratio test statistic as a function of the position of the centre of the circle.

If the test result X was based on circles of several different radii r, then as.im(X) is a pixel image of the profile (maximum value over all radii r) likelihood ratio test statistic as a function of the position of the centre of the circle, and as.im(X, what="radius") is a pixel image giving for each location u the value of r which maximised the likelihood ratio test statistic at that location.

The plot method plots the corresponding image.

Value

The value of `as.im.scan.test` is a pixel image (object of class `"im"`). The value of `plot.scan.test` is `NULL`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

`scan.test`, `scanLRTS`

Examples

```
online <- interactive()
Nsim <- if(online) 19 else 2
r <- if(online) seq(0.04, 0.1, by=0.01) else c(0.05, 0.1)
a <- scan.test(redwood, r=r, method="poisson", nsim=Nsim)
plot(a)
as.im(a)
plot(a, what="radius")
```

---

plot.ssf *Plot a Spatially Sampled Function*

Description

Plot a spatially sampled function object.

Usage

```
## S3 method for class 'ssf'
plot(x, ...,
                  how = c("smoothed", "nearest", "points"),
                  style = c("image", "contour", "imagecontour"),
                  sigma = NULL, contourargs=list())

## S3 method for class 'ssf'
image(x, ...)

## S3 method for class 'ssf'
contour(x, ..., main, sigma = NULL)
```

Arguments

x	Spatially sampled function (object of class "ssf").
...	Arguments passed to image.default or plot.ppp to control the plot.
how	Character string determining whether to display the function values at the data points (how="points"), a smoothed interpolation of the function (how="smoothed"), or the function value at the nearest data point (how="nearest").
style	Character string indicating whether to plot the smoothed function as a colour image, a contour map, or both.
contourargs	Arguments passed to contour.default to control the contours, if style="contour" or style="imagecontour".
sigma	Smoothing bandwidth for smooth interpolation.
main	Optional main title for the plot.

Details

These are methods for the generic plot, image and contour for the class "ssf".

An object of class "ssf" represents a function (real- or vector-valued) that has been sampled at a finite set of points.

For plot.ssf there are three types of display. If how="points" the exact function values will be displayed as circles centred at the locations where they were computed. If how="smoothed" (the default) these values will be kernel-smoothed using Smooth.ppp and displayed as a pixel image. If how="nearest" the values will be interpolated by nearest neighbour interpolation using nnmark and displayed as a pixel image.

For image.ssf and contour.ssf the values are kernel-smoothed before being displayed.

Value

NULL.

Author(s)

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

References

Baddeley, A. (2017) Local composite likelihood for spatial point processes. *Spatial Statistics* **22**, 261–295.

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. Chapman and Hall/CRC Press.

See Also

ssf

Examples

```
a <- ssf(cells, nndist(cells, k=1:3))
plot(a, how="points")
plot(a, how="smoothed")
plot(a, how="nearest")
```

`plot.studpermutest` *Plot a Studentised Permutation Test*

Description

Plot the result of the studentised permutation test.

Usage

```
  ## S3 method for class 'studpermutest'
plot(x, fmla, ...,
          lty = NULL, col = NULL, lwd = NULL,
          lty.theo = NULL, col.theo = NULL, lwd.theo = NULL,
          lwd.mean = if (meanonly) 1 else NULL,
          lty.mean = lty, col.mean = col,
          separately = FALSE, meanonly = FALSE,
          main = if (meanonly) "group means" else NULL,
          xlim = NULL, ylim = NULL, ylab = NULL,
          legend = !add, legendpos = "topleft", lbox = FALSE, add = FALSE)
```

Arguments

`x`	An object of class `"studpermutest"` generated by `studpermu.test` and representing the result of a studentised permutation test for spatial point pattern data.
`fmla`	Plot formula used in `plot.fv`.
`...`	Additional graphical arguments passed to `plot.fv`.
`lty, col, lwd`	Line type, colour, and line width of the curves plotting the summary function for each point pattern in the original data. Either a single value or a vector of length equal to the number of point patterns.
`lty.theo, col.theo, lwd.theo`	Line type, colour, and line width of the curve representing the theoretical value of the summary function.
`lty.mean, col.mean, lwd.mean`	Line type, colour, and line width (as a multiple of `lwd`) of the curve representing the group mean of the summary function.
`separately`	Logical value indicating whether to plot each group of data in a separate panel.
`meanonly`	Logical value indicating whether to plot only the group means of the summary function.

main	Character string giving a main title for the plot.
xlim, ylim	Numeric vectors of length 2 giving the limits for the x and y coordinates of the plot or plots.
ylab	Character string or expression to be used for the label on the y axis.
legend	Logical value indicating whether to plot a legend explaining the meaning of each curve.
legendpos	Position of legend. See plot.fv.
lbox	Logical value indicating whether to plot a box around the plot.
add	Logical value indicating whether the plot should be added to the existing plot (add=TRUE) or whether a new frame should be created (add=FALSE, the default).

Details

This is the plot method for objects of class "studpermutest" which represent the result of a studentised permutation test applied to several point patterns. The test is performed by studpermu.test.

The plot shows the summary functions for each point pattern, coloured according to group. Optionally it can show the different groups in separate plot panels, or show only the group means in a single panel.

Value

Null.

Author(s)

Ute Hahn.

Modified for spatstat by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

studpermu.test

Examples

```
np <- if(interactive()) 99 else 19
testpyramidal <- studpermu.test(pyramidal, Neurons ~ group, nperm=np)
plot(testpyramidal)
plot(testpyramidal, meanonly=TRUE)
plot(testpyramidal, col.theo=8, lwd.theo=4, lty.theo=1)
plot(testpyramidal, . ~ pi * r^2)
op <- par(mfrow=c(1,3))
plot(testpyramidal, separately=TRUE)
plot(testpyramidal, separately=TRUE, col=2, lty=1, lwd.mean=2, col.mean=4)
par(op)
```

pool *Pool Data*

Description

Pool the data from several objects of the same class.

Usage

```
pool(...)
```

Arguments

...	Objects of the same type.

Details

The function pool is generic. There are methods for several classes, listed below.

pool is used to combine the data from several objects of the same type, and to compute statistics based on the combined dataset. It may be used to pool the estimates obtained from replicated datasets. It may also be used in high-performance computing applications, when the objects ... have been computed on different processors or in different batch runs, and we wish to combine them.

Value

An object of the same class as the arguments ....

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

pool.envelope, pool.fasp, pool.rat, pool.fv

pool.anylist *Pool Data from a List of Objects*

Description

Pool the data from the objects in a list.

Usage

```
## S3 method for class 'anylist'
pool(x, ...)
```

Arguments

x	A list, belonging to the class "anylist", containing objects that can be pooled.
...	Optional additional objects which can be pooled with the elements of x.

Details

The function pool is generic. Its purpose is to combine data from several objects of the same type (typically computed from different datasets) into a common, pooled estimate.

The function pool.anyist is the method for the class "anylist". It is used when the objects to be pooled are given in a list x.

Each of the elements of the list x, and each of the subsequent arguments ... if provided, must be an object of the same class.

Value

An object of the same class as each of the entries in x.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

anylist, pool.

Examples

```
Keach <- anylapply(waterstriders, Kest, ratio=TRUE, correction="iso")
K <- pool(Keach)
```

`pool.envelope` *Pool Data from Several Envelopes*

Description

Pool the simulation data from several simulation envelopes (objects of class `"envelope"`) and compute a new envelope.

Usage

```
## S3 method for class 'envelope'
pool(..., savefuns=FALSE, savepatterns=FALSE)
```

Arguments

`...`	Objects of class `"envelope"`.
`savefuns`	Logical flag indicating whether to save all the simulated function values.
`savepatterns`	Logical flag indicating whether to save all the simulated point patterns.

Details

The function `pool` is generic. This is the method for the class `"envelope"` of simulation envelopes. It is used to combine the simulation data from several simulation envelopes and to compute an envelope based on the combined data.

Each of the arguments `...` must be an object of class `"envelope"`. These envelopes must be compatible, in that they are envelopes for the same function, and were computed using the same options.

- In normal use, each envelope object will have been created by running the command `envelope` with the argument `savefuns=TRUE`. This ensures that each object contains the simulated data (summary function values for the simulated point patterns) that were used to construct the envelope.

  The simulated data are extracted from each object and combined. A new envelope is computed from the combined set of simulations.
- Alternatively, if each envelope object was created by running `envelope` with `VARIANCE=TRUE`, then the saved functions are not required.

  The sample means and sample variances from each envelope will be pooled. A new envelope is computed from the pooled mean and variance.

Warnings or errors will be issued if the envelope objects `...` appear to be incompatible. Apart from these basic checks, the code is not smart enough to decide whether it is sensible to pool the data.

To modify the envelope parameters or the type of envelope that is computed, first pool the envelope data using `pool.envelope`, then use `envelope.envelope` to modify the envelope parameters.

Value

An object of class `"envelope"`.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

envelope, envelope.envelope, pool, pool.fasp

Examples

```
E1 <- envelope(cells, Kest, nsim=10, savefuns=TRUE)
E2 <- envelope(cells, Kest, nsim=20, savefuns=TRUE)
pool(E1, E2)

V1 <- envelope(E1, VARIANCE=TRUE)
V2 <- envelope(E2, VARIANCE=TRUE)
pool(V1, V2)
```

---

pool.fasp *Pool Data from Several Function Arrays*

Description

Pool the simulation data from several function arrays (objects of class "fasp") and compute a new function array.

Usage

```
## S3 method for class 'fasp'
pool(...)
```

Arguments

...	Objects of class "fasp".

Details

The function pool is generic. This is the method for the class "fasp" of function arrays. It is used to combine the simulation data from several arrays of simulation envelopes and to compute a new array of envelopes based on the combined data.

Each of the arguments ... must be a function array (object of class "fasp") containing simulation envelopes. This is typically created by running the command alltypes with the arguments envelope=TRUE and savefuns=TRUE. This ensures that each object is an array of simulation envelopes, and that each envelope contains the simulated data (summary function values) that were used to construct the envelope.

The simulated data are extracted from each object and combined. A new array of envelopes is computed from the combined set of simulations.

Warnings or errors will be issued if the objects ... appear to be incompatible. However, the code is not smart enough to decide whether it is sensible to pool the data.

Value

An object of class "fasp".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

fasp, alltypes, pool.envelope, pool

Examples

```
A1 <- alltypes(amacrine,"K",nsim=9,envelope=TRUE,savefuns=TRUE)
A2 <- alltypes(amacrine,"K",nsim=10,envelope=TRUE,savefuns=TRUE)
pool(A1, A2)
```

---

pool.fv *Pool Several Functions*

Description

Combine several summary functions into a single function.

Usage

```
## S3 method for class 'fv'
pool(..., weights=NULL, relabel=TRUE, variance=TRUE)
```

Arguments

...	Objects of class "fv".
weights	Optional numeric vector of weights for the functions.
relabel	Logical value indicating whether the columns of the resulting function should be labelled to show that they were obtained by pooling.
variance	Logical value indicating whether to compute the sample variance and related terms.

Details

The function pool is generic. This is the method for the class "fv" of summary functions. It is used to combine several estimates of the same function into a single function.

Each of the arguments ... must be an object of class "fv". They must be compatible, in that they are estimates of the same function, and were computed using the same options.

The sample mean and sample variance of the corresponding estimates will be computed.

Value

An object of class "fv".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

pool, pool.anylist, pool.rat

Examples

```
K <- lapply(waterstriders, Kest, correction="iso")
Kall <- pool(K[[1]], K[[2]], K[[3]])
Kall <- pool(as.anylist(K))
plot(Kall, cbind(pooliso, pooltheo) ~ r,
           shade=c("loiso", "hiiso"),
           main="Pooled K function of waterstriders")
```

---

pool.quadrattest *Pool Several Quadrat Tests*

Description

Pool several quadrat tests into a single quadrat test.

Usage

```
## S3 method for class 'quadrattest'
pool(..., df=NULL, df.est=NULL, nsim=1999,
                                Xname=NULL, CR=NULL)
```

Arguments

...	Any number of objects, each of which is a quadrat test (object of class "quadrattest").
df	Optional. Number of degrees of freedom of the test statistic. Relevant only for χ^2 tests. Incompatible with df.est.
df.est	Optional. The number of fitted parameters, or the number of degrees of freedom lost by estimation of parameters. Relevant only for χ^2 tests. Incompatible with df.
nsim	Number of simulations, for Monte Carlo test.
Xname	Optional. Name of the original data.
CR	Optional. Numeric value of the Cressie-Read exponent CR overriding the value used in the tests.

Details

The function pool is generic. This is the method for the class "quadrattest".

An object of class "quadrattest" represents a χ^2 test or Monte Carlo test of goodness-of-fit for a point process model, based on quadrat counts. Such objects are created by the command quadrat.test.

Each of the arguments ... must be an object of class "quadrattest". They must all be the same type of test (chi-squared test or Monte Carlo test, conditional or unconditional) and must all have the same type of alternative hypothesis.

The test statistic of the pooled test is the Pearson X^2 statistic taken over all cells (quadrats) of all tests. The p value of the pooled test is then computed using either a Monte Carlo test or a χ^2 test.

For a pooled χ^2 test, the number of degrees of freedom of the combined test is computed by adding the degrees of freedom of all the tests (equivalent to assuming the tests are independent) unless it is determined by the arguments df or df.est. The resulting p value is computed to obtain the pooled test.

For a pooled Monte Carlo test, new simulations are performed to determine the pooled Monte Carlo p value.

Value

Another object of class "quadrattest".

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

pool, quadrat.test

Examples

```
Y <- split(humberside)
test1 <- quadrat.test(Y[[1]])
test2 <- quadrat.test(Y[[2]])
pool(test1, test2, Xname="Humberside")
```

---

pool.rat *Pool Data from Several Ratio Objects*

Description

Pool the data from several ratio objects (objects of class "rat") and compute a pooled estimate.

Usage

```
## S3 method for class 'rat'
pool(..., weights=NULL, relabel=TRUE, variance=TRUE)
```

Arguments

`...`	Objects of class `"rat"`.
`weights`	Numeric vector of weights.
`relabel`	Logical value indicating whether the result should be relabelled to show that it was obtained by pooling.
`variance`	Logical value indicating whether to compute the sample variance and related terms.

Details

The function `pool` is generic. This is the method for the class `"rat"` of ratio objects. It is used to combine several estimates of the same quantity when each estimate is a ratio.

Each of the arguments `...` must be an object of class `"rat"` representing a ratio object (basically a numerator and a denominator; see `rat`). We assume that these ratios are all estimates of the same quantity.

If the objects are called $R_1, \ldots, R_n$ and if R_i has numerator Y_i and denominator X_i, so that notionally $R_i = Y_i/X_i$, then the pooled estimate is the ratio-of-sums estimator

$$R = \frac{\sum_i Y_i}{\sum_i X_i}.$$

The standard error of R is computed using the delta method as described in Baddeley *et al.* (1993) or Cochran (1977, pp 154, 161).

If the argument `weights` is given, it should be a numeric vector of length equal to the number of objects to be pooled. The pooled estimator is the ratio-of-sums estimator

$$R = \frac{\sum_i w_i Y_i}{\sum_i w_i X_i}$$

where w_i is the `i`th weight.

This calculation is implemented only for certain classes of objects where the arithmetic can be performed.

This calculation is currently implemented only for objects which also belong to the class `"fv"` (function value tables). For example, if `Kest` is called with argument `ratio=TRUE`, the result is a suitable object (belonging to the classes `"rat"` and `"fv"`).

Warnings or errors will be issued if the ratio objects `...` appear to be incompatible. However, the code is not smart enough to decide whether it is sensible to pool the data.

Value

An object of the same class as the input.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A.J, Moyeed, R.A., Howard, C.V. and Boyde, A. (1993) Analysis of a three-dimensional point pattern with replication. *Applied Statistics* **42**, 641–668.

Cochran, W.G. (1977) *Sampling techniques*, 3rd edition. New York: John Wiley and Sons.

See Also

rat, pool, pool.fv, Kest

Examples

```
K1 <- Kest(runifpoint(42), ratio=TRUE, correction="iso")
K2 <- Kest(runifpoint(42), ratio=TRUE, correction="iso")
K3 <- Kest(runifpoint(42), ratio=TRUE, correction="iso")
K <- pool(K1, K2, K3)
plot(K, pooliso ~ r, shade=c("hiiso", "loiso"))
```

PPversion *Transform a Function into its P-P or Q-Q Version*

Description

Given a function object f containing both the estimated and theoretical versions of a summary function, these operations combine the estimated and theoretical functions into a new function. When plotted, the new function gives either the P-P plot or Q-Q plot of the original f.

Usage

```
PPversion(f, theo = "theo", columns = ".")

QQversion(f, theo = "theo", columns = ".")
```

Arguments

f	The function to be transformed. An object of class "fv".
theo	The name of the column of f that should be treated as the theoretical value of the function.
columns	Character vector, specifying the columns of f to which the transformation will be applied. Either a vector of names of columns of f, or one of the abbreviations recognised by fvnames.

Details

The argument f should be an object of class "fv", containing both empirical estimates $\widehat{f}(r)$ and a theoretical value $f_0(r)$ for a summary function.

The *P–P version* of f is the function $g(x) = \widehat{f}(f_0^{-1}(x))$ where f_0^{-1} is the inverse function of f_0. A plot of $g(x)$ against x is equivalent to a plot of $\widehat{f}(r)$ against $f_0(r)$ for all r. If f is a cumulative distribution function (such as the result of Fest or Gest) then this is a P–P plot, a plot of the observed versus theoretical probabilities for the distribution. The diagonal line $y = x$ corresponds to perfect agreement between observed and theoretical distribution.

The *Q–Q version* of f is the function $h(x) = f_0^{-1}(\widehat{f}(x))$. If f is a cumulative distribution function, a plot of $h(x)$ against x is a Q–Q plot, a plot of the observed versus theoretical quantiles of the distribution. The diagonal line $y = x$ corresponds to perfect agreement between observed and theoretical distribution. Another straight line corresponds to the situation where the observed variable is a linear transformation of the theoretical variable. For a point pattern X, the Q–Q version of Kest(X) is essentially equivalent to Lest(X).

Value

Another object of class "fv".

Author(s)

Tom Lawrence and Adrian Baddeley.

Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

plot.fv

Examples

```
opa <- par(mar=0.1+c(5,5,4,2))
G <- Gest(redwoodfull)
plot(PPversion(G))
plot(QQversion(G))
par(opa)
```

---

ptwise.envelope *Pointwise Statistics on an Envelope Object*

Description

Compute pointwise statistics from the simulated function values in an envelope object.

