Package 'spatstat.core'

May 18, 2022

Version 2.4-4

Date 2022-05-17

Title Core Functionality of the 'spatstat' Family

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

Depends R (>= 3.5.0), spatstat.data (>= 2.1-0), spatstat.geom (>= 2.4-0), spatstat.random (>= 2.2-0), stats, graphics, grDevices, utils, methods, nlme, rpart

Imports spatstat.utils (>= 2.3-1), spatstat.sparse (>= 2.0-0), mgcv, Matrix, abind, tensor, goftest (>= 1.2-2)

Suggests sm, maptools (>= 0.9-9), gsl, locfit, spatial, RandomFields (>= 3.1.24.1), RandomFieldsUtils(>= 0.3.3.1), fftwtools (>= 0.9-8), nleqslv, spatstat.linnet (>= 2.0-0), spatstat (>= 2.3-3)

Description Functionality for data analysis and modelling 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'.) Exploratory 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.
Parametric models can be fitted to point pattern data using the functions ppm(), kppm(), slrm(), dppm() similar to glm(). Types of models include Poisson, Gibbs and Cox point processes, Neyman-Scott cluster processes, and determinantal point processes. Models may involve dependence on covariates, inter-point interaction, cluster formation and dependence on marks. Models are fitted by maximum likelihood, logistic regression, minimum contrast, and composite likelihood methods.
A model can be fitted to a list of point patterns (replicated point pattern data) using the function mppm(). The model can include random effects and fixed effects depending on the experimental design, in addition to all the features listed above.

Fitted point process models can be simulated, automatically. Formal hypothesis tests of a fitted model are supported (likelihood ratio test, analysis of deviance, Monte Carlo tests) along with basic tools for model selection (stepwise(), AIC()) and variable selection (sdr). Tools for validating the fitted model include simulation envelopes, residuals, residual plots and Q-Q plots, leverage and influence diagnostics, partial residuals, and added variable plots.

License GPL (>= 2)

URL http://spatstat.org/

NeedsCompilation yes

ByteCompile true

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

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

Repository CRAN

Date/Publication 2022-05-18 07:30:02 UTC

R topics documented:

`spatstat.core-package` *The spatstat.core Package*

Description

The **spatstat.core** 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.core** contains all the main user-level functions that perform statistical analysis and modelling 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.geom** containing geometrical objects and geometrical operations
- **spatstat.core** containing the main functionality for statistical analysis and modelling of spatial data
- **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.core

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.core** 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	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
bw.stoyan	Stoyan's rule of thumb for bandwidth for pcf

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

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

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 (Cox and cluster models)

Cluster process models (with homogeneous or inhomogeneous intensity) and Cox processes can be fitted by the function kppm. Its result is an object of class "kppm". The fitted model can be printed, plotted, predicted, simulated and updated.

kppm	Fit model
plot.kppm	Plot the fitted model
summary.kppm	Summarise the fitted model
fitted.kppm	Compute fitted intensity
predict.kppm	Compute fitted intensity
update.kppm	Update the model
improve.kppm	Refine the estimate of trend
simulate.kppm	Generate simulated realisations
vcov.kppm	Variance-covariance matrix of coefficients
coef.kppm	Extract trend coefficients
formula.kppm	Extract trend formula
parameters	Extract all model parameters
clusterfield	Compute offspring density
clusterradius	Radius of support of offspring density
Kmodel.kppm	K function of fitted model
pcfmodel.kppm	Pair correlation of fitted model

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

The theoretical models can also be simulated, for any choice of parameter values, using rThomas, rMatClust, rCauchy, rVarGamma, and rLGCP.

Lower-level fitting functions include:

lgcp.estK	fit a log-Gaussian Cox process model
lgcp.estpcf	fit a log-Gaussian Cox process model
thomas.estK	fit the Thomas process model
thomas.estpcf	fit the Thomas process model
matclust.estK	fit the Matérn Cluster process model
matclust.estpcf	fit the Matérn Cluster process model
cauchy.estK	fit a Neyman-Scott Cauchy cluster process
cauchy.estpcf	fit a Neyman-Scott Cauchy cluster process
vargamma.estK	fit a Neyman-Scott Variance Gamma process
vargamma.estpcf	fit a Neyman-Scott Variance Gamma process
mincontrast	low-level algorithm for fitting models by the method of minimum contrast

Model fitting (Poisson and Gibbs models)

Poisson point processes are the simplest models for point patterns. A Poisson model assumes that the points are stochastically independent. It may allow the points to have a non-uniform spatial density. The special case of a Poisson process with a uniform spatial density is often called Complete Spatial Randomness.