Usage

```
ptwise.envelope(object,
                stats=c("mean", "median", "bias",
                        "var", "sd", "se", "mse", "rmse",
                        "confint", "predint"),
                ...,
                level=0.95, transform=NULL, theo=NULL)

bias.envelope(object, theo, CI=TRUE, level=0.95)

RMSE.envelope(object, theo)
```

Arguments

object	An object of class "envelope" generated by the function envelope
stats	Summary statistic(s) to be calculated. A character string or character vector (partially matched) selected from the options given, or a function(x) provided by the user to compute the summary statistic.
...	Arguments passed to stats if it is a function.
level	Confidence level required for confint or predint. A probability value between 0 and 1.
transform	Optional expression (passed to with.fv) which should be applied to the function values before the summary statistics are calculated.
theo	Function in the R language that evaluates the true (theoretically expected) value of the spatial summary function. This is required if stats includes "bias", "mse" or "rmse".
CI	Logical value specifying whether to calculate confidence interval as well as bias.

Details

These functions compute pointwise summary statistics from n spatial summary functions which were obtained from n simulated point patterns.

The object should have been generated by the function envelope with the argument savefuns=TRUE specified.

The function envelope is normally used to generate simulation envelopes for a particular spatial summary function, such as the K function, by simulating n realisations of Complete Spatial Randomness or another model. However, when envelope is called with the argument savefuns=TRUE, it returns all the individual summary functions for the n simulated point patterns.

These individual functions are extracted by ptwise.envelope which then computes the desired summary statistics. The argument stats specifies the desired summary statistics. It can be a character string, or vector of character strings, containing any of the following (partially matched):

mean the pointwise sample mean of the functions

median the pointwise sample median of the functions

bias the pointwise bias of the functions

var the pointwise sample variance of the functions

sd the pointwise sample standard deviation of the functions

se the standard error of the pointwise sample mean

mse the pointwise mean squared error

rmse the pointwise root-mean-squared error

confint a confidence interval for the true mean

predint a prediction interval for the function value

For `confint` or `predint` the argument `level` specifies the confidence level.

Alternatively, the argument `stats` can be a user-specified function in the R language, which computes the summary statistic. It should accept a vector argument and return a single numerical value.

The result is an object of class `"fv"` that can be plotted directly. See the Examples for different styles of plot.

The functions `bias.envelope` and `RMSE.envelope` are wrappers for `ptwise.envelope` which calculate the bias and root-mean-square error respectively.

Value

A function value table (object of class `"fv"`) containing some or all of the following columns

`r`	Distance argument r
`mean`	Pointwise sample mean
`median`	Pointwise sample median
`bias`	Pointwise estimated bias
`var`	Pointwise sample variance
`sd`	Pointwise sample standard deviation
`se`	Pointwise standard error of pointwise mean
`mse`	Pointwise estimated mean squared error
`rmse`	Pointwise estimated root-mean-squared error
`loci`, `hici`	Pointwise confidence interval for the true mean
`lopi`, `hipi`	Pointwise prediction interval for the function value

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>, Tilman Davies <Tilman.Davies@otago.ac.nz> and Martin Hazelton <Martin.Hazelton@otago.ac.nz>.

See Also

`ISB.envelope`, `IV.envelope`, `ISE.envelope`, `MISE.envelope`.

Examples

```
E <- envelope(cells, Kest, nsim=20, savefuns=TRUE)
plot(ptwise.envelope(E, c("mean", "confint"), level=0.9))
plot(ptwise.envelope(E, max, na.rm=TRUE))

## statistics for L function
plot(ptwise.envelope(E, c("mean","confint"),
     transform=quote(sqrt(./pi)), level=0.9))

## calculate pointwise bias and RMSE
## using the theoretical true value K(r) = pi * r^2
trueK <- function(r) { pi * r^2 }
plot(bias.envelope(E, theo=trueK))
plot(RMSE.envelope(E, theo=trueK))
```

---

quadrat.test *Dispersion Test for Spatial Point Pattern Based on Quadrat Counts*

Description

Performs a test of Complete Spatial Randomness for a given point pattern, based on quadrat counts. Alternatively performs a goodness-of-fit test of a fitted inhomogeneous Poisson model. By default performs chi-squared tests; can also perform Monte Carlo based tests.

Usage

```
quadrat.test(X, ...)

## S3 method for class 'ppp'
quadrat.test(X, nx=5, ny=nx,
                          alternative=c("two.sided", "regular", "clustered"),
                           method=c("Chisq", "MonteCarlo"),
                           conditional=TRUE, CR=1,
                           lambda=NULL, df.est=NULL,
                           ...,
                           xbreaks=NULL, ybreaks=NULL, tess=NULL,
                           nsim=1999)
## S3 method for class 'quadratcount'
quadrat.test(X,
                          alternative=c("two.sided", "regular", "clustered"),
                          method=c("Chisq", "MonteCarlo"),
                          conditional=TRUE, CR=1,
                          lambda=NULL, df.est=NULL,
                          ...,
                          nsim=1999)
```

Arguments

X	A point pattern (object of class "ppp") to be subjected to the goodness-of-fit test. Alternatively a fitted point process model (object of class "ppm" or "slrm") to be tested. Alternatively X can be the result of applying quadratcount to a point pattern.
nx, ny	Numbers of quadrats in the x and y directions. Incompatible with xbreaks and ybreaks.
alternative	Character string (partially matched) specifying the alternative hypothesis.
method	Character string (partially matched) specifying the test to use: either method="Chisq" for the chi-squared test (the default), or method="MonteCarlo" for a Monte Carlo test.
conditional	Logical. Should the Monte Carlo test be conducted conditionally upon the observed number of points of the pattern? Ignored if method="Chisq".
CR	Optional. Numerical value. The exponent for the Cressie-Read test statistic. See Details.
lambda	Optional. Pixel image (object of class "im") or function (class "funxy") giving the predicted intensity of the point process.
df.est	Optional. Advanced use only. The number of fitted parameters, or the number of degrees of freedom lost by estimation of parameters.
...	Ignored.
xbreaks	Optional. Numeric vector giving the x coordinates of the boundaries of the quadrats. Incompatible with nx.
ybreaks	Optional. Numeric vector giving the y coordinates of the boundaries of the quadrats. Incompatible with ny.
tess	Tessellation (object of class "tess" or something acceptable to as.tess) determining the quadrats. Incompatible with nx, ny, xbreaks, ybreaks.
nsim	The number of simulated samples to generate when method="MonteCarlo".

Details

These functions perform χ^2 tests or Monte Carlo tests of goodness-of-fit for a point process model, based on quadrat counts.

The function quadrat.test is generic, with methods for point patterns (class "ppp"), split point patterns (class "splitppp"), point process models (class "ppm" or "slrm") and quadrat count tables (class "quadratcount").

- if X is a point pattern, we test the null hypothesis that the data pattern is a realisation of Complete Spatial Randomness (the uniform Poisson point process). Marks in the point pattern are ignored. (If lambda is given then the null hypothesis is the Poisson process with intensity lambda.)
- if X is a split point pattern, then for each of the component point patterns (taken separately) we test the null hypotheses of Complete Spatial Randomness. See quadrat.test.splitppp for documentation.

- If X is a fitted point process model, then it should be a Poisson point process model. The data to which this model was fitted are extracted from the model object, and are treated as the data point pattern for the test. We test the null hypothesis that the data pattern is a realisation of the (inhomogeneous) Poisson point process specified by X.

In all cases, the window of observation is divided into tiles, and the number of data points in each tile is counted, as described in quadratcount. The quadrats are rectangular by default, or may be regions of arbitrary shape specified by the argument tess. The expected number of points in each quadrat is also calculated, as determined by CSR (in the first case) or by the fitted model (in the second case). Then the Pearson X^2 statistic

$$X^2 = sum((observed - expected)^2/expected)$$

is computed.

If method="Chisq" then a χ^2 test of goodness-of-fit is performed by comparing the test statistic to the χ^2 distribution with $m - k$ degrees of freedom, where m is the number of quadrats and k is the number of fitted parameters (equal to 1 for quadrat.test.ppp). The default is to compute the *two-sided* p-value, so that the test will be declared significant if X^2 is either very large or very small. One-sided p-values can be obtained by specifying the alternative. An important requirement of the χ^2 test is that the expected counts in each quadrat be greater than 5.

If method="MonteCarlo" then a Monte Carlo test is performed, obviating the need for all expected counts to be at least 5. In the Monte Carlo test, nsim random point patterns are generated from the null hypothesis (either CSR or the fitted point process model). The Pearson X^2 statistic is computed as above. The p-value is determined by comparing the X^2 statistic for the observed point pattern, with the values obtained from the simulations. Again the default is to compute the *two-sided* p-value.

If conditional is TRUE then the simulated samples are generated from the multinomial distribution with the number of "trials" equal to the number of observed points and the vector of probabilities equal to the expected counts divided by the sum of the expected counts. Otherwise the simulated samples are independent Poisson counts, with means equal to the expected counts.

If the argument CR is given, then instead of the Pearson X^2 statistic, the Cressie-Read (1984) power divergence test statistic

$$2nI = \frac{2}{CR(CR+1)} \sum_i \left[\left(\frac{X_i}{E_i} \right)^C R - 1 \right]$$

is computed, where X_i is the ith observed count and E_i is the corresponding expected count. The value CR=1 gives the Pearson X^2 statistic; CR=0 gives the likelihood ratio test statistic G^2; CR=-1/2 gives the Freeman-Tukey statistic T^2; CR=-1 gives the modified likelihood ratio test statistic GM^2; and CR=-2 gives Neyman's modified statistic NM^2. In all cases the asymptotic distribution of this test statistic is the same χ^2 distribution as above.

The return value is an object of class "htest". Printing the object gives comprehensible output about the outcome of the test.

The return value also belongs to the special class "quadrat.test". Plotting the object will display the quadrats, annotated by their observed and expected counts and the Pearson residuals. See the examples.

Value

An object of class "htest". See chisq.test for explanation.

The return value is also an object of the special class "quadrattest", and there is a plot method for this class. See the examples.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

References

Cressie, N. and Read, T.R.C. (1984) Multinomial goodness-of-fit tests. *Journal of the Royal Statistical Society, Series B* **46**, 440–464.

See Also

quadrat.test.splitppp, quadratcount, quadrats, quadratresample, chisq.test, cdf.test.

To test a Poisson point process model against a specific alternative, use anova.ppm.

Examples

```
quadrat.test(simdat)
quadrat.test(simdat, 4, 3)

quadrat.test(simdat, alternative="regular")
quadrat.test(simdat, alternative="clustered")

## Likelihood ratio test
quadrat.test(simdat, CR=0)
## Power divergence tests
quadrat.test(simdat, CR=-1)$p.value
quadrat.test(simdat, CR=-2)$p.value

# Using Monte Carlo p-values
quadrat.test(swedishpines) # Get warning, small expected values.
Nsim <- if(interactive()) 4999 else 9
quadrat.test(swedishpines, method="M", nsim=Nsim)
quadrat.test(swedishpines, method="M", nsim=Nsim, conditional=FALSE)

# quadrat counts
qS <- quadratcount(simdat, 4, 3)
quadrat.test(qS)


te <- quadrat.test(simdat, 4)
residuals(te)  # Pearson residuals

plot(te)

plot(simdat, pch="+", cols="green", lwd=2)
plot(te, add=TRUE, col="red", cex=1.4, lty=2, lwd=3)
```

```
sublab <- eval(substitute(expression(p[chi^2]==z),
                     list(z=signif(te$p.value,3))))
title(sub=sublab, cex.sub=3)

# quadrats of irregular shape
B <- dirichlet(runifpoint(6, Window(simdat)))
qB <- quadrat.test(simdat, tess=B)
plot(simdat, main="quadrat.test(simdat, tess=B)", pch="+")
plot(qB, add=TRUE, col="red", lwd=2, cex=1.2)
```

`quadrat.test.splitppp` *Dispersion Test of CSR for Split Point Pattern Based on Quadrat Counts*

Description

Performs a test of Complete Spatial Randomness for each of the component patterns in a split point pattern, based on quadrat counts. By default performs chi-squared tests; can also perform Monte Carlo based tests.

Usage

```
## S3 method for class 'splitppp'
quadrat.test(X, ..., df=NULL, df.est=NULL, Xname=NULL)
```

Arguments

`X`	A split point pattern (object of class `"splitppp"`), each component of which will be subjected to the goodness-of-fit test.
`...`	Arguments passed to `quadrat.test.ppp`.
`df`, `df.est`, `Xname`	Arguments passed to `pool.quadrattest`.

Details

The function `quadrat.test` is generic, with methods for point patterns (class `"ppp"`), split point patterns (class `"splitppp"`) and point process models (class `"ppm"`).

If `X` is a split point pattern, then for each of the component point patterns (taken separately) we test the null hypotheses of Complete Spatial Randomness, then combine the result into a single test.

The method `quadrat.test.ppp` is applied to each component point pattern. Then the results are pooled using `pool.quadrattest` to obtain a single test.

Value

An object of class `"quadrattest"` which can be printed and plotted.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

quadrat.test, quadratcount, quadrats, quadratresample, chisq.test, cdf.test.

To test a Poisson point process model against a specific Poisson alternative, use anova.ppm.

Examples

```
qH <- quadrat.test(split(humberside), 2, 3)
plot(qH)
qH
```

---

radcumint *Radial Cumulative Integral*

Description

Compute the cumulative integral of an image over increasing radial distances from the origin.

Usage

```
radcumint(X, ..., origin, Xname, result = c("fv", "im"))
```

Arguments

X	A pixel image (object of class "im") with numerical or logical values.
...	Ignored.
origin	Optional. Origin about which the rotations should be performed. Either a numeric vector or a character string as described in the help for shift.owin.
Xname	Optional name for X to be used in the function labels.
result	Character string specifying the kind of result required: either a function object or a pixel image.

Details

This command computes, for each possible distance r, the integral of the pixel values lying inside the disc of radius r centred at the origin.

If result="fv" (the default) the result is a function object f of class "fv". For each value of radius r, the function value f(r) is the integral of X over the disc of radius r.

If result="im" the result is a pixel image, with the same dimensions as X. At a given pixel, the result is equal to f(r) where r is the distance from the given pixel to the origin. That is, at any given pixel, the resulting value is the integral of X over the disc centred at the origin whose boundary passes through the given pixel.

Value

An object of class "fv" or "im", with the same coordinate units as X.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

rotmean, spatialcdf

Examples

```
D <- density(redwood)
plot(radcumint(D))
plot(radcumint(D, result="im"))
```

---

rat *Ratio object*

Description

Stores the numerator, denominator, and value of a ratio as a single object.

Usage

```
rat(ratio, numerator, denominator, check = TRUE)
```

Arguments

ratio, numerator, denominator	Three objects belonging to the same class.
check	Logical. Whether to check that the objects are compatible.

Details

The class "rat" is a simple mechanism for keeping track of the numerator and denominator when calculating a ratio. Its main purpose is simply to signal that the object is a ratio.

The function rat creates an object of class "rat" given the numerator, the denominator and the ratio. No calculation is performed; the three objects are simply stored together.

The arguments ratio, numerator, denominator can be objects of any kind. They should belong to the same class. It is assumed that the relationship

$$\text{ratio} = \frac{\text{numerator}}{\text{denominator}}$$

holds in some version of arithmetic. However, no calculation is performed.

By default the algorithm checks whether the three arguments ratio, numerator, denominator are compatible objects, according to compatible.

The result is equivalent to ratio except for the addition of extra information.

Value

An object equivalent to the object ratio except that it also belongs to the class "rat" and has additional attributes numerator and denominator.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>.

See Also

compatible, pool

---

rectcontact *Contact Distribution Function using Rectangular Structuring Element*

Description

Computes an estimate of the contact distribution function of a set, using a rectangular structuring element.

Usage

```
rectcontact(X, ..., asp = 1, npasses=4,
            eps = NULL, r = NULL, breaks = NULL, correction = c("rs", "km"))
```

Arguments

X	Logical-valued image. The TRUE values in the image determine the spatial region whose contact distribution function should be estimated.
...	Ignored.
asp	Aspect ratio for the rectangular metric. A single positive number. See rectdistmap for explanation.
npasses	Number of passes to perform in the distance algorithm. A positive integer. See rectdistmap for explanation.
eps	Pixel size, if the image should be converted to a finer grid.
r	Optional vector of distance values. Do Not Use This.
breaks	Do Not Use This.
correction	Character vector specifying the edge correction.

Details

To be written.

Value

Object of class "fv".

Author(s)

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

See Also

Hest

Examples

```
## make an image which is TRUE/FALSE inside/outside the letter R
V <- letterR
Frame(V) <- grow.rectangle(Frame(V), 0.5)
Z <- as.im(V, value=TRUE, na.replace=FALSE)
## analyse
plot(rectcontact(Z))
```

---

reload.or.compute *Perform Computations or Retrieve Results From File*

Description

This utility either performs computations and saves the results in a file, or retrieves the results of previous computations stored in a file. If the designated file does not yet exist, the expression will be evaluated, and the results will be saved in the file. If the file already exists, the results will be re-loaded from the file.

Usage

```
reload.or.compute(filename, expr, objects = NULL,
                  context = parent.frame(),
                  destination = parent.frame(),
                  force=FALSE, verbose=TRUE, exclude=NULL)
```

Arguments

filename	Name of data file. A character string.
expr	R language expression to be evaluated.
objects	Optional character vector of names of objects to be saved in filename after evaluating expr, or names of objects that should be present in filename when loaded.

exclude	Optional character vector of names of objects that should **not** be saved in filename and are **not** expected to be present in filename.
context	Environment containing objects that are mentioned in expr (other than objects in the global environment).
destination	Environment into which the resulting objects should be assigned.
force	Logical value indicating whether to perform the computation in any case.
verbose	Logical value indicating whether to print a message indicating whether the data were recomputed or reloaded from the file.

Details

This facility is useful for saving, and later re-loading, the results of time-consuming computations. It would typically be used in an R script file or an Sweave document.

If the file called filename does not yet exist (or if force=TRUE), then expr will be evaluated and the results will be saved in filename using save. By default, all objects that were created by evaluating the expression will be saved in the file. The optional argument objects specifies which results should be saved to the file. The optional argument exclude specifies results which should *not* be saved.

If the file called filename already exists (and if force=FALSE, the default), then this file will be loaded into R using load. The optional argument objects specifies the names of objects that must be present in the file; a warning is issued if any of them are missing.

The resulting objects (either evaluated or loaded from file) can be assigned into any desired destination environment. The default behaviour is equivalent to evaluating expr in the current environment.

If force=TRUE then expr will be evaluated (regardless of whether the file already exists or not) and the results will be saved in filename, overwriting any previously-existing file with that name. This is a convenient way to force the code to re-compute everything in an R script file or Sweave document.

Value

Character vector (invisible) giving the names of the objects computed or loaded.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>.

Examples

```
## Demonstration using a temporary file
## (For real applications, use a permanent file in your own filespace)
myfile <- paste0(tempdir(), .Platform$file.sep, "mydata.rda")
reload.or.compute(myfile, {
   # some very long computation ending with ..
   x <- 42
   intermediateWorking <- 12345
   y <- sqrt(x)
}, exclude="intermediateWorking")
## the values x and y are saved
```

relrisk *Estimate of Spatially-Varying Relative Risk*

Description

Generic command to estimate the spatially-varying probability of each type of point, or the ratios of such probabilities.

Usage

```
relrisk(X, ...)
```

Arguments

X	Either a point pattern (class "ppp") or a fitted point process model (class "ppm") from which the probabilities will be estimated.
...	Additional arguments appropriate to the method.

Details

In a point pattern containing several different types of points, we may be interested in the spatially-varying probability of each possible type, or the relative risks which are the ratios of such probabilities.

The command relrisk is generic and can be used to estimate relative risk in different ways.

The function relrisk.ppp is the method for point pattern datasets. It computes *nonparametric* estimates of relative risk by kernel smoothing.

The function relrisk.ppm is the method for fitted point process models (class "ppm"). It computes *parametric* estimates of relative risk, using the fitted model.

Value

A pixel image, or a list of pixel images, or a numeric vector or matrix, containing the requested estimates of relative risk.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

relrisk.ppp, relrisk.ppm.

relrisk.ppp *Nonparametric Estimate of Spatially-Varying Relative Risk*

Description

Given a multitype point pattern, this function estimates the spatially-varying probability of each type of point, or the ratios of such probabilities, using kernel smoothing. The default smoothing bandwidth is selected by cross-validation.

Usage

```
## S3 method for class 'ppp'
relrisk(X, sigma = NULL, ...,
           at = c("pixels", "points"),
           weights = NULL, varcov = NULL,
           relative=FALSE, normalise=FALSE,
           adjust=1, edge=TRUE, diggle=FALSE,
           se=FALSE, wtype=c("value", "multiplicity"),
           casecontrol=TRUE, control=1, case, shrink=0, fudge=0)
```

Arguments

X	A multitype point pattern (object of class "ppp" which has factor valued marks).
sigma	Optional. The numeric value of the smoothing bandwidth (the standard deviation of isotropic Gaussian smoothing kernel). Alternatively sigma may be a function which can be used to select a different bandwidth for each type of point. See Details.
...	Arguments passed to bw.relrisk to select the bandwidth, or passed to density.ppp to control the pixel resolution.
at	Character string specifying whether to compute the probability values at a grid of pixel locations (at="pixels") or only at the points of X (at="points").
weights	Optional. Weights for the data points of X. A numeric vector, an expression, or a pixel image.
varcov	Optional. Variance-covariance matrix of anisotopic Gaussian smoothing kernel. Incompatible with sigma.
relative	Logical. If FALSE (the default) the algorithm computes the probabilities of each type of point. If TRUE, it computes the *relative risk*, the ratio of probabilities of each type relative to the probability of a control.
normalise	Logical value specifying whether the results should be normalised so that constant risk corresponds to the value 1.
adjust	Optional. Adjustment factor for the bandwidth sigma.
edge	Logical value indicating whether to apply edge correction.
diggle	Logical. If TRUE, use the Jones-Diggle improved edge correction, which is more accurate but slower to compute than the default correction.

se	Logical value indicating whether to compute standard errors as well.
wtype	Character string (partially matched) specifying how the weights should be interpreted for the calculation of standard error. See Details.
casecontrol	Logical. Whether to treat a bivariate point pattern as consisting of cases and controls, and return only the probability or relative risk of a case. Ignored if there are more than 2 types of points. See Details.
control	Integer, or character string, identifying which mark value corresponds to a control.
case	Integer, or character string, identifying which mark value corresponds to a case (rather than a control) in a bivariate point pattern. This is an alternative to the argument control in a bivariate point pattern. Ignored if there are more than 2 types of points.
shrink, fudge	Optional factors for shrinkage estimation as proposed by Bithell (1991). Numeric values, or numeric vectors with one entry for each type of point. See Details.

Details

The command relrisk is generic and can be used to estimate relative risk in different ways.

This function relrisk.ppp is the method for point pattern datasets. It computes *nonparametric* estimates of relative risk by kernel smoothing (Bithell, 1990, 1991; Diggle, 2003; Baddeley, Rubak and Turner, 2015).

If X is a bivariate point pattern (a multitype point pattern consisting of two types of points) then by default, the points of the first type (the first level of marks(X)) are treated as controls or non-events, and points of the second type are treated as cases or events. Then by default this command computes the spatially-varying *probability* of a case, i.e. the probability $p(u)$ that a point at spatial location u will be a case. If relative=TRUE, it computes the spatially-varying *relative risk* of a case relative to a control, $r(u) = p(u)/(1 - p(u))$.

If X is a multitype point pattern with $m > 2$ types, or if X is a bivariate point pattern and casecontrol=FALSE, then by default this command computes, for each type j, a nonparametric estimate of the spatially-varying *probability* of an event of type j. This is the probability $p_j(u)$ that a point at spatial location u will belong to type j. If relative=TRUE, the command computes the *relative risk* of an event of type j relative to a control, $r_j(u) = p_j(u)/p_k(u)$, where events of type k are treated as controls. The argument control determines which type k is treated as a control.

If at = "pixels" the calculation is performed for every spatial location u on a fine pixel grid, and the result is a pixel image representing the function $p(u)$ or a list of pixel images representing the functions $p_j(u)$ or $r_j(u)$ for $j = 1, \ldots, m$. An infinite value of relative risk (arising because the probability of a control is zero) will be returned as NA.

If at = "points" the calculation is performed only at the data points x_i. By default the result is a vector of values $p(x_i)$ giving the estimated probability of a case at each data point, or a matrix of values $p_j(x_i)$ giving the estimated probability of each possible type j at each data point. If relative=TRUE then the relative risks $r(x_i)$ or $r_j(x_i)$ are returned. An infinite value of relative risk (arising because the probability of a control is zero) will be returned as Inf.

Estimation is performed by a simple Nadaraja-Watson type kernel smoother (Bithell, 1990, 1991; Diggle, 2003; Baddeley, Rubak and Turner, 2015, section 14.4). The smoothing bandwidth can be specified in any of the following ways:

- sigma is a single numeric value, giving the standard deviation of the isotropic Gaussian kernel.
- sigma is a numeric vector of length 2, giving the standard deviations in the x and y directions of a Gaussian kernel.
- varcov is a 2 by 2 matrix giving the variance-covariance matrix of the Gaussian kernel.
- sigma is a function which selects the bandwidth. Bandwidth selection will be applied **separately to each type of point**. An example of such a function is bw.diggle.
- sigma and varcov are both missing or null. Then a **common** smoothing bandwidth sigma will be selected by cross-validation using bw.relrisk.
- An infinite smoothing bandwidth, sigma=Inf, is permitted and yields a constant estimate of relative risk.

If se=TRUE then standard errors will also be computed, based on asymptotic theory, *assuming a Poisson process*.

The optional argument weights may provide numerical weights for the points of X. It should be a numeric vector of length equal to npoints(X).

The argument weights can also be an expression. It will be evaluated in the data frame as.data.frame(X) to obtain a vector of weights. The expression may involve the symbols x and y representing the Cartesian coordinates, and the symbol marks representing the mark values.

The argument weights can also be a pixel image (object of class "im"). numerical weights for the data points will be extracted from this image (by looking up the pixel values at the locations of the data points in X).

Value

If se=FALSE (the default), the format is described below. If se=TRUE, the result is a list of two entries, estimate and SE, each having the format described below.

If X consists of only two types of points, and if casecontrol=TRUE, the result is a pixel image (if at="pixels") or a vector (if at="points"). The pixel values or vector values are the probabilities of a case if relative=FALSE, or the relative risk of a case (probability of a case divided by the probability of a control) if relative=TRUE.

If X consists of more than two types of points, or if casecontrol=FALSE, the result is:

- (if at="pixels") a list of pixel images, with one image for each possible type of point. The result also belongs to the class "solist" so that it can be printed and plotted.
- (if at="points") a matrix of probabilities, with rows corresponding to data points x_i, and columns corresponding to types j.

The pixel values or matrix entries are the probabilities of each type of point if relative=FALSE, or the relative risk of each type (probability of each type divided by the probability of a control) if relative=TRUE.

If relative=FALSE, the resulting values always lie between 0 and 1. If relative=TRUE, the results are either non-negative numbers, or the values Inf or NA.

Shrinkage estimate

If the argument shrink is given, the shrinkage estimate proposed by Bithell (1991) is calculated. In this method, a constant value is added to the estimated intensity of the points of each type, before the relative risk calculation is performed.

The argument shrink should be a numeric value, or a numeric vector with one entry for each type of point. All values should be non-negative.

The constant added to the estimated intensity of the points of type j is shrink * K0 * pbar[j] if relative=FALSE and shrink * K0 if relative=TRUE, where K0 is the value of the smoothing kernel at the origin, and pbar[j] is the fraction of points of type j in the data pattern X. Bithell's original proposal corresponds to shrink=4, relative=TRUE.

The argument fudge is rarely used, but is retained for research purposes. It is added to the estimate of the intensity without any rescaling, before the relative risk calculation.

Standard error

If se=TRUE, the standard error of the estimate will also be calculated. The calculation assumes a Poisson point process.

If weights are given, then the calculation of standard error depends on the interpretation of the weights. This is controlled by the argument wtype.

- If wtype="value" (the default), the weights are interpreted as numerical values observed at the data locations. Roughly speaking, standard errors are proportional to the absolute values of the weights.
- If wtype="multiplicity" the weights are interpreted as multiplicities so that a weight of 2 is equivalent to having a pair of duplicated points at the data location. Roughly speaking, standard errors are proportional to the square roots of the weights. Negative weights are not permitted.

The default rule is now wtype="value" but previous versions of relrisk.ppp (in **spatstat.explore** versions 3.1-0 and earlier) effectively used wtype="multiplicity".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>.

References

Baddeley, A., Rubak, E. and Turner, R. (2015) *Spatial Point Patterns: Methodology and Applications with R*. Chapman and Hall/CRC Press.

Bithell, J.F. (1990) An application of density estimation to geographical epidemiology. *Statistics in Medicine* **9**, 691–701.

Bithell, J.F. (1991) Estimation of relative risk functions. *Statistics in Medicine* **10**, 1745–1751.

Diggle, P.J. (2003) *Statistical analysis of spatial point patterns*, Second edition. Arnold.

Diggle, P.J., Zheng, P. and Durr, P. (2005) Non-parametric estimation of spatial segregation in a multivariate point process: bovine tuberculosis in Cornwall, UK. *Applied Statistics* **54**, 645–658.

See Also

There is another method relrisk.ppm for point process models which computes *parametric* estimates of relative risk, using the fitted model.

See also bw.relrisk, density.ppp, Smooth.ppp, eval.im

Examples

```
p.oak <- relrisk(urkiola, 20)
if(interactive()) {
   plot(p.oak, main="proportion of oak")
   plot(eval.im(p.oak > 0.3), main="More than 30 percent oak")
   plot(split(lansing), main="Lansing Woods")
   p.lan <- relrisk(lansing, 0.05, se=TRUE)
   plot(p.lan$estimate, main="Lansing Woods species probability")
   plot(p.lan$SE, main="Lansing Woods standard error")
   wh <- im.apply(p.lan$estimate, which.max)
   types <- levels(marks(lansing))
   wh <- eval.im(types[wh])
   plot(wh, main="Most common species")
}
```

---

relriskHeat *Diffusion Estimate of Conditional Probabilities*

Description

Computes the conditional probability estimator of relative risk based on a multitype point pattern using the diffusion estimate of the type-specific intensities.

Usage

```
relriskHeat(X, ...)

## S3 method for class 'ppp'
relriskHeat(X, ..., sigmaX=NULL, weights=NULL)
```

Arguments

X	A multitype point pattern (object of class "ppp").
...	Arguments passed to densityHeat controlling the estimation of each marginal intensity.
sigmaX	Optional. Numeric vector of bandwidths, one associated with each data point in X.
weights	Optional numeric vector of weights associated with each point of X.

Details

The function relriskHeat is generic. This file documents the method relriskHeat.ppp for spatial point patterns (objects of class "ppp").

This function estimates the spatially-varying conditional probability that a random point (given that it is present) will belong to a given type.

The algorithm separates X into the sub-patterns consisting of points of each type. It then applies densityHeat to each sub-pattern, using the same bandwidth and smoothing regimen for each sub-pattern, as specified by the arguments ....

If weights is specified, it should be a numeric vector of length equal to the number of points in X, so that weights[i] is the weight for data point X[i].

Similarly when performing lagged-arrival smoothing, the argument sigmaX must be a numeric vector of the same length as the number of points in X, and thus contain the point-specific bandwidths in the order corresponding to each of these points regardless of mark.

Value

A named list (of class solist) containing pixel images, giving the estimated conditional probability surfaces for each type.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Tilman Davies <Tilman.Davies@otago.ac.nz>.

References

Agarwal, N. and Aluru, N.R. (2010) A data-driven stochastic collocation approach for uncertainty quantification in MEMS. *International Journal for Numerical Methods in Engineering* **83**, 575–597.

Baddeley, A., Davies, T., Rakshit, S., Nair, G. and McSwiggan, G. (2022) Diffusion smoothing for spatial point patterns. *Statistical Science* **37**, 123–142.

Barry, R.P. and McIntyre, J. (2011) Estimating animal densities and home range in regions with irregular boundaries and holes: a lattice-based alternative to the kernel density estimator. *Ecological Modelling* **222**, 1666–1672.

Botev, Z.I. and Grotowski, J.F. and Kroese, D.P. (2010) Kernel density estimation via diffusion. *Annals of Statistics* **38**, 2916–2957.

See Also

relrisk.ppp for the traditional convolution-based kernel estimator of conditional probability surfaces, and the function risk in the **sparr** package for the density-ratio-based estimator.

Examples

```
## bovine tuberculosis data
X <- subset(btb, select=spoligotype)
plot(X)
P <- relriskHeat(X,sigma=9)
plot(P)
```

rho2hat *Smoothed Relative Density of Pairs of Covariate Values*

Description

Given a point pattern and two spatial covariates Z_1 and Z_2, construct a smooth estimate of the relative risk of the pair (Z_1, Z_2).

Usage

```
rho2hat(object, cov1, cov2, ..., method=c("ratio", "reweight"))
```

Arguments

object	A point pattern (object of class "ppp"), a quadrature scheme (object of class "quad") or a fitted point process model (object of class "ppm").
cov1, cov2	The two covariates. Each argument is either a function(x,y) or a pixel image (object of class "im") providing the values of the covariate at any location, or one of the strings "x" or "y" signifying the Cartesian coordinates.
...	Additional arguments passed to density.ppp to smooth the scatterplots.
method	Character string determining the smoothing method. See Details.

Details

This is a bivariate version of rhohat.

If object is a point pattern, this command produces a smoothed version of the scatterplot of the values of the covariates cov1 and cov2 observed at the points of the point pattern.

The covariates cov1, cov2 must have continuous values.

If object is a fitted point process model, suppose X is the original data point pattern to which the model was fitted. Then this command assumes X is a realisation of a Poisson point process with intensity function of the form

$$\lambda(u) = \rho(Z_1(u), Z_2(u))\kappa(u)$$

where $\kappa(u)$ is the intensity of the fitted model object, and $\rho(z_1, z_2)$ is a function to be estimated. The algorithm computes a smooth estimate of the function ρ.

The method determines how the density estimates will be combined to obtain an estimate of $\rho(z_1, z_2)$:

- If method="ratio", then $\rho(z_1, z_2)$ is estimated by the ratio of two density estimates. The numerator is a (rescaled) density estimate obtained by smoothing the points $(Z_1(y_i), Z_2(y_i))$ obtained by evaluating the two covariate Z_1, Z_2 at the data points y_i. The denominator is a density estimate of the reference distribution of (Z_1, Z_2).
- If method="reweight", then $\rho(z_1, z_2)$ is estimated by applying density estimation to the points $(Z_1(y_i), Z_2(y_i))$ obtained by evaluating the two covariate Z_1, Z_2 at the data points y_i, with weights inversely proportional to the reference density of (Z_1, Z_2).

Value

A pixel image (object of class "im"). Also belongs to the special class "rho2hat" which has a plot method.

Author(s)

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

References

Baddeley, A., Chang, Y.-M., Song, Y. and Turner, R. (2012) Nonparametric estimation of the dependence of a point process on spatial covariates. *Statistics and Its Interface* **5** (2), 221–236.

See Also

rhohat, methods.rho2hat

Examples

```
attach(bei.extra)
plot(rho2hat(bei, elev, grad))
if(require("spatstat.model")) {
  fit <- ppm(bei ~elev, covariates=bei.extra)

    plot(rho2hat(fit, elev, grad))

  plot(rho2hat(fit, elev, grad, method="reweight"))
}
```

---

rhohat *Nonparametric Estimate of Intensity as Function of a Covariate*

Description

Computes a nonparametric estimate of the intensity of a point process, as a function of a (continuous) spatial covariate.

Usage

```
rhohat(object, covariate, ...)

## S3 method for class 'ppp'
rhohat(object, covariate, ...,
       baseline=NULL, weights=NULL,
       method=c("ratio", "reweight", "transform"),
       horvitz=FALSE,
       smoother=c("kernel", "local", "decreasing", "increasing",
       "mountain", "valley", "piecewise"),
```

```
        subset=NULL,
        do.CI=TRUE,
        jitter=TRUE, jitterfactor=1, interpolate=TRUE,
        dimyx=NULL, eps=NULL,
        rule.eps = c("adjust.eps", "grow.frame", "shrink.frame"),
        n = 512, bw = "nrd0", adjust=1, from = NULL, to = NULL,
        bwref=bw,
        covname, confidence=0.95, positiveCI, breaks=NULL)

## S3 method for class 'quad'
rhohat(object, covariate, ...,
       baseline=NULL, weights=NULL,
       method=c("ratio", "reweight", "transform"),
       horvitz=FALSE,
       smoother=c("kernel", "local", "decreasing", "increasing",
                  "mountain", "valley", "piecewise"),
       subset=NULL,
       do.CI=TRUE,
       jitter=TRUE, jitterfactor=1, interpolate=TRUE,
       dimyx=NULL, eps=NULL,
       rule.eps = c("adjust.eps", "grow.frame", "shrink.frame"),
       n = 512, bw = "nrd0", adjust=1, from = NULL, to = NULL,
       bwref=bw,
       covname, confidence=0.95, positiveCI, breaks=NULL)
```

Arguments

object	A point pattern (object of class "ppp" or "lpp"), a quadrature scheme (object of class "quad") or a fitted point process model (object of class "ppm", "slrm" or "lppm").
covariate	Either a function(x,y) or a pixel image (object of class "im") providing the values of the covariate at any location. Alternatively one of the strings "x" or "y" signifying the Cartesian coordinates.
weights	Optional weights attached to the data points. Either a numeric vector of weights for each data point, or a pixel image (object of class "im") or a function(x,y) providing the weights.
baseline	Optional baseline for intensity function. A function(x,y) or a pixel image (object of class "im") providing the values of the baseline at any location.
method	Character string determining the estimation method. See Details.
horvitz	Logical value indicating whether to use Horvitz-Thompson weights. See Details.
smoother	Character string determining the smoothing algorithm and the type of curve that will be estimated. See Details.
subset	Optional. A spatial window (object of class "owin") specifying a subset of the data, from which the estimate should be calculated.

do.CI	Logical value specifying whether to calculate standard errors and confidence bands.
jitter	Logical value. If jitter=TRUE (the default), the values of the covariate at the data points will be jittered (randomly perturbed by adding a small amount of noise) using the function jitter. If jitter=FALSE, the covariate values at the data points will not be altered. See the section on *Randomisation and discretisation*.
jitterfactor	Numeric value controlling the scale of noise added to the covariate values at the data points when jitter=TRUE. Passed to the function jitter as the argument factor.
interpolate	Logical value specifying whether to use spatial interpolation to obtain the values of the covariate at the data points, when the covariate is a pixel image (object of class "im"). If interpolate=FALSE, the covariate value for each data point is simply the value of the covariate image at the pixel centre that is nearest to the data point. If interpolate=TRUE, the covariate value for each data point is obtained by interpolating the nearest pixel values using interp.im.
dimyx, eps, rule.eps	Arguments controlling the pixel resolution at which the covariate will be evaluated. See Details.
bw	Smoothing bandwidth or bandwidth rule (passed to density.default).
adjust	Smoothing bandwidth adjustment factor (passed to density.default).
n, from, to	Arguments passed to density.default to control the number and range of values at which the function will be estimated.
bwref	Optional. An alternative value of bw to use when smoothing the reference density (the density of the covariate values observed at all locations in the window).
...	Additional arguments passed to density.default or locfit.
covname	Optional. Character string to use as the name of the covariate.
confidence	Confidence level for confidence intervals. A number between 0 and 1.
positiveCI	Logical value. If TRUE, confidence limits are always positive numbers; if FALSE, the lower limit of the confidence interval may sometimes be negative. Default is FALSE if smoother="kernel" and TRUE if smoother="local". See Details.
breaks	Breakpoints for the piecewise-constant function computed when smoother='piecewise'. Either a vector of numeric values specifying the breakpoints, or a single integer specifying the number of equally-spaced breakpoints. There is a sensible default.

Details

This command estimates the relationship between point process intensity and a given spatial covariate. Such a relationship is sometimes called a *resource selection function* (if the points are organisms and the covariate is a descriptor of habitat) or a *prospectivity index* (if the points are mineral deposits and the covariate is a geological variable). This command uses nonparametric methods which do not assume a particular form for the relationship.

If `object` is a point pattern, and `baseline` is missing or null, this command assumes that `object` is a realisation of a point process with intensity function $\lambda(u)$ of the form

$$\lambda(u) = \rho(Z(u))$$

where Z is the spatial covariate function given by `covariate`, and $\rho(z)$ is the resource selection function or prospectivity index. A nonparametric estimator of the function $\rho(z)$ is computed.

If `object` is a point pattern, and `baseline` is given, then the intensity function is assumed to be

$$\lambda(u) = \rho(Z(u))B(u)$$

where $B(u)$ is the baseline intensity at location u. A nonparametric estimator of the relative intensity $\rho(z)$ is computed.

If `object` is a fitted point process model, suppose `X` is the original data point pattern to which the model was fitted. Then this command assumes `X` is a realisation of a Poisson point process with intensity function of the form

$$\lambda(u) = \rho(Z(u))\kappa(u)$$

where $\kappa(u)$ is the intensity of the fitted model `object`. A nonparametric estimator of the relative intensity $\rho(z)$ is computed.

The nonparametric estimation procedure is controlled by the arguments `smoother`, `method` and `horvitz`.

The argument `smoother` selects the type of estimation technique.

- If `smoother="kernel"` (the default), the nonparametric estimator is a *kernel smoothing estimator* of $\rho(z)$ (Guan, 2008; Baddeley et al, 2012). The estimated function $\rho(z)$ will be a smooth function of z which takes nonnegative values. If `do.CI=TRUE` (the default), confidence bands are also computed, assuming a Poisson point process. See the section on *Smooth estimates*.
- If `smoother="local"`, the nonparametric estimator is a *local regression estimator* of $\rho(z)$ (Baddeley et al, 2012) obtained using local likelihood. The estimated function $\rho(z)$ will be a smooth function of z. If `do.CI=TRUE` (the default), confidence bands are also computed, assuming a Poisson point process. See the section on *Smooth estimates*.
- If `smoother="increasing"`, we assume that $\rho(z)$ is an increasing function of z, and use the *nonparametric maximum likelihood estimator* of $\rho(z)$ described by Sager (1982). The estimated function will be a step function, that is increasing as a function of z. Confidence bands are not computed. See the section on *Monotone estimates*.
- If `smoother="decreasing"`, we assume that $\rho(z)$ is a decreasing function of z, and use the *nonparametric maximum likelihood estimator* of $\rho(z)$ described by Sager (1982). The estimated function will be a step function, that is decreasing as a function of z. Confidence bands are not computed. See the section on *Monotone estimates*.
- If `smoother="mountain"`, we assume that $\rho(z)$ is a function with an inverted U shape, with a single peak at a value z_0, so that $\rho(z)$ is an increasing function of z for $z < z_0$ and a decreasing function of z for $z > z_0$. We compute the *nonparametric maximum likelihood estimator*. The estimated function will be a step function, which is increasing and then decreasing as a function of z. Confidence bands are not computed. See the section on *Unimodal estimates*.

- If `smoother="valley"`, we assume that $\rho(z)$ is a function with a U shape, with a single minimum at a value z_0, so that $\rho(z)$ is a decreasing function of z for $z < z_0$ and an increasing function of z for $z > z_0$. We compute the *nonparametric maximum likelihood estimator*. The estimated function will be a step function, which is decreasing and then increasing as a function of z. Confidence bands are not computed. See the section on *Unimodal estimates*.
- If `smoother="piecewise"`, the estimate of $\rho(z)$ is piecewise constant. The range of covariate values is divided into several intervals (ranges or bands). The endpoints of these intervals are the breakpoints, which may be specified by the argument `breaks`; there is a sensible default. The estimate of $\rho(z)$ takes a constant value on each interval. The estimate of $\rho(z)$ in each interval of covariate values is simply the average intensity (number of points per unit area) in the relevant sub-region. If `do.CI=TRUE` (the default), confidence bands are computed assuming a Poisson process.

See Baddeley (2018) for a comparison of these estimation techniques (except for `"mountain"` and `"valley"`).

If the argument `weights` is present, then the contribution from each data point `X[i]` to the estimate of ρ is multiplied by `weights[i]`.

If the argument `subset` is present, then the calculations are performed using only the data inside this spatial region.

This technique assumes that `covariate` has continuous values. It is not applicable to covariates with categorical (factor) values or discrete values such as small integers. For a categorical covariate, use `intensity.quadratcount` applied to the result of `quadratcount(X, tess=covariate)`.

The argument `covariate` should be a pixel image, or a function, or one of the strings `"x"` or `"y"` signifying the cartesian coordinates. It will be evaluated on a fine grid of locations, with spatial resolution controlled by the arguments `dimyx,eps,rule.eps` which are passed to `as.mask`.

Value

A function value table (object of class `"fv"`) containing the estimated values of ρ (and confidence limits) for a sequence of values of Z. Also belongs to the class `"rhohat"` which has special methods for `print`, `plot` and `predict`.

Smooth estimates

Smooth estimators of $\rho(z)$ were proposed by Baddeley and Turner (2005) and Baddeley et al (2012). Similar estimators were proposed by Guan (2008) and in the literature on relative distributions (Handcock and Morris, 1999).

The estimated function $\rho(z)$ will be a smooth function of z.

The smooth estimation procedure involves computing several density estimates and combining them. The algorithm used to compute density estimates is determined by `smoother`:

- If `smoother="kernel"`, the smoothing procedure is based on fixed-bandwidth kernel density estimation, performed by `density.default`.
- If `smoother="local"`, the smoothing procedure is based on local likelihood density estimation, performed by `locfit`.

The argument `method` determines how the density estimates will be combined to obtain an estimate of $\rho(z)$:

- If `method="ratio"`, then $\rho(z)$ is estimated by the ratio of two density estimates, The numerator is a (rescaled) density estimate obtained by smoothing the values $Z(y_i)$ of the covariate Z observed at the data points y_i. The denominator is a density estimate of the reference distribution of Z. See Baddeley et al (2012), equation (8). This is similar but not identical to an estimator proposed by Guan (2008).
- If `method="reweight"`, then $\rho(z)$ is estimated by applying density estimation to the values $Z(y_i)$ of the covariate Z observed at the data points y_i, with weights inversely proportional to the reference density of Z. See Baddeley et al (2012), equation (9).
- If `method="transform"`, the smoothing method is variable-bandwidth kernel smoothing, implemented by applying the Probability Integral Transform to the covariate values, yielding values in the range 0 to 1, then applying edge-corrected density estimation on the interval $[0, 1]$, and back-transforming. See Baddeley et al (2012), equation (10).

If `horvitz=TRUE`, then the calculations described above are modified by using Horvitz-Thompson weighting. The contribution to the numerator from each data point is weighted by the reciprocal of the baseline value or fitted intensity value at that data point; and a corresponding adjustment is made to the denominator.

Pointwise confidence intervals for the true value of $\rho(z)$ are also calculated for each z, and will be plotted as grey shading. The confidence intervals are derived using the central limit theorem, based on variance calculations which assume a Poisson point process. If `positiveCI=FALSE`, the lower limit of the confidence interval may sometimes be negative, because the confidence intervals are based on a normal approximation to the estimate of $\rho(z)$. If `positiveCI=TRUE`, the confidence limits are always positive, because the confidence interval is based on a normal approximation to the estimate of $\log(\rho(z))$. For consistency with earlier versions, the default is `positiveCI=FALSE` for `smoother="kernel"` and `positiveCI=TRUE` for `smoother="local"`.

Monotone estimates

The nonparametric maximum likelihood estimator of a monotone function $\rho(z)$ was described by Sager (1982). This method assumes that $\rho(z)$ is either an increasing function of z, or a decreasing function of z. The estimated function will be a step function, increasing or decreasing as a function of z.

This estimator is chosen by specifying `smoother="increasing"` or `smoother="decreasing"`. The argument `method` is ignored this case.

To compute the estimate of $\rho(z)$, the algorithm first computes several primitive step-function estimates, and then takes the maximum of these primitive functions.

If `smoother="decreasing"`, each primitive step function takes the form $\rho(z) = \lambda$ when $z \leq t$, and $\rho(z) = 0$ when $z > t$, where and λ is a primitive estimate of intensity based on the data for $Z \leq t$. The jump location t will be the value of the covariate Z at one of the data points. The primitive estimate λ is the average intensity (number of points divided by area) for the region of space where the covariate value is less than or equal to t.

If `horvitz=TRUE`, then the calculations described above are modified by using Horvitz-Thompson weighting. The contribution to the numerator from each data point is weighted by the reciprocal of the baseline value or fitted intensity value at that data point; and a corresponding adjustment is made to the denominator.

Confidence intervals are not available for the monotone estimators.

Unimodal estimators

If `smoother="valley"` then we estimate a U-shaped function. A function $\rho(z)$ is U-shaped if it is decreasing when $z < z_0$ and increasing when $z > z_0$, where z_0 is called the critical value. The nonparametric maximum likelihood estimate of such a function can be computed by profiling over z_0. The algorithm considers all possible candidate values of the critical value z_0, and estimates the function $\rho(z)$ separately on the left and right of z_0 using the monotone estimators described above. These function estimates are combined into a single function, and the Poisson point process likelihood is computed. The optimal value of z_0 is the one which maximises the Poisson point process likelihood.

If `smoother="mountain"` then we estimate a function which has an inverted U shape. A function $\rho(z)$ is inverted-U-shaped if it is increasing when $z < z_0$ and decreasing when $z > z_0$. The nonparametric maximum likelihood estimate of such a function can be computed by profiling over z_0 using the same technique *mutatis mutandis*.

Confidence intervals are not available for the unimodal estimators.

Randomisation

By default, `rhohat` adds a small amount of random noise to the data. This is designed to suppress the effects of discretisation in pixel images.

This strategy means that `rhohat` does not produce exactly the same result when the computation is repeated. If you need the results to be exactly reproducible, set `jitter=FALSE`.

By default, the values of the covariate at the data points will be randomly perturbed by adding a small amount of noise using the function `jitter`. To reduce this effect, set `jitterfactor` to a number smaller than 1. To suppress this effect entirely, set `jitter=FALSE`.

Author(s)

Smoothing algorithm by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Ya-Mei Chang, Yong Song, and Rolf Turner <rolfturner@posteo.net>.

Nonparametric maximum likelihood algorithm by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

References

Baddeley, A., Chang, Y.-M., Song, Y. and Turner, R. (2012) Nonparametric estimation of the dependence of a point process on spatial covariates. *Statistics and Its Interface* **5** (2), 221–236.

Baddeley, A. and Turner, R. (2005) Modelling spatial point patterns in R. In: A. Baddeley, P. Gregori, J. Mateu, R. Stoica, and D. Stoyan, editors, *Case Studies in Spatial Point Pattern Modelling*, Lecture Notes in Statistics number 185. Pages 23–74. Springer-Verlag, New York, 2006. ISBN: 0-387-28311-0.

Baddeley, A. (2018) A statistical commentary on mineral prospectivity analysis. Chapter 2, pages 25–65 in *Handbook of Mathematical Geosciences: Fifty Years of IAMG*, edited by B.S. Daya Sagar, Q. Cheng and F.P. Agterberg. Springer, Berlin.

Guan, Y. (2008) On consistent nonparametric intensity estimation for inhomogeneous spatial point processes. *Journal of the American Statistical Association* **103**, 1238–1247.

Handcock, M.S. and Morris, M. (1999) *Relative Distribution Methods in the Social Sciences*. Springer, New York.

Sager, T.W. (1982) Nonparametric maximum likelihood estimation of spatial patterns. *Annals of Statistics* **10**, 1125–1136.

See Also

rho2hat, methods.rhohat, parres.

See ppm for a parametric method for the same problem.

Examples

```
X <-  rpoispp(function(x,y){exp(3+3*x)})
rho <- rhohat(X, "x")
rho <- rhohat(X, function(x,y){x})
plot(rho)
curve(exp(3+3*x), lty=3, col=4, lwd=2, add=TRUE)

rhoB <- rhohat(X, "x", method="reweight")
rhoC <- rhohat(X, "x", method="transform")

rhoI <- rhohat(X, "x", smoother="increasing")
rhoM <- rhohat(X, "x", smoother="mountain")

plot(rhoI, add=TRUE, .y ~ .x, col=6)
legend("top", lty=c(3, 1), col=c(4, 6), lwd=c(2, 1),
       legend=c("true", "increasing"))
```

---

roc *Receiver Operating Characteristic*

Description

Computes the Receiver Operating Characteristic curve for a point pattern or a fitted point process model.

Usage

```
 roc(X, ...)

## S3 method for class 'ppp'
roc(X, covariate,
                  ...,
                  baseline = NULL, high = TRUE, weights = NULL,
                  observations=c("exact", "presence"),
                  method = "raw",
                  CI = "none", alpha=0.05,
```

```
                  subset=NULL)

## S3 method for class 'cdftest'
roc(X, ..., high=TRUE)

## S3 method for class 'bermantest'
roc(X, ..., high=TRUE)

## S3 method for class 'im'
roc(X, covariate, ..., high=TRUE)
```

Arguments

X	Point pattern (object of class "ppp" or "lpp") or fitted point process model (object of class "ppm" or "kppm" or "lppm") or fitted spatial logistic regression model (object of class "slrm") or some other kind of data.
covariate	Spatial covariate. Either a function(x,y), a pixel image (object of class "im"), or one of the strings "x" or "y" indicating the Cartesian coordinates. Traditionally omitted when X is a fitted model.
...	Arguments passed to as.mask controlling the pixel resolution for calculations.
baseline	Optional. A spatial object giving a baseline intensity. Usually a function(x,y) or a pixel image (object of class "im") giving the baseline intensity at any location within the observation window. Alternatively a point pattern (object of class "ppp") with the locations of the reference population.
high	Logical value indicating whether the threshold operation should favour high or low values of the covariate.
weights	Optional. Numeric vector of weights attached to the data points.
observations	Character string (partially matched) specifying whether to compute the ROC curve using the exact point coordinates (observations="exact", the default) or using the discretised presence-absence data (observations="presence").
method	The method or methods that should be used to estimate the ROC curve. A character vector: current choices are "raw", "monotonic", "smooth" and "all". See Details.
CI	Character string (partially matched) specifying whether confidence intervals should be computed, and for which method. See Details.
alpha	Numeric value between 0 and 1. The confidence intervals will have confidence level 1-alpha. The default gives 95% confidence intervals.
subset	Optional. A spatial window (object of class "owin") specifying a subset of the data, from which the ROC should be calculated.

Details

This command computes the Receiver Operating Characteristic (ROC) curve. The area under the ROC is computed by auc.

The function roc is generic, with methods for point patterns, fitted point process models, and other kinds of data.

For a point pattern X and a spatial covariate Z, the ROC is a plot showing the ability of the covariate to separate the spatial domain into areas of high and low density of points. For each possible threshold z, the algorithm calculates the fraction $a(z)$ of area in the study region where the covariate takes a value greater than z, and the fraction $b(z)$ of data points for which the covariate value is greater than z. The ROC is a plot of $b(z)$ against $a(z)$ for all thresholds z. This is called the 'raw' ROC curve.

There are currently three methods to estimate the ROC curve:

"raw" uses the raw empirical spatial cummulative distribution function of the covariate.

"monotonic" uses a monotonic regression to estimate the relation between the covariate and the point process intensity and then calculates the ROC from that. This corresponds to a either a convex minorant or a concave majorant of the raw ROC curve.

"smooth" uses a smooth estimate of the relation between the covariate and the point process intensity and then calculates the ROC from that. See roc.rhohat for details.

"all" uses all of the above methods.

If CI is one of the strings 'raw', 'monotonic' or 'smooth', then pointwise 95% confidence intervals for the true ROC curve will be computed based on the raw, monotonic or smooth estimates, respectively. The confidence level is 1-alpha, so that for example alpha=0.01 would give 99% confidence intervals. By default, confidence bands for the ROC curve are not computed.

Some other kinds of objects in **spatstat** contain sufficient data to compute the ROC curve. These include the objects returned by rhohat, cdf.test and berman.test. Methods are provided here to compute the ROC curve from these objects.

The method for pixel images (objects of class "im") assumes that X represents a density or intensity function, and that the objective is to segregate the spatial region into subregions of high and low total density by thresholding the covariate.

Value

Function value table (object of class "fv") which can be plotted to show the ROC curve. Also belongs to class "roc".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Ege Rubak <rubak@math.aau.dk> and Suman Rakshit <Suman.Rakshit@curtin.edu.au>.

References

Baddeley, A., Rubak, E., Rakshit, S. and Nair, G. (2025) ROC curves for spatial point patterns and presence-absence data. doi:10.48550/arXiv.2506.03414..

Lobo, J.M., Jiménez-Valverde, A. and Real, R. (2007) AUC: a misleading measure of the performance of predictive distribution models. *Global Ecology and Biogeography* **17**(2) 145–151.

Nam, B.-H. and D'Agostino, R. (2002) Discrimination index, the area under the ROC curve. Pages 267–279 in Huber-Carol, C., Balakrishnan, N., Nikulin, M.S. and Mesbah, M., *Goodness-of-fit tests and model validity*, Birkhäuser, Basel.

See Also

roc.ppm, roc.lpp, roc.rhohat.

auc

Examples

```
gold <- rescale(murchison$gold, 1000, "km")
faults <- rescale(murchison$faults, 1000, "km")
dfault <- distfun(faults)

if(interactive()) {
  plot(roc(gold, dfault, method = "all", high=FALSE))
} else {
  ## reduce sample resolution to save computation time in test
  plot(roc(gold, dfault, method = "all", high=FALSE, eps=8))
}

# Using either an image or reference population as baseline
cases <- split(chorley)$larynx
controls <- split(chorley)$lung
covar <- distfun(as.ppp(chorley.extra$incin, W = Window(chorley)))
if(interactive()) {
  population <- density(controls, sigma=0.15, eps=0.1)
} else {
  ## reduce resolution to save computation time in test
  population <- density(controls, sigma=0.3, eps=0.25)
}
population <- eval.im(pmax(population, 1e-10))
roc1 <- roc(cases, covar, baseline = population, high = FALSE, method="all")
roc2 <- roc(cases, covar, baseline = controls, high = FALSE, method="all")
plot(anylist(roc1=roc1, roc2=roc2), main = "")
```

---

roc.rhohat *Receiver Operating Characteristic*

Description

Computes the Receiver Operating Characteristic curve for a point pattern from a given intensity function.

Usage

```
## S3 method for class 'rhohat'
roc(X, ..., high = TRUE)
```

Arguments

X	Estimate of rho function (object of class "rhohat").
...	Ignored.
high	Logical value indicating whether the threshold operation should favour high or low values of the covariate.

Details

This command computes Receiver Operating Characteristic curve from an estimate of rhohat.

See roc for general information about ROC curves.

It is assumed that the rhohat function X describes the true functional dependency of the point process intensity on the given covariate (included in the rhohat-object). The ROC curve is estimated by integrating this function as described in Baddeley et al (2025).

Value

Function value table (object of class "fv" and "roc") which can be plotted to show the ROC curve.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Ege Rubak <rubak@math.aau.dk> and Suman Rakshit <Suman.Rakshit@curtin.edu.au>.

References

Baddeley, A., Rubak, E., Rakshit, S. and Nair, G. (2025) ROC curves for spatial point patterns and presence-absence data. doi:10.48550/arXiv.2506.03414.

See Also

roc, auc

Examples

```
rh <- rhohat(swedishpines, "x")
plot(roc(rh))
```

---

rose *Rose Diagram*

Description

Plots a rose diagram (rose of directions), the analogue of a histogram or density plot for angular data.

Usage

```
rose(x, ...)

## Default S3 method:
rose(x, breaks = NULL, ...,
                       weights=NULL,
                       nclass = NULL,
                       unit = c("degree", "radian"),
                       start=0, clockwise=FALSE,
                       main)

## S3 method for class 'histogram'
rose(x, ...,
                       unit = c("degree", "radian"),
                       start=0, clockwise=FALSE,
                       main, labels=TRUE, at=NULL, do.plot = TRUE)

## S3 method for class 'density'
rose(x, ...,
                   unit = c("degree", "radian"),
                   start=0, clockwise=FALSE,
                   main, labels=TRUE, at=NULL, do.plot = TRUE)

## S3 method for class 'fv'
rose(x, ...,
                   unit = c("degree", "radian"),
                   start=0, clockwise=FALSE,
                   main, labels=TRUE, at=NULL, do.plot = TRUE)
```

Arguments

x	Data to be plotted. A numeric vector containing angles, or a histogram object containing a histogram of angular values, or a density object containing a smooth density estimate for angular data, or an fv object giving a function of an angular argument.
breaks, nclass	Arguments passed to hist to determine the histogram breakpoints.
...	Additional arguments passed to polygon controlling the appearance of the plot (or passed from rose.default to hist to control the calculation of the histogram).
unit	The unit in which the angles are expressed.
start	The starting direction for measurement of angles, that is, the spatial direction which corresponds to a measured angle of zero. Either a character string giving a compass direction ("N" for north, "S" for south, "E" for east, or "W" for west) or a number giving the angle from the the horizontal (East) axis to the starting direction. For example, if unit="degree" and clockwise=FALSE, then start=90 and start="N" are equivalent. The default is to measure angles anti-clockwise from the horizontal axis (East direction).

clockwise	Logical value indicating whether angles increase in the clockwise direction (clockwise=TRUE) or anti-clockwise, counter-clockwise direction (clockwise=FALSE, the default).
weights	Optional vector of numeric weights associated with x.
main	Optional main title for the plot.
labels	Either a logical value indicating whether to plot labels next to the tick marks, or a vector of labels for the tick marks.
at	Optional vector of angles at which tick marks should be plotted. Set at=numeric(0) to suppress tick marks.
do.plot	Logical value indicating whether to really perform the plot.

Details

A rose diagram or rose of directions is the analogue of a histogram or bar chart for data which represent angles in two dimensions. The bars of the bar chart are replaced by circular sectors in the rose diagram.

The function rose is generic, with a default method for numeric data, and methods for histograms and function tables.

If x is a numeric vector, it must contain angular values in the range 0 to 360 (if unit="degree") or in the range 0 to 2 * pi (if unit="radian"). A histogram of the data will first be computed using hist. Then the rose diagram of this histogram will be plotted by rose.histogram.

If x is an object of class "histogram" produced by the function hist, representing the histogram of angular data, then the rose diagram of the densities (rather than the counts) in this histogram object will be plotted.

If x is an object of class "density" produced by circdensity or density.default, representing a kernel smoothed density estimate of angular data, then the rose diagram of the density estimate will be plotted.

If x is a function value table (object of class "fv") then the argument of the function will be interpreted as an angle, and the value of the function will be interpreted as the radius.

By default, angles are interpreted using the mathematical convention where the zero angle is the horizontal x axis, and angles increase anti-clockwise. Other conventions can be specified using the arguments start and clockwise. Standard compass directions are obtained by setting unit="degree", start="N" and clockwise=TRUE.

Value

A window (class "owin") containing the plotted region.

Author(s)

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

Rolf Turner <rolfturner@posteo.net>

and Ege Rubak <rubak@math.aau.dk>

See Also

fv, hist, circdensity, density.default.

Examples

```
ang <- runif(1000, max=360)
rose(ang, col="grey")
rose(ang, col="grey", start="N", clockwise=TRUE)
```

`rotmean` *Rotational Average of a Pixel Image*

Description

Compute the average pixel value over all rotations of the image about the origin, as a function of distance from the origin.

Usage

```
rotmean(X, ..., origin, padzero=TRUE, Xname, result=c("fv", "im"), adjust=1)
```

Arguments

`X`	A pixel image.
`...`	Ignored.
`origin`	Optional. Origin about which the rotations should be performed. Either a numeric vector or a character string as described in the help for `shift.owin`.
`padzero`	Logical. If `TRUE` (the default), the value of `X` is assumed to be zero outside the window of `X`. If `FALSE`, the value of `X` is taken to be undefined outside the window of `X`.
`Xname`	Optional name for `X` to be used in the function labels.
`result`	Character string specifying the kind of result required: either a function object or a pixel image.
`adjust`	Adjustment factor for bandwidth used in kernel smoothing.

Details

This command computes, for each possible distance r, the average pixel value of the pixels lying at distance r from the origin. Kernel smoothing is used to obtain a smooth function of r.

If `result="fv"` (the default) the result is a function object of class `"fv"` giving the mean pixel value of `X` as a function of distance from the origin.

If `result="im"` the result is a pixel image, with the same dimensions as `X`, giving the mean value of `X` over all pixels lying at the same distance from the origin as the current pixel.

If `padzero=TRUE` (the default), the value of `X` is assumed to be zero outside the window of `X`. The rotational mean at a given distance r is the average value of the image `X` over the *entire* circle of radius r, including zero values outside the window if the circle lies partly outside the window.

If `padzero=FALSE`, the value of `X` is taken to be undefined outside the window of `X`. The rotational mean is the average of the `X` values over the *subset* of the circle of radius r that lies entirely inside the window.

Value

An object of class "fv" or "im", with the same coordinate units as X.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

radcumint

Examples

```
online <- interactive()
resolution <- if(online) 128 else 32
Z <- setcov(square(1), dimyx=resolution)
f <- rotmean(Z)
if(online) {
  plot(rotmean(Z))
  plot(rotmean(Z, result="im"))
}
```

---

scan.test *Spatial Scan Test*

Description

Performs the Spatial Scan Test for clustering in a spatial point pattern, or for clustering of one type of point in a bivariate spatial point pattern.

Usage

```
scan.test(X, r, ...,
          method = c("poisson", "binomial"),
          nsim = 19,
          baseline = NULL,
          case = 2,
          alternative = c("greater", "less", "two.sided"),
          verbose = TRUE)
```

Arguments

X	A point pattern (object of class "ppp").
r	Radius of circle to use. A single number or a numeric vector.
...	Optional. Arguments passed to as.mask to determine the spatial resolution of the computations.

method	Either "poisson" or "binomial" specifying the type of likelihood.
nsim	Number of simulations for computing Monte Carlo p-value.
baseline	Baseline for the Poisson intensity, if method="poisson". A pixel image or a function.
case	Which type of point should be interpreted as a case, if method="binomial". Integer or character string.
alternative	Alternative hypothesis: "greater" if the alternative postulates that the mean number of points inside the circle will be greater than expected under the null.
verbose	Logical. Whether to print progress reports.

Details

The spatial scan test (Kulldorf, 1997) is applied to the point pattern X.

In a nutshell,

- If method="poisson" then a significant result would mean that there is a circle of radius r, located somewhere in the spatial domain of the data, which contains a significantly higher than expected number of points of X. That is, the pattern X exhibits spatial clustering.
- If method="binomial" then X must be a bivariate (two-type) point pattern. By default, the first type of point is interpreted as a control (non-event) and the second type of point as a case (event). A significant result would mean that there is a circle of radius r which contains a significantly higher than expected number of cases. That is, the cases are clustered together, conditional on the locations of all points.

Following is a more detailed explanation.

- If method="poisson" then the scan test based on Poisson likelihood is performed (Kulldorf, 1997). The dataset X is treated as an unmarked point pattern. By default (if baseline is not specified) the null hypothesis is complete spatial randomness CSR (i.e. a uniform Poisson process). The alternative hypothesis is a Poisson process with one intensity β_1 inside some circle of radius r and another intensity β_0 outside the circle. If baseline is given, then it should be a pixel image or a function(x,y). The null hypothesis is an inhomogeneous Poisson process with intensity proportional to baseline. The alternative hypothesis is an inhomogeneous Poisson process with intensity beta1 * baseline inside some circle of radius r, and beta0 * baseline outside the circle.
- If method="binomial" then the scan test based on binomial likelihood is performed (Kulldorf, 1997). The dataset X must be a bivariate point pattern, i.e. a multitype point pattern with two types. The null hypothesis is that all permutations of the type labels are equally likely. The alternative hypothesis is that some circle of radius r has a higher proportion of points of the second type, than expected under the null hypothesis.

The result of scan.test is a hypothesis test (object of class "htest") which can be plotted to report the results. The component p.value contains the p-value.

The result of scan.test can also be plotted (using the plot method for the class "scan.test"). The plot is a pixel image of the Likelihood Ratio Test Statistic (2 times the log likelihood ratio) as a function of the location of the centre of the circle. This pixel image can be extracted from the object using as.im.scan.test. The Likelihood Ratio Test Statistic is computed by scanLRTS.

Value

An object of class "htest" (hypothesis test) which also belongs to the class "scan.test". Printing this object gives the result of the test. Plotting this object displays the Likelihood Ratio Test Statistic as a function of the location of the centre of the circle.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

References

Kulldorff, M. (1997) A spatial scan statistic. *Communications in Statistics — Theory and Methods* **26**, 1481–1496.

See Also

plot.scan.test, as.im.scan.test, relrisk, scanLRTS

Examples

```
nsim <- if(interactive()) 19 else 2
rr <- if(interactive()) seq(0.5, 1, by=0.1) else c(0.5, 1)
scan.test(redwood, 0.1 * rr, method="poisson", nsim=nsim)
scan.test(chorley, rr, method="binomial", case="larynx", nsim=nsim)
```

---

scanLRTS *Likelihood Ratio Test Statistic for Scan Test*

Description

Calculate the Likelihood Ratio Test Statistic for the Scan Test, at each spatial location.

Usage

```
scanLRTS(X, r, ...,
   method = c("poisson", "binomial"),
   baseline = NULL, case = 2,
   alternative = c("greater", "less", "two.sided"),
   saveopt = FALSE,
   Xmask = NULL)
```

Arguments

X	A point pattern (object of class "ppp").
r	Radius of circle to use. A single number or a numeric vector.
...	Optional. Arguments passed to as.mask to determine the spatial resolution of the computations.
method	Either "poisson" or "binomial" specifying the type of likelihood.
baseline	Baseline for the Poisson intensity, if method="poisson". A pixel image or a function.
case	Which type of point should be interpreted as a case, if method="binomial". Integer or character string.
alternative	Alternative hypothesis: "greater" if the alternative postulates that the mean number of points inside the circle will be greater than expected under the null.
saveopt	Logical value indicating to save the optimal value of r at each location.
Xmask	Internal use only.

Details

This command computes, for all spatial locations u, the Likelihood Ratio Test Statistic $\Lambda(u)$ for a test of homogeneity at the location u, as described below. The result is a pixel image giving the values of $\Lambda(u)$ at each pixel.

The **maximum** value of $\Lambda(u)$ over all locations u is the *scan statistic*, which is the basis of the *scan test* performed by scan.test.

- If method="poisson" then the test statistic is based on Poisson likelihood. The dataset X is treated as an unmarked point pattern. By default (if baseline is not specified) the null hypothesis is complete spatial randomness CSR (i.e. a uniform Poisson process). At the spatial location u, the alternative hypothesis is a Poisson process with one intensity β_1 inside the circle of radius r centred at u, and another intensity β_0 outside the circle. If baseline is given, then it should be a pixel image or a function(x,y). The null hypothesis is an inhomogeneous Poisson process with intensity proportional to baseline. The alternative hypothesis is an inhomogeneous Poisson process with intensity beta1 * baseline inside the circle, and beta0 * baseline outside the circle.
- If method="binomial" then the test statistic is based on binomial likelihood. The dataset X must be a bivariate point pattern, i.e. a multitype point pattern with two types. The null hypothesis is that all permutations of the type labels are equally likely. The alternative hypothesis is that the circle of radius r centred at u has a higher proportion of points of the second type, than expected under the null hypothesis.

If r is a vector of more than one value for the radius, then the calculations described above are performed for every value of r. Then the maximum over r is taken for each spatial location u. The resulting pixel value of scanLRTS at a location u is the profile maximum of the Likelihood Ratio Test Statistic, that is, the maximum of the Likelihood Ratio Test Statistic for circles of all radii, centred at the same location u.

If you have already performed a scan test using scan.test, the Likelihood Ratio Test Statistic can be extracted from the test result using the function as.im.scan.test.

Value

A pixel image (object of class "im") whose pixel values are the values of the (profile) Likelihood Ratio Test Statistic at each spatial location.

Warning: window size

Note that the result of scanLRTS is a pixel image on a larger window than the original window of X. The expanded window contains the centre of any circle of radius r that has nonempty intersection with the original window.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

References

Kulldorff, M. (1997) A spatial scan statistic. *Communications in Statistics — Theory and Methods* **26**, 1481–1496.

See Also

scan.test, as.im.scan.test

Examples

```
plot(scanLRTS(redwood, 0.1, method="poisson"))
sc <- scanLRTS(chorley, 1, method="binomial", case="larynx")
plot(sc)
scanstatchorley <- max(sc)
```

---

sdr *Sufficient Dimension Reduction*

Description

Given a point pattern and a set of predictors, find a minimal set of new predictors, each constructed as a linear combination of the original predictors.

Usage

```
sdr(X, covariates, ...)

## S3 method for class 'ppp'
sdr(X, covariates,
                  method = c("DR", "NNIR", "SAVE", "SIR", "TSE"),
                  Dim1 = 1, Dim2 = 1, predict=FALSE, ...)
```

Arguments

X	A point pattern (object of class "ppp").
covariates	A list of pixel images (objects of class "im") to serve as predictor variables.
method	Character string indicating which method to use. See Details.
Dim1	Dimension of the first order Central Intensity Subspace (applicable when method is "DR", "NNIR", "SAVE" or "TSE").
Dim2	Dimension of the second order Central Intensity Subspace (applicable when method="TSE").
predict	Logical value indicating whether to compute the new predictors as well.
...	Additional arguments (ignored by sdr.ppp).

Details

Given a point pattern X and predictor variables $Z_1, \ldots, Z_p$, Sufficient Dimension Reduction methods (Guan and Wang, 2010) attempt to find a minimal set of new predictor variables, each constructed by taking a linear combination of the original predictors, which explain the dependence of X on $Z_1, \ldots, Z_p$. The methods do not assume any particular form of dependence of the point pattern on the predictors. The predictors are assumed to be Gaussian random fields.

Available methods are:

method="DR"	directional regression
method="NNIR"	nearest neighbour inverse regression
method="SAVE"	sliced average variance estimation
method="SIR"	sliced inverse regression
method="TSE"	two-step estimation

The result includes a matrix B whose columns are estimates of the basis vectors of the space of new predictors. That is, the jth column of B expresses the jth new predictor as a linear combination of the original predictors.

If predict=TRUE, the new predictors are also evaluated. They can also be evaluated using sdrPredict.

Value

A list with components B, M or B, M1, M2 where B is a matrix whose columns are estimates of the basis vectors for the space, and M or M1,M2 are matrices containing estimates of the kernel.

If predict=TRUE, the result also includes a component Y which is a list of pixel images giving the values of the new predictors.

Author(s)

Matlab original by Yongtao Guan, translated to R by Suman Rakshit.

References

Guan, Y. and Wang, H. (2010) Sufficient dimension reduction for spatial point processes directed by Gaussian random fields. *Journal of the Royal Statistical Society, Series B*, **72**, 367–387.

See Also

sdrPredict to compute the new predictors from the coefficient matrix.

dimhat to estimate the subspace dimension.

subspaceDistance

Examples

```
A <- sdr(bei, bei.extra, predict=TRUE)
A
Y1 <- A$Y[[1]]
plot(Y1)
points(bei, pch=".", cex=2)
# investigate likely form of dependence
plot(rhohat(bei, Y1))
```

---

sdrPredict *Compute Predictors from Sufficient Dimension Reduction*

Description

Given the result of a Sufficient Dimension Reduction method, compute the new predictors.

Usage

```
sdrPredict(covariates, B)
```

Arguments

covariates	A list of pixel images (objects of class "im").
B	Either a matrix of coefficients for the covariates, or the result of a call to sdr.

Details

This function assumes that sdr has already been used to find a minimal set of predictors based on the covariates. The argument B should be either the result of sdr or the coefficient matrix returned as one of the results of sdr. The columns of this matrix define linear combinations of the covariates. This function evaluates those linear combinations, and returns a list of pixel images containing the new predictors.

Value

A list of pixel images (objects of class "im") with one entry for each column of B.

Author(s)

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

See Also

sdr

Examples

```
A <- sdr(bei, bei.extra)
Y <- sdrPredict(bei.extra, A)
Y
```

---

segregation.test *Test of Spatial Segregation of Types*

Description

Performs a Monte Carlo test of spatial segregation of the types in a multitype point pattern.

Usage

```
segregation.test(X, ...)

## S3 method for class 'ppp'
segregation.test(X, ..., nsim = 19,
       permute = TRUE, verbose = TRUE, Xname)
```

Arguments

X	Multitype point pattern (object of class "ppp" with factor-valued marks).
...	Additional arguments passed to relrisk.ppp to control the smoothing parameter or bandwidth selection.
nsim	Number of simulations for the Monte Carlo test.
permute	Argument passed to rlabel. If TRUE (the default), randomisation is performed by randomly permuting the labels of X. If FALSE, randomisation is performing by resampling the labels with replacement.
verbose	Logical value indicating whether to print progress reports.
Xname	Optional character string giving the name of the dataset X.

Details

The Monte Carlo test of spatial segregation of types, proposed by Kelsall and Diggle (1995) and Diggle et al (2005), is applied to the point pattern X. The test statistic is

$$T = \sum_i \sum_m (\widehat{p}(m \mid x_i) - \overline{p}_m)^2$$

where $\widehat{p}(m \mid x_i)$ is the leave-one-out kernel smoothing estimate of the probability that the i-th data point has type m, and $\overline{p}_m$ is the average fraction of data points which are of type m. The statistic T

is evaluated for the data and for `nsim` randomised versions of `X`, generated by randomly permuting or resampling the marks.

Note that, by default, automatic bandwidth selection will be performed separately for each randomised pattern. This computation can be very time-consuming but is necessary for the test to be valid in most conditions. A short-cut is to specify the value of the smoothing bandwidth `sigma` as shown in the examples.

Value

An object of class `"htest"` representing the result of the test.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

Bithell, J.F. (1991) Estimation of relative risk functions. *Statistics in Medicine* **10**, 1745–1751.

Kelsall, J.E. and Diggle, P.J. (1995) Kernel estimation of relative risk. *Bernoulli* **1**, 3–16.

Diggle, P.J., Zheng, P. and Durr, P. (2005) Non-parametric estimation of spatial segregation in a multivariate point process: bovine tuberculosis in Cornwall, UK. *Applied Statistics* **54**, 645–658.

See Also

relrisk

Examples

```
segregation.test(hyytiala, 5)

if(interactive()) segregation.test(hyytiala, hmin=0.05)
```

---

sharpen *Data Sharpening of Point Pattern*

Description

Performs Choi-Hall data sharpening of a spatial point pattern.

Usage

```
sharpen(X, ...)
## S3 method for class 'ppp'
sharpen(X, sigma=NULL, ...,
                      varcov=NULL, edgecorrect=FALSE)
```

Arguments

X	A marked point pattern (object of class "ppp").
sigma	Standard deviation of isotropic Gaussian smoothing kernel.
varcov	Variance-covariance matrix of anisotropic Gaussian kernel. Incompatible with sigma.
edgecorrect	Logical value indicating whether to apply edge effect bias correction.
...	Arguments passed to density.ppp to control the pixel resolution of the result.

Details

Choi and Hall (2001) proposed a procedure for *data sharpening* of spatial point patterns. This procedure is appropriate for earthquake epicentres and other point patterns which are believed to exhibit strong concentrations of points along a curve. Data sharpening causes such points to concentrate more tightly along the curve.

If the original data points are $X_1, \ldots, X_n$ then the sharpened points are

$$\hat{X}_i = \frac{\sum_j X_j k(X_j - X_i)}{\sum_j k(X_j - X_i)}$$

where k is a smoothing kernel in two dimensions. Thus, the new point $\hat{X}_i$ is a vector average of the nearby points $X[j]$.

The function sharpen is generic. It currently has only one method, for two-dimensional point patterns (objects of class "ppp").

If sigma is given, the smoothing kernel is the isotropic two-dimensional Gaussian density with standard deviation sigma in each axis. If varcov is given, the smoothing kernel is the Gaussian density with variance-covariance matrix varcov.

The data sharpening procedure tends to cause the point pattern to contract away from the boundary of the window. That is, points X_i that lie 'quite close to the edge of the window of the point pattern tend to be displaced inward. If edgecorrect=TRUE then the algorithm is modified to correct this vector bias.

Value

A point pattern (object of class "ppp") in the same window as the original pattern X, and with the same marks as X.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>

References

Choi, E. and Hall, P. (2001) Nonparametric analysis of earthquake point-process data. In M. de Gunst, C. Klaassen and A. van der Vaart (eds.) *State of the art in probability and statistics: Festschrift for Willem R. van Zwet*, Institute of Mathematical Statistics, Beachwood, Ohio. Pages 324–344.

See Also

density.ppp, Smooth.ppp.

Examples

```
X <- unmark(shapley)

Y <- sharpen(X, sigma=0.5)
Z <- sharpen(X, sigma=0.5, edgecorrect=TRUE)
opa <- par(mar=rep(0.2, 4))
plot(solist(X, Y, Z), main= " ",
     main.panel=c("data", "sharpen", "sharpen, correct"),
     pch=".", equal.scales=TRUE, mar.panel=0.2)
par(opa)
```

Smooth *Spatial smoothing of data*

Description

Generic function to perform spatial smoothing of spatial data.

Usage

```
Smooth(X, ...)
```

Arguments

X	Some kind of spatial data
...	Arguments passed to methods.

Details

This generic function calls an appropriate method to perform spatial smoothing on the spatial dataset X.

Methods for this function include

- Smooth.ppp for point patterns
- Smooth.msr for measures
- Smooth.fv for function value tables

Value

An object containing smoothed values of the input data, in an appropriate format. See the documentation for the methods.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

Smooth.ppp, Smooth.im, Smooth.msr, Smooth.fv.

Smooth.fv *Apply Smoothing to Function Values*

Description

Applies smoothing to the values in selected columns of a function value table.

Usage

```
## S3 method for class 'fv'
Smooth(X, which = "*", ...,
          method=c("smooth.spline", "loess"),
          xinterval=NULL)
```

Arguments

X	Values to be smoothed. A function value table (object of class "fv", see fv.object).
which	Character vector identifying which columns of the table should be smoothed. Either a vector containing names of columns, or one of the wildcard strings "*" or "." explained below.
...	Extra arguments passed to smooth.spline or loess to control the smoothing.
method	Smoothing algorithm. A character string, partially matched to either "smooth.spline" or "loess".
xinterval	Optional. Numeric vector of length 2 specifying a range of x values. Smoothing will be performed only on the part of the function corresponding to this range.

Details

The command Smooth.fv applies smoothing to the function values in a function value table (object of class "fv").

Smooth.fv is a method for the generic function Smooth.

The smoothing is performed either by smooth.spline or by loess.

Smoothing is applied to every column (or to each of the selected columns) of function values in turn, using the function argument as the x coordinate and the selected column as the y coordinate. The original function values are then replaced by the corresponding smooth interpolated function values.

The optional argument which specifies which of the columns of function values in X will be smoothed. The default (indicated by the wildcard which="*") is to smooth all function values, i.e.\ all columns

except the function argument. Alternatively `which="."` designates the subset of function values that are displayed in the default plot. Alternatively `which` can be a character vector containing the names of columns of `x`.

If the argument `xinterval` is given, then smoothing will be performed only in the specified range of x values.

Value

Another function value table (object of class `"fv"`) of the same format.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

Smooth, with.fv, fv.object, smooth.spline, smooth.spline

Examples

```
G <- Gest(cells)
plot(G)
plot(Smooth(G, df=9), add=TRUE)
```

Smooth.ppp *Spatial smoothing of observations at irregular points*

Description

Performs spatial smoothing of numeric values observed at a set of irregular locations. Uses kernel smoothing and least-squares cross-validated bandwidth selection.

Usage

```
## S3 method for class 'ppp'
Smooth(X, sigma=NULL,
                    ...,
                    weights = rep(1, npoints(X)),
                    at = "pixels", leaveoneout=TRUE,
                    adjust = 1, varcov = NULL,
                    edge = TRUE, diggle = FALSE,
                    kernel = "gaussian",
                    scalekernel = is.character(kernel),
                    se = FALSE,
                    loctype = c("random", "fixed"),
                    wtype = c("multiplicity", "importance"),
                    geometric = FALSE,
                    shrink=0, shrinktype=c("mean", "median"))
```

```
markmean(X, ...)

markvar(X, sigma=NULL, ..., weights=NULL, varcov=NULL)
```

Arguments

X	A marked point pattern (object of class "ppp").
sigma	Smoothing bandwidth. A single positive number, a numeric vector of length 2, or a function that selects the bandwidth automatically. See density.ppp.
...	Further arguments passed to bw.smoothppp and density.ppp to control the kernel smoothing and the pixel resolution of the result.
weights	Optional weights attached to the observations. A numeric vector, a function(x,y), a pixel image, or an expression. See density.ppp.
at	String specifying whether to compute the smoothed values at a grid of pixel locations (at="pixels") or only at the points of X (at="points").
leaveoneout	Logical value indicating whether to compute a leave-one-out estimator. Applicable only when at="points".
edge, diggle	Arguments passed to density.ppp to determine the edge correction.
adjust	Optional. Adjustment factor for the bandwidth sigma.
varcov	Variance-covariance matrix. An alternative to sigma. See density.ppp.
kernel	The smoothing kernel. A character string specifying the smoothing kernel (current options are "gaussian", "epanechnikov", "quartic" or "disc"), or a pixel image (object of class "im") containing values of the kernel, or a function(x,y) which yields values of the kernel.
scalekernel	Logical value. If scalekernel=TRUE, then the kernel will be rescaled to the bandwidth determined by sigma and varcov: this is the default behaviour when kernel is a character string. If scalekernel=FALSE, then sigma and varcov will be ignored: this is the default behaviour when kernel is a function or a pixel image.
se	Logical value specifying whether to calculate standard errors. This calculation is experimental.
loctype	Character string (partially matched) specifying whether the point locations are assumed to be fixed or random, in the calculation of standard error. Experimental.
wtype	Character string (partially matched) specifying whether the weights should be interpreted as multiplicities or as importance weights, in the calculation of standard error. Experimental.
geometric	Logical value indicating whether to perform geometric mean smoothing instead of arithmetic mean smoothing. See Details.
shrink, shrinktype	Experimental. Do Not Use.

Details

The function Smooth.ppp performs spatial smoothing of numeric values observed at a set of irregular locations. The functions markmean and markvar are wrappers for Smooth.ppp which compute the spatially-varying mean and variance of the marks of a point pattern.

Smooth.ppp is a method for the generic function Smooth for the class "ppp" of point patterns. Thus you can type simply Smooth(X).

Smoothing is performed by kernel weighting, using the Gaussian kernel by default. If the observed values are $v_1, \ldots, v_n$ at locations $x_1, \ldots, x_n$ respectively, then the smoothed value at a location u is (ignoring edge corrections)

$$g(u) = \frac{\sum_i k(u - x_i) v_i}{\sum_i k(u - x_i)}$$

where k is the kernel (a Gaussian kernel by default). This is known as the Nadaraya-Watson smoother (Nadaraya, 1964, 1989; Watson, 1964). By default, the smoothing kernel bandwidth is chosen by least squares cross-validation (see below).

The argument X must be a marked point pattern (object of class "ppp", see ppp.object). The points of the pattern are taken to be the observation locations x_i, and the marks of the pattern are taken to be the numeric values v_i observed at these locations.

The marks are allowed to be a data frame (in Smooth.ppp and markmean). Then the smoothing procedure is applied to each column of marks.

The numerator and denominator are computed by density.ppp. The arguments ... control the smoothing kernel parameters and determine whether edge correction is applied. The smoothing kernel bandwidth can be specified by either of the arguments sigma or varcov which are passed to density.ppp. If neither of these arguments is present, then by default the bandwidth is selected by least squares cross-validation, using bw.smoothppp.

The optional argument weights allows numerical weights to be applied to the data. If a weight w_i is associated with location x_i, then the smoothed function is (ignoring edge corrections)

$$g(u) = \frac{\sum_i k(u - x_i) v_i w_i}{\sum_i k(u - x_i) w_i}$$

If geometric=TRUE then geometric mean smoothing is performed instead of arithmetic mean smoothing. The mark values must be non-negative numbers. The logarithm of the mark values is computed; these logarithmic values are kernel-smoothed as described above; then the exponential function is applied to the smoothed values.

An alternative to kernel smoothing is inverse-distance weighting, which is performed by idw.

Value

If X *has a single column of marks:*

- If at="pixels" (the default), the result is a pixel image (object of class "im"). Pixel values are values of the interpolated function.
- If at="points", the result is a numeric vector of length equal to the number of points in X. Entries are values of the interpolated function at the points of X.

If X *has a data frame of marks:*

- If at="pixels" (the default), the result is a named list of pixel images (object of class "im"). There is one image for each column of marks. This list also belongs to the class "solist", for which there is a plot method.
- If at="points", the result is a data frame with one row for each point of X, and one column for each column of marks. Entries are values of the interpolated function at the points of X.

The return value has attributes "sigma" and "varcov" which report the smoothing bandwidth that was used.

Very small bandwidth

If the chosen bandwidth sigma is very small, kernel smoothing is mathematically equivalent to nearest-neighbour interpolation; the result will be computed by nnmark with ties resolved by taking the average mark. This is unless at="points" and leaveoneout=FALSE, when the original mark values are returned.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

Nadaraya, E.A. (1964) On estimating regression. *Theory of Probability and its Applications* **9**, 141–142.

Nadaraya, E.A. (1989) *Nonparametric estimation of probability densities and regression curves*. Kluwer, Dordrecht.

Watson, G.S. (1964) Smooth regression analysis. *Sankhya A* **26**, 359–372.

See Also

Smooth,

density.ppp, bw.smoothppp, nnmark, ppp.object, im.object.

See idw for inverse-distance weighted smoothing.

To perform interpolation, see also the akima package.

Examples

```
# Longleaf data - tree locations, marked by tree diameter
# Local smoothing of tree diameter (automatic bandwidth selection)
Z <- Smooth(longleaf)
# Kernel bandwidth sigma=5
plot(Smooth(longleaf, 5))
# mark variance
plot(markvar(longleaf, sigma=5))
# data frame of marks: trees marked by diameter and height
plot(Smooth(finpines, sigma=2))
head(Smooth(finpines, sigma=2, at="points"))
```

`Smooth.ssf` *Smooth a Spatially Sampled Function*

Description

Applies kernel smoothing to a spatially sampled function.

Usage

```
 ## S3 method for class 'ssf'
Smooth(X, ...)
```

Arguments

`X`	Object of class `"ssf"`.
`...`	Arguments passed to `Smooth.ppp` to control the smoothing.

Details

An object of class `"ssf"` represents a real-valued or vector-valued function that has been evaluated or sampled at an irregular set of points.

The function values will be smoothed using a Gaussian kernel.

Value

A pixel image or a list of pixel images.

Author(s)

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

See Also

`ssf`, `Smooth.ppp`

Examples

```
f <- ssf(redwood, nndist(redwood))
Smooth(f, sigma=0.1)
```

Smoothfun.ppp *Smooth Interpolation of Marks as a Spatial Function*

Description

Perform spatial smoothing of numeric values observed at a set of irregular locations, and return the result as a function of spatial location.

Usage

```
Smoothfun(X, ...)

## S3 method for class 'ppp'
Smoothfun(X, sigma = NULL, ...,
                        weights = NULL, edge = TRUE, diggle = FALSE)
```

Arguments

X	Marked point pattern (object of class "ppp").
sigma	Smoothing bandwidth, or bandwidth selection function, passed to Smooth.ppp.
...	Additional arguments passed to Smooth.ppp.
weights	Optional vector of weights associated with the points of X.
edge, diggle	Logical arguments controlling the edge correction. Arguments passed to Smooth.ppp.

Details

The commands Smoothfun and Smooth both perform kernel-smoothed spatial interpolation of numeric values observed at irregular spatial locations. The difference is that Smooth returns a pixel image, containing the interpolated values at a grid of locations, while Smoothfun returns a function(x,y) which can be used to compute the interpolated value at *any* spatial location. For purposes such as model-fitting it is more accurate to use Smoothfun to interpolate data.

Value

A function with arguments x,y. The function also belongs to the class "Smoothfun" which has methods for print and as.im. It also belongs to the class "funxy" which has methods for plot, contour and persp.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

Smooth

Examples

```
f <- Smoothfun(longleaf)
f
f(120, 80)
plot(f)
```

SmoothHeat *Spatial Smoothing of Data by Diffusion*

Description

Generic function to perform spatial smoothing of spatial data by diffusion.

Usage

```
SmoothHeat(X, ...)
```

Arguments

X	Some kind of spatial data
...	Arguments passed to methods.

Details

This generic function calls an appropriate method to perform spatial smoothing on the spatial dataset X using diffusion.

Methods for this function include

- SmoothHeat.ppp for point patterns
- SmoothHeat.im for pixel images.

Value

An object containing smoothed values of the input data, in an appropriate format. See the documentation for the methods.

Author(s)

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

See Also

SmoothHeat.ppp, SmoothHeat.im.

SmoothHeat.ppp *Spatial Smoothing of Observations using Diffusion Estimate of Density*

Description

Performs spatial smoothing of numeric values observed at a set of irregular locations, using the diffusion estimate of the density.

Usage

```
## S3 method for class 'ppp'
SmoothHeat(X, sigma, ..., weights=NULL)
```

Arguments

X	Point pattern (object of class "ppp") with numeric marks.
sigma	Smoothing bandwidth. A single number giving the equivalent standard deviation of the smoother.
...	Arguments passed to densityHeat controlling the estimation of each marginal intensity, or passed to pixellate.ppp controlling the pixel resolution.
weights	Optional numeric vector of weights associated with each data point.

Details

This is the analogue of the Nadaraya-Watson smoother, using the diffusion smoothing estimation procedure (Baddeley et al, 2022). The numerator and denominator of the Nadaraya-Watson smoother are calculated using densityHeat.ppp.

Value

Pixel image (object of class "im") giving the smoothed mark value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Tilman Davies <Tilman.Davies@otago.ac.nz> and Suman Rakshit.

References

Baddeley, A., Davies, T., Rakshit, S., Nair, G. and McSwiggan, G. (2022) Diffusion smoothing for spatial point patterns. *Statistical Science* **37**, 123–142.

See Also

Smooth.ppp for the usual kernel-based smoother (the Nadaraya-Watson smoother) and densityHeat for the diffusion estimate of density.

Examples

```
plot(SmoothHeat(longleaf, 10))
```

---

spatcov *Estimate the Spatial Covariance Function of a Random Field*

Description

Given a pixel image, calculate an estimate of the spatial covariance function. Given two pixel images, calculate an estimate of their spatial cross-covariance function.

Usage

```
spatcov(X, Y=X, ..., correlation=FALSE, isotropic = TRUE,
        clip = TRUE, pooling=TRUE)
```

Arguments

X	A pixel image (object of class "im").
Y	Optional. Another pixel image.
correlation	Logical value specifying whether to standardise so that the spatial correlation function is returned.
isotropic	Logical value specifying whether to assume the covariance is isotropic, so that the result is a function of the lag distance.
clip	Logical value specifying whether to restrict the results to the range of spatial lags where the estimate is reliable.
pooling	Logical value specifying the estimation method when isotropic=TRUE.
...	Ignored.

Details

In normal usage, only the first argument X is given. Then the pixel image X is treated as a realisation of a stationary random field, and its spatial covariance function is estimated.

Alternatively if Y is given, then X and Y are assumed to be jointly stationary random fields, and their spatial cross-covariance function is estimated.

For any random field X, the spatial covariance is defined for any two spatial locations u and v by

$$C(u, v) = \mathrm{cov}(X(u), X(v))$$

where $X(u)$ and $X(v)$ are the values of the random field at those locations. Herecov denotes the statistical covariance, defined for any random variables A and B by $\mathrm{cov}(A, B) = E(AB) - E(A)E(B)$ where $E(A)$ denotes the expected value of A.

If the random field is assumed to be stationary (at least second-order stationary) then the spatial covariance $C(u, v)$ depends only on the lag vector $v - u$:

$$C(u, v) = C_2(v - u)$$

$$C(u, v) = C2(v - u)$$

where C_2 is a function of a single vector argument.

If the random field is stationary and isotropic, then the spatial covariance depends only on the lag distance $\|v - u\|$:

$$C_2(v - u) = C_1(\|v - u\|)$$

where C_1 is a function of distance.

The function `spatcov` computes estimates of the covariance function C_1 or C_2 as follows:

- If `isotropic=FALSE`, an estimate of the covariance function C_2 is computed, assuming the random field is stationary, using the naive moment estimator, `C2 = imcov(X-mean(X))/setcov(Window(X))`. The result is a pixel image.
- If `isotropic=TRUE` (the default) an estimate of the covariance function C_1 is computed, assuming the random field is stationary and isotropic.
  - When `pooling=FALSE`, the estimate of C_1 is the rotational average of the naive estimate of C_2.
  - When `pooling=TRUE` (the default), the estimate of C_1 is the ratio of the rotational averages of the numerator and denominator which form the naive estimate of C_2.

  The result is a function object (class `"fv"`).

If the argument `Y` is given, it should be a pixel image compatible with `X`. An estimate of the spatial cross-covariance function between `X` and `Y` will be computed.

Value

If `isotropic=TRUE` (the default), the result is a function value table (object of class `"fv"`) giving the estimated values of the covariance function or spatial correlation function for a sequence of values of the spatial lag distance `r`.

If `isotropic=FALSE`, the result is a pixel image (object of class `"im"`) giving the estimated values of the spatial covariance function or spatial correlation function for a grid of values of the spatial lag vector.

Author(s)

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

See Also

`imcov`, `setcov`

Examples

```
if(offline <- !interactive()) op <- spatstat.options(npixel=32)

  D <- density(cells)
  plot(spatcov(D))

if(offline) spatstat.options(op)
```

spatialcdf *Spatial Cumulative Distribution Function*

Description

Compute the spatial cumulative distribution function of a spatial covariate, optionally using spatially-varying weights.

Usage

```
spatialcdf(Z, weights = NULL, normalise = FALSE, ..., W = NULL, Zname = NULL)
```

Arguments

Z	Spatial covariate. A pixel image or a function(x,y,...)
weights	Spatial weighting for different locations. A pixel image, a function(x,y,...), a window, a constant value, or a fitted point process model (object of class "ppm" or "kppm").
normalise	Logical. Whether the weights should be normalised so that they sum to 1.
...	Arguments passed to as.mask to determine the pixel resolution, or extra arguments passed to Z if it is a function.
W	Optional window (object of class "owin") defining the spatial domain.
Zname	Optional character string for the name of the covariate Z used in plots.

Details

If weights is missing or NULL, it defaults to 1. The values of the covariate Z are computed on a grid of pixels. The weighted cumulative distribution function of Z values is computed, taking each value with weight equal to the pixel area. The resulting function F is such that $F(t)$ is the area of the region of space where $Z \leq t$.

If weights is a pixel image or a function, then the values of weights and of the covariate Z are computed on a grid of pixels. The weights are multiplied by the pixel area. Then the weighted empirical cumulative distribution function of Z values is computed using ewcdf. The resulting function F is such that $F(t)$ is the total weight (or weighted area) of the region of space where $Z \leq t$.

If weights is a fitted point process model, then it should be a Poisson process. The fitted intensity of the model, and the value of the covariate Z, are evaluated at the quadrature points used to fit the model. The weights are multiplied by the weights of the quadrature points. Then the weighted empirical cumulative distribution of Z values is computed using ewcdf. The resulting function F is such that $F(t)$ is the expected number of points in the point process that will fall in the region of space where $Z \leq t$.

If normalise=TRUE, the function is normalised so that its maximum value equals 1, so that it gives the cumulative *fraction* of weight or cumulative fraction of points.

The result can be printed, plotted, and used as a function.

Value

A cumulative distribution function object belonging to the classes "spatialcdf", "ewcdf", "ecdf" (only if normalise=TRUE) and "stepfun".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

ewcdf, cdf.test

Examples

```
 with(bei.extra, {
   plot(spatialcdf(grad))
   if(require("spatstat.model")) {
     fit <- ppm(bei ~ elev)
     plot(spatialcdf(grad, predict(fit)))
     A <- spatialcdf(grad, fit)
     A(0.1)
   }
})
plot(spatialcdf("x", W=letterR))
```

SpatialMedian.ppp *Spatially Weighted Median of Values at Points*

Description

Given a spatial point pattern with numeric marks, compute a weighted median of the mark values, with spatially-varying weights that depend on distance to the data points.

Usage

```
 ## S3 method for class 'ppp'
SpatialMedian(X, sigma = NULL, ...,
        type = 4, at = c("pixels", "points"), leaveoneout = TRUE,
        weights = NULL, edge = TRUE, diggle = FALSE, verbose = FALSE)
```

Arguments

X	A spatial point pattern (object of class "ppp") with numeric marks.
sigma	Smoothing bandwidth, passed to density.ppp.
...	Further arguments passed to density.ppp controlling the spatial smoothing.

type	Integer specifying the type of median (using the convention of quantile.default; see Details). Only types 1 and 4 are currently implemented.
at	Character string indicating whether to compute the median at every pixel of a pixel image (at="pixels", the default) or at every data point of X (at="points").
leaveoneout	Logical value indicating whether to compute a leave-one-out estimator. Applicable only when at="points".
weights	Optional vector of numeric weights attached to the points of X.
edge, diggle	Arguments passed to density.ppp to determine the edge correction.
verbose	Logical value specifying whether to print progress reports during the calculation.

Details

The argument X should be a spatial point pattern (object of class "ppp") with numeric marks.

The algorithm computes the weighted median of the mark values at each desired spatial location, using spatially-varying weights which depend on distance to the data points.

Suppose the data points are at spatial locations $x_1, \ldots, x_n$ and have mark values $y_1, \ldots, y_n$. For a query location u, the smoothed median is defined as the weighted median of the mark values $y_1, \ldots, y_n$ with weights $w_1, \ldots, w_n$, where

$$w_i = \frac{k(u, x_i)}{\sum_{j=1}^{n} k(u, x_j)}$$

where $k(u, v)$ is the smoothing kernel with bandwidth sigma

If at="points" and leaveoneout=TRUE, then a leave-one-out calculation is performed, which means that when the query location is a data point x_i, the value at the data point is ignored, and the weighted median is computed from the values y_j for all j not equal to i.

Value

If X *has a single column of marks:*

- If at="pixels" (the default), the result is a pixel image (object of class "im").
- If at="points", the result is a numeric vector of length equal to the number of points in X.

If X *has a data frame of marks:*

- If at="pixels" (the default), the result is a named list of pixel images (object of class "im"). There is one image for each column of marks. This list also belongs to the class "solist", for which there is a plot method.
- If at="points", the result is a data frame with one row for each point of X, and one column for each column of marks. Entries are values of the interpolated function at the points of X.

The return value has attributes "sigma" and "varcov" which report the smoothing bandwidth that was used.

The calculation of the median value depends on the argument type which is interpreted in the same way as for quantile.default. Currently, only types 1 and 4 are implemented. If type=1, the median is always one of the mark values (one of the values in marks(x)). If type=4 (the default), the median value is obtained by linearly interpolating between mark values. Note that the default values of type in SpatialMedian.ppp and SpatialQuantile.ppp are different.

Author(s)

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

See Also

Generic function SpatialMedian.

SpatialQuantile and SpatialQuantile.ppp for other quantiles.

Smooth.ppp for the spatially weighted average.

Examples

```
X <- longleaf
if(!interactive()) {
 ## mark values rounded to nearest multiple of 10 to reduce check time
 marks(X) <- round(marks(X), -1)
}
Z <- SpatialMedian(X, sigma=30)
ZX <- SpatialMedian(X, sigma=30, at="points")
```

SpatialQuantile *Spatially Weighted Median or Quantile*

Description

Compute a weighted median or weighted quantile of spatial data.

Usage

```
SpatialMedian(X, ...)

SpatialQuantile(X, prob = 0.5, ...)
```

Arguments

X	A spatial data object.
prob	Probability for which the quantile is required. A single numeric value between 0 and 1. Default is to calculate the median.
...	Further arguments passed to methods.

Details

The functions SpatialMedian and SpatialQuantile are generic. They calculate spatially weighted medians and quantiles of spatial data. The details depend on the class of X.

There are methods for spatial point patterns (class "ppp") and possibly for other objects.

Author(s)

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

See Also

Methods SpatialMedian.ppp, SpatialQuantile.ppp.

Smooth for the spatially weighted average.

SpatialQuantile.ppp *Spatially Weighted Quantile of Values at Points*

Description

Given a spatial point pattern with numeric marks, compute a weighted quantile of the mark values, with spatially-varying weights that depend on distance to the data points.

Usage

```
 ## S3 method for class 'ppp'
SpatialQuantile(X, prob = 0.5, sigma = NULL, ...,
        type = 1, at = c("pixels", "points"), leaveoneout = TRUE,
        weights = NULL, edge = TRUE, diggle = FALSE, verbose = FALSE)
```

Arguments

X	A spatial point pattern (object of class "ppp") with numeric marks.
prob	Probability for which the quantile is required. A single numeric value between 0 and 1.
sigma	Smoothing bandwidth, passed to density.ppp.
...	Further arguments passed to density.ppp controlling the spatial smoothing.
type	Integer specifying the type of quantile (using the convention of quantile.default; see Details). Only types 1 and 4 are currently implemented.
at	Character string indicating whether to compute the quantile at every pixel of a pixel image (at="pixels", the default) or at every data point of X (at="points").
leaveoneout	Logical value indicating whether to compute a leave-one-out estimator. Applicable only when at="points".
weights	Optional vector of numeric weights attached to the points of X.
edge, diggle	Arguments passed to density.ppp to determine the edge correction.
verbose	Logical value specifying whether to print progress reports during the calculation.

Details

The argument X should be a spatial point pattern (object of class "ppp") with numeric marks.

The algorithm computes the weighted quantile of the mark values at each desired spatial location, using spatially-varying weights which depend on distance to the data points.

Suppose the data points are at spatial locations $x_1, \ldots, x_n$ and have mark values $y_1, \ldots, y_n$. For a query location u, the smoothed quantile is defined as the weighted quantile of the mark values $y_1, \ldots, y_n$ with weights $w_1(u), \ldots, w_n(u)$, where

$$w_i(u) = \frac{k(u, x_i)}{\sum_{j=1}^{n} k(u, x_j)}$$

where $k(u, v)$ is the smoothing kernel with bandwidth sigma.

If at="points" and leaveoneout=TRUE, then a leave-one-out calculation is performed, which means that when the query location is a data point x_i, the value at the data point is ignored, and the weighted quantile is computed from the values y_j for all j not equal to i.

The calculation of the quantile value depends on the argument type which is interpreted in the same way as for quantile.default. Currently, only types 1 and 4 are implemented. If type=1 (the default), the quantile value is one of the mark values (one of the values in marks(x)). If type=4, the quantile value is obtained by linearly interpolating between mark values. Note that the default values of type in SpatialQuantile.ppp and SpatialMedian.ppp are different.

Value

If X *has a single column of marks:*

- If at="pixels" (the default), the result is a pixel image (object of class "im").
- If at="points", the result is a numeric vector of length equal to the number of points in X.

If X *has a data frame of marks:*

- If at="pixels" (the default), the result is a named list of pixel images (object of class "im"). There is one image for each column of marks. This list also belongs to the class "solist", for which there is a plot method.
- If at="points", the result is a data frame with one row for each point of X, and one column for each column of marks. Entries are values of the interpolated function at the points of X.

The return value has attributes "sigma" and "varcov" which report the smoothing bandwidth that was used.

Author(s)

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

See Also

SpatialMedian.ppp, SpatialMedian.

Examples

```
X <- longleaf
if(!interactive()) {
 ## mark values rounded to nearest multiple of 10 to reduce check time
 marks(X) <- round(marks(X), -1)
}
Z <- SpatialQuantile(X, prob=0.25, sigma=30)
ZX <- SpatialQuantile(X, prob=0.25, sigma=30, at="points")
```

---

ssf *Spatially Sampled Function*

Description

Create an object that represents a spatial function which has been evaluated or sampled at an irregular set of points.

Usage

```
ssf(loc, val)
```

Arguments

loc	The spatial locations at which the function has been evaluated. A point pattern (object of class "ppp").
val	The function values at these locations. A numeric vector with one entry for each point of loc, or a data frame with one row for each point of loc.

Details

An object of class "ssf" represents a real-valued or vector-valued function that has been evaluated or sampled at an irregular set of points. An example would be a spatial covariate that has only been measured at certain locations.

An object of this class also inherits the class "ppp", and is essentially the same as a marked point pattern, except for the class membership which enables it to be handled in a different way.

There are methods for plot, print etc; see plot.ssf and methods.ssf.

Use unmark to extract only the point locations, and marks.ssf to extract only the function values.

Value

Object of class "ssf".

Author(s)

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

See Also

plot.ssf, methods.ssf, Smooth.ssf, with.ssf, [.ssf.

Examples

```
ssf(cells, nndist(cells, k=1:3))
```

---

stienen *Stienen Diagram*

Description

Draw the Stienen diagram of a point pattern, or compute the region covered by the Stienen diagram.

Usage

```
stienen(X, ..., bg = "grey", border = list(bg = NULL))
stienenSet(X, edge=TRUE)
```

Arguments

X	Point pattern (object of class "ppp").
...	Arguments passed to plot.ppp to control the plot.
bg	Fill colour for circles.
border	Either a list of arguments passed to plot.ppp to control the display of circles at the border of the diagram, or the value FALSE indicating that the border circles should not be plotted.
edge	Logical value indicating whether to include the circles at the border of the diagram.

Details

The Stienen diagram of a point pattern (Stienen, 1982) is formed by drawing a circle around each point of the pattern, with diameter equal to the nearest-neighbour distance for that point. These circles do not overlap. If two points are nearest neighbours of each other, then the corresponding circles touch.

stienenSet(X) computes the union of these circles and returns it as a window (object of class "owin").

stienen(X) generates a plot of the Stienen diagram of the point pattern X. By default, circles are shaded in grey if they lie inside the window of X, and are not shaded otherwise.

Value

The plotting function stienen returns NULL.

The return value of stienenSet is a window (object of class "owin").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

Stienen, H. (1982) *Die Vergroeberung von Karbiden in reinen Eisen-Kohlenstoff Staehlen*. Dissertation, RWTH Aachen.

See Also

nndist, plot.ppp

Examples

```
Y <- stienenSet(cells)
stienen(redwood)
stienen(redwood, border=list(bg=NULL, lwd=2, cols="red"))
```

---

studpermu.test *Studentised Permutation Test*

Description

Perform a studentised permutation test for a difference between groups of point patterns.

Usage

```
studpermu.test(X, formula, summaryfunction = Kest,
    ..., rinterval = NULL, nperm = 999,
     use.Tbar = FALSE, minpoints = 20, rsteps = 128,
     r = NULL, arguments.in.data = FALSE)
```

Arguments

X	Data. Either a hyperframe or a list of lists of point patterns.
formula	Formula describing the grouping, when X is a hyperframe. The left side of the formula identifies which column of X contains the point patterns. The right side identifies the grouping factor. If the formula is missing, the grouping variable is taken to be the first column of X that contains a factor, and the point patterns are taken from the first column that contains point patterns.
summaryfunction	Summary function applicable to point patterns.
...	Additional arguments passed to summaryfunction.

`rinterval` Interval of distance values r over which the summary function should be evaluated and over which the test statistic will be integrated. If `NULL`, the default range of the summary statistic is used (taking the intersection of these ranges over all patterns).

`nperm` Number of random permutations for the test.

`use.Tbar` Logical value indicating choice of test statistic. If `TRUE`, use the alternative test statistic, which is appropriate for summary functions with roughly constant variance, such as $K(r)/r$ or $L(r)$.

`minpoints` Minimum permissible number of points in a point pattern for inclusion in the test calculation.

`rsteps` Number of discretisation steps in the `rinterval`.

`r` Optional vector of distance values as the argument for `summaryfunction`. Should not usually be given. There is a sensible default.

`arguments.in.data` Logical. If `TRUE`, individual extra arguments to `summaryfunction` will be taken from `X` (which must be a hyperframe). This assumes that the first argument of `summaryfunction` is the point pattern dataset.

Details

This function performs the studentized permutation test of Hahn (2012) for a difference between groups of point patterns.

The first argument `X` should be either

a list of lists of point patterns. Each element of `X` will be interpreted as a group of point patterns, assumed to be replicates of the same point process.

a hyperframe: One column of the hyperframe should contain point patterns, and another column should contain a factor indicating the grouping. The argument `formula` should be a formula in the R language specifying the grouping: it should be of the form `P ~ G` where `P` is the name of the column of point patterns, and `G` is the name of the factor.

A group needs to contain at least two point patterns with at least `minpoints` points in each pattern.

The function returns an object of class `"htest"` and `"studpermutest"` that can be printed and plotted. The printout shows the test result and p-value. The plot shows the summary functions for the groups (and the group means if requested).

Value

Object of class `"studpermutest"`.

Author(s)

Ute Hahn.

Modified for `spatstat` by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

Hahn, U. (2012) A studentized permutation test for the comparison of spatial point patterns. *Journal of the American Statistical Association* **107** (498), 754–764.

See Also

`plot.studpermutest`

Examples

```
np <- if(interactive()) 99 else 19
testpyramidal <- studpermu.test(pyramidal, Neurons ~ group, nperm=np)
testpyramidal
```

`subspaceDistance` *Distance Between Linear Spaces*

Description

Evaluate the distance between two linear subspaces using the measure proposed by Li, Zha and Chiaromonte (2005).

Usage

```
subspaceDistance(B0, B1)
```

Arguments

`B0`	Matrix whose columns are a basis for the first subspace.
`B1`	Matrix whose columns are a basis for the second subspace.

Details

This algorithm calculates the maximum absolute value of the eigenvalues of $P1 - P0$ where $P0, P1$ are the projection matrices onto the subspaces generated by `B0,B1`. This measure of distance was proposed by Li, Zha and Chiaromonte (2005). See also Xia (2007).

Value

A single numeric value.

Author(s)

Matlab original by Yongtao Guan, translated to R by Suman Rakshit.

References

Guan, Y. and Wang, H. (2010) Sufficient dimension reduction for spatial point processes directed by Gaussian random fields. *Journal of the Royal Statistical Society, Series B*, **72**, 367–387.

Li, B., Zha, H. and Chiaromonte, F. (2005) Contour regression: a general approach to dimension reduction. *Annals of Statistics* **33**, 1580–1616.

Xia, Y. (2007) A constructive approach to the estimation of dimension reduction directions. *Annals of Statistics* **35**, 2654–2690.

`thresholdCI` *Confidence Interval for Threshold of Numerical Predictor*

Description

Given a point pattern and a spatial covariate that has some predictive value for the point pattern, compute a confidence interval for the optimal value of the threshold that should be used to convert the covariate to a binary predictor.

Usage

```
thresholdCI(X, Z, confidence = 0.95, nsim = 1000, parametric = FALSE)
```

Arguments

`X`	Point pattern (object of class `"ppp"`).
`Z`	Spatial covariate with numerical values. Either a pixel image (object of class `"im"`), a distance function (object of class `"distfun"`) or a `function(x,y)` in the R language.
`confidence`	Confidence level. A number between 0 and 1.
`nsim`	Number of bootstrap simulations to perform.
`parametric`	Logical value specifying whether to use the parametric bootstrap.

Details

The spatial covariate `Z` is assumed to have some utility as a predictor of the point pattern `X`.

This code computes a bootstrap confidence interval for the best threshold value z for converting the numerical predictor to a binary predictor, for use in techniques such as Weights of Evidence.

Value

A matrix containing upper and lower limits for the threshold `z` and the corresponding upper and lower limits for the fraction of area of the study region.

Author(s)

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

References

Baddeley, A., Brown, W., Milne, R.K., Nair, G., Rakshit, S., Lawrence, T., Phatak, A. and Fu, S.C. (2021) Optimal thresholding of predictors in mineral prospectivity analysis. *Natural Resources Research* **30** 923–969.

See Also

thresholdSelect

Examples

```
gold <- rescale(murchison$gold, 1000, "km")
faults <- rescale(murchison$faults, 1000, "km")
distfault <- distfun(faults)
Nsim <- if(interactive()) 250 else 25
thresholdCI(gold, distfault, nsim=Nsim)
```

---

thresholdSelect *Select Threshold to Convert Numerical Predictor to Binary Predictor*

Description

Given a point pattern and a spatial covariate that has some predictive value for the point pattern, determine the optimal value of the threshold for converting the covariate to a binary predictor.

Usage

```
thresholdSelect(X, Z, method = c("Y", "LL", "AR", "t", "C"), Zname)
```

Arguments

X	Point pattern (object of class "ppp").
Z	Spatial covariate with numerical values. Either a pixel image (object of class "im"), a distance function (object of class "distfun") or a function(x,y) in the R language.
method	Character string (partially matched) specifying the method to be used to select the optimal threshold value. See Details.
Zname	Optional character string giving a short name for the covariate.

Details

The spatial covariate Z is assumed to have some utility as a predictor of the point pattern X.

This code chooses the best threshold value v for converting the numerical predictor Z to a binary predictor, for use in techniques such as Weights of Evidence.

The best threshold is selected by maximising the criterion specified by the argument method. Options are:

- method="Y" (the default): the Youden criterion
- method="LL": log-likelihood
- method="AR": the Akman-Raftery criterion
- method="t": the Studentised Weights-of-Evidence contrast
- method="C": the Weights-of-Evidence contrast

These criteria are explained in Baddeley et al (2021).

Value

A single numerical value giving the selected bandwidth. The result also belongs to the class "bw.optim" (see bw.optim.object) which can be plotted to show the criterion used to select the threshold.

Author(s)

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

References

Baddeley, A., Brown, W., Milne, R.K., Nair, G., Rakshit, S., Lawrence, T., Phatak, A. and Fu, S.C. (2021) Optimal thresholding of predictors in mineral prospectivity analysis. *Natural Resources Research* **30** 923–969.

See Also

thresholdCI

Examples

```
gold <- rescale(murchison$gold, 1000, "km")
faults <- rescale(murchison$faults, 1000, "km")
distfault <- distfun(faults)
z <- thresholdSelect(gold, distfault)
z
plot(z, xlim=c(0, 20))
```

---

transect.im *Pixel Values Along a Transect*

Description

Extract the pixel values of a pixel image at each point along a linear transect.

Usage

```
transect.im(X, ..., from="bottomleft", to="topright",
            nsample=512, click=FALSE, add=FALSE, curve=NULL)
```

Arguments

X	A pixel image (object of class "im").
...	Ignored.
from, to	Optional. Start point and end point of the transect. Pairs of (x, y) coordinates in a format acceptable to xy.coords, or keywords "bottom", "left", "top", "right", "bottomleft" etc.
nsample	Integer. Number of sample locations along the transect.
click	Optional. Logical value. If TRUE, the linear transect is determined interactively by the user, who clicks two points on the current plot.
add	Logical. If click=TRUE, this argument determines whether to perform interactive tasks on the current plot (add=TRUE) or to start by plotting X (add=FALSE).
curve	Optional. A specification of a curved transect. See the section on Curved Transect.

Details

The pixel values of the image X along a line segment will be extracted. The result is a function table ("fv" object) which can be plotted directly.

If click=TRUE, then the user is prompted to click two points on the plot of X. These endpoints define the transect.

Otherwise, the transect is defined by the endpoints from and to. The default is a diagonal transect from bottom left to top right of the frame.

Value

An object of class "fv" which can be plotted.

Curved Transect

If curve is given, then the transect will be a curve. The argument curve should be a list with the following arguments:

f A function in the R language with one argument t.

tlim A numeric vector of length 2 giving the range of values of the argument t.

tname (Optional) a character string giving the symbolic name of the function argument t; defaults to "t".

tdescrip (Optional) a character string giving a short description of the function argument t; defaults to "curve parameter".

The function f must return a 2-column matrix or data frame specifying the spatial coordinates (x,y) of locations along the curve, determined by the values of the input argument t.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

im

Examples

```
Z <- bei.extra$elev
plot(transect.im(Z))
```

Tstat *Third order summary statistic*

Description

Computes the third order summary statistic $T(r)$ of a spatial point pattern.

Usage

```
Tstat(X, ..., r = NULL, rmax = NULL,
    correction = c("border", "translate"), ratio = FALSE, verbose=TRUE)
```

Arguments

X	The observed point pattern, from which an estimate of $T(r)$ will be computed. An object of class "ppp", or data in any format acceptable to as.ppp().
...	Ignored.
r	Optional. Vector of values for the argument r at which $T(r)$ should be evaluated. Users are advised *not* to specify this argument; there is a sensible default.
rmax	Optional. Numeric. The maximum value of r for which $T(r)$ should be estimated.
correction	Optional. A character vector containing any selection of the options "none", "border", "bord.modif", "translate", "translation", or "best". It specifies the edge correction(s) to be applied. Alternatively correction="all" selects all options.
ratio	Logical. If TRUE, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.
verbose	Logical. If TRUE, an estimate of the computation time is printed.

Details

This command calculates the third-order summary statistic $T(r)$ for a spatial point patterns, defined by Schladitz and Baddeley (2000).

The definition of $T(r)$ is similar to the definition of Ripley's K function $K(r)$, except that $K(r)$ counts pairs of points while $T(r)$ counts triples of points. Essentially $T(r)$ is a rescaled cumulative distribution function of the diameters of triangles in the point pattern. The diameter of a triangle is the length of its longest side.

Value

An object of class "fv", see fv.object, which can be plotted directly using plot.fv.

Computation time

If the number of points is large, the algorithm can take a very long time to inspect all possible triangles. A rough estimate of the total computation time will be printed at the beginning of the calculation. If this estimate seems very large, stop the calculation using the user interrupt signal, and call Tstat again, using rmax to restrict the range of r values, thus reducing the number of triangles to be inspected.

Author(s)

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

References

Schladitz, K. and Baddeley, A. (2000) A third order point process characteristic. *Scandinavian Journal of Statistics* **27** (2000) 657–671.

See Also

Kest

Examples

```
plot(Tstat(redwood))
```

---

varblock *Estimate Variance of Summary Statistic by Subdivision*

Description

This command estimates the variance of any summary statistic (such as the K-function) by spatial subdivision of a single point pattern dataset.

Usage

```
varblock(X, fun = Kest,
         blocks = quadrats(X, nx = nx, ny = ny),
         ...,
         nx = 3, ny = nx,
         confidence=0.95)
```

Arguments

X	Point pattern dataset (object of class "ppp").
fun	Function that computes the summary statistic.
blocks	Optional. A tessellation that specifies the division of the space into blocks.
...	Arguments passed to fun.
nx, ny	Optional. Number of rectangular blocks in the x and y directions. Incompatible with blocks.
confidence	Confidence level, as a fraction between 0 and 1.

Details

This command computes an estimate of the variance of the summary statistic fun(X) from a single point pattern dataset X using a subdivision method. It can be used to plot **confidence intervals** for the true value of a summary function such as the K-function.

The window containing X is divided into pieces by an nx * ny array of rectangles (or is divided into pieces of more general shape, according to the argument blocks if it is present). The summary statistic fun is applied to each of the corresponding sub-patterns of X as described below. Then the pointwise sample mean, sample variance and sample standard deviation of these summary statistics are computed. Then pointwise confidence intervals are computed, for the specified level of confidence, defaulting to 95 percent.

The variance is estimated by equation (4.21) of Diggle (2003, page 52). This assumes that the point pattern X is stationary. For further details see Diggle (2003, pp 52–53).

The estimate of the summary statistic from each block is computed as follows. For most functions fun, the estimate from block B is computed by finding the subset of X consisting of points that fall inside B, and applying fun to these points, by calling fun(X[B]).

However if fun is the K-function Kest, or any function which has an argument called domain, the estimate for each block B is computed by calling fun(X, domain=B). In the case of the K-function this means that the estimate from block B is computed by counting pairs of points in which the *first* point lies in B, while the second point may lie anywhere.

Value

A function value table (object of class "fv") that contains the result of fun(X) as well as the sample mean, sample variance and sample standard deviation of the block estimates, together with the upper and lower two-standard-deviation confidence limits.

Errors

If the blocks are too small, there may be insufficient data in some blocks, and the function fun may report an error. If this happens, you need to take larger blocks.

An error message about incompatibility may occur. The different function estimates may be incompatible in some cases, for example, because they use different default edge corrections (typically because the tiles of the tessellation are not the same kind of geometric object as the window of X, or because the default edge correction depends on the number of points). To prevent this, specify the choice of edge correction, in the correction argument to fun, if it has one.

An alternative to varblock is Loh's mark bootstrap lohboot.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

References

Diggle, P.J. (2003) *Statistical analysis of spatial point patterns*, Second edition. Arnold.

See Also

tess, quadrats for basic manipulation.

lohboot for an alternative bootstrap technique.

Examples

```
v <- varblock(amacrine, Kest, nx=4, ny=2)
v <- varblock(amacrine, Kcross, nx=4, ny=2)
if(interactive()) plot(v, iso ~ r, shade=c("hiiso", "loiso"))
```

Window.quadrattest *Extract Window of Spatial Object*

Description

Given a spatial object (such as a point pattern or pixel image) in two dimensions, these functions extract the window in which the object is defined.

Usage

```
 ## S3 method for class 'quadrattest'
Window(X, ...)
```

Arguments

X	A spatial object.
...	Ignored.

Details

These are methods for the generic function Window which extract the spatial window in which the object X is defined.

Value

An object of class "owin" (see owin.object) specifying an observation window.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

Window, Window.ppp, Window.psp.

owin.object

Examples

```
A <- quadrat.test(cells, 4)
Window(A)
```

---

with.fv *Evaluate an Expression in a Function Table*

Description

Evaluate an R expression in a function value table (object of class "fv").

Usage

```
## S3 method for class 'fv'
with(data, expr, ..., fun = NULL, enclos=NULL)
```

Arguments

data	A function value table (object of class "fv") in which the expression will be evaluated.
expr	The expression to be evaluated. An R language expression, which may involve the names of columns in data, the special abbreviations ., .x and .y, and global constants or functions.
...	Ignored.
fun	Logical value, specifying whether the result should be interpreted as another function (fun=TRUE) or simply returned as a numeric vector or array (fun=FALSE). See Details.
enclos	An environment in which to search for variables that are not found in data. Defaults to parent.frame().

Details

This is a method for the generic command with for an object of class "fv" (function value table).

An object of class "fv" is a convenient way of storing and plotting several different estimates of the same function. It is effectively a data frame with extra attributes. See fv.object for further explanation.

This command makes it possible to perform computations that involve different estimates of the same function. For example we use it to compute the arithmetic difference between two different edge-corrected estimates of the K function of a point pattern.

The argument expr should be an R language expression. The expression may involve

- the name of any column in data, referring to one of the estimates of the function;
- the symbol . which stands for all the available estimates of the function;
- the symbol .y which stands for the recommended estimate of the function (in an "fv" object, one of the estimates is always identified as the recommended estimate);
- the symbol .x which stands for the argument of the function;
- global constants or functions.

See the Examples. The expression should be capable of handling vectors and matrices.

The interpretation of the argument fun is as follows:

- If fun=FALSE, the result of evaluating the expression expr will be returned as a numeric vector, matrix or data frame.
- If fun=TRUE, then the result of evaluating expr will be interpreted as containing the values of a new function. The return value will be an object of class "fv". (This can only happen if the result has the right dimensions.)
- The default is fun=TRUE if the result of evaluating expr has more than one column, and fun=FALSE otherwise.

To perform calculations involving *several* objects of class "fv", use eval.fv.

Value

A function value table (object of class "fv") or a numeric vector or data frame.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

with, fv.object, eval.fv, Kest

Examples

```
# compute 4 estimates of the K function
X <- runifrect(42)
K <- Kest(X)
plot(K)

# derive 4 estimates of the L function L(r) = sqrt(K(r)/pi)
L <- with(K, sqrt(./pi))
plot(L)

# compute 4 estimates of V(r) = L(r)/r
V <- with(L, ./.x)
plot(V)

# compute the maximum absolute difference between
# the isotropic and translation correction estimates of K(r)
D <- with(K, max(abs(iso - trans)))
```

`with.ssf` *Evaluate Expression in a Spatially Sampled Function*

Description

Given a spatially sampled function, evaluate an expression involving the function values.

Usage

```
  apply.ssf(X, ...)

  ## S3 method for class 'ssf'
with(data, ...)
```

Arguments

`X`, `data`	A spatially sampled function (object of class `"ssf"`).
`...`	Arguments passed to `with.default` or `apply` specifying what to compute.

Details

An object of class `"ssf"` represents a function (real- or vector-valued) that has been sampled at a finite set of points. It contains a data frame which provides the function values at the sample points.

In `with.ssf`, the expression specified by `...` will be evaluated in this dataframe. In `apply.ssf`, the dataframe will be subjected to the `apply` operator using the additional arguments `....`

If the result of evaluation is a data frame with one row for each data point, or a numeric vector with one entry for each data point, then the result will be an object of class `"ssf"` containing this information. Otherwise, the result will be a numeric vector.

Value

An object of class `"ssf"` or a numeric vector.

Author(s)

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

See Also

`ssf`

Examples

```
a <- ssf(cells, data.frame(d=nndist(cells), i=1:npoints(cells)))
with(a, i/d)
```

---

youden *Youden Statistic*

Description

Calculate the Youden statistic for an ROC curve.

Usage

```
youden(X, sign = c("positive", "absolute", "negative"))
```

Arguments

`X`	ROC curve (object of class `"roc"` produced by `roc`).
`sign`	Character string indicating which version of the statistic to calculate.

Details

For a receiver operating characteristic (ROC) curve, the Youden statistic is the maximum vertical deviation between the curve and the diagonal line $y = x$.

Suppose $R(p)$ denotes the ROC curve as a function of the horizontal coordinate p. If `sign="positive"` (the default), deviation is defined as the positive part $\max(0, R(p) - p)$. If `sign="absolute"`, deviation is defined as the absolute value $|R(p) - p|$. If `sign="negative"`, deviation is defined as the negative part $\max(0, p - R(p))$. The maximum deviation over all values of p is determined. The result is always nonnegative.

Value

Numeric value or vector, containing nonnegative values.

Author(s)

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

References

Youden, W.J. (1950) Index for rating diagnostic tests. *Cancer* **3**, 32–35.

See Also

roc, auc

Examples

```
a <- with(split(mucosa), roc(ECL, "y", baseline=other, high=FALSE))
youden(a)
```

[.ssf *Subset of spatially sampled function*

Description

Extract a subset of the data for a spatially sampled function.

Usage

```
 ## S3 method for class 'ssf'
x[i, j, ..., drop]
```

Arguments

x	Object of class "ssf".
i	Subset index applying to the locations where the function is sampled.
j	Subset index applying to the columns (variables) measured at each location.
..., drop	Ignored.

Details

This is the subset operator for the class "ssf".

Value

Another object of class "ssf".

Author(s)

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

See Also

ssf, with.ssf

Examples

```
f <- ssf(cells, data.frame(d=nndist(cells), i=1:42))
f
f[1:10,]
f[ ,1]
```

Index