Poisson point processes are included in the more general class of Gibbs point process models. In a Gibbs model, there is *interaction* or dependence between points. Many different types of interaction can be specified.

For a detailed explanation of how to fit Poisson or Gibbs point process models to point pattern data using **spatstat**, see Baddeley and Turner (2005b) or Baddeley (2008).

To fit a Poisson or Gibbs point process model:

Model fitting in **spatstat** is performed mainly by the function `ppm`. Its result is an object of class `"ppm"`.

Here are some examples, where `X` is a point pattern (class `"ppp"`):

command	*model*
`ppm(X)`	Complete Spatial Randomness
`ppm(X ~ 1)`	Complete Spatial Randomness
`ppm(X ~ x)`	Poisson process with intensity loglinear in x coordinate
`ppm(X ~ 1, Strauss(0.1))`	Stationary Strauss process
`ppm(X ~ x, Strauss(0.1))`	Strauss process with conditional intensity loglinear in x

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

Manipulating the fitted model:

`plot.ppm`	Plot the fitted model
`predict.ppm`	Compute the spatial trend and conditional intensity of the fitted point process model
`coef.ppm`	Extract the fitted model coefficients
`parameters`	Extract all model parameters
`formula.ppm`	Extract the trend formula
`intensity.ppm`	Compute fitted intensity
`Kmodel.ppm`	K function of fitted model
`pcfmodel.ppm`	pair correlation of fitted model
`fitted.ppm`	Compute fitted conditional intensity at quadrature points
`residuals.ppm`	Compute point process residuals at quadrature points
`update.ppm`	Update the fit
`vcov.ppm`	Variance-covariance matrix of estimates
`rmh.ppm`	Simulate from fitted model
`simulate.ppm`	Simulate from fitted model
`print.ppm`	Print basic information about a fitted model
`summary.ppm`	Summarise a fitted model
`effectfun`	Compute the fitted effect of one covariate
`logLik.ppm`	log-likelihood or log-pseudolikelihood

anova.ppm	Analysis of deviance
model.frame.ppm	Extract data frame used to fit model
model.images	Extract spatial data used to fit model
model.depends	Identify variables in the model
as.interact	Interpoint interaction component of model
fitin	Extract fitted interpoint interaction
is.hybrid	Determine whether the model is a hybrid
valid.ppm	Check the model is a valid point process
project.ppm	Ensure the model is a valid point process

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

See spatstat.options to control plotting of fitted model.

To specify a point process model:

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

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

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

Poisson()	the Poisson point process
AreaInter()	Area-interaction process
BadGey()	multiscale Geyer process
Concom()	connected component interaction
DiggleGratton()	Diggle-Gratton potential
DiggleGatesStibbard()	Diggle-Gates-Stibbard potential
Fiksel()	Fiksel pairwise interaction process
Geyer()	Geyer's saturation process
Hardcore()	Hard core process
HierHard()	Hierarchical multiype hard core process
HierStrauss()	Hierarchical multiype Strauss process
HierStraussHard()	Hierarchical multiype Strauss-hard core process
Hybrid()	Hybrid of several interactions
LennardJones()	Lennard-Jones potential
MultiHard()	multitype hard core process
MultiStrauss()	multitype Strauss process
MultiStraussHard()	multitype Strauss/hard core process
OrdThresh()	Ord process, threshold potential
Ord()	Ord model, user-supplied potential

PairPiece()	pairwise interaction, piecewise constant
Pairwise()	pairwise interaction, user-supplied potential
Penttinen()	Penttinen pairwise interaction
SatPiece()	Saturated pair model, piecewise constant potential
Saturated()	Saturated pair model, user-supplied potential
Softcore()	pairwise interaction, soft core potential
Strauss()	Strauss process
StraussHard()	Strauss/hard core point process
Triplets()	Geyer triplets process

Note that it is also possible to combine several such interactions using Hybrid.

Simulation and goodness-of-fit for fitted models:

rmh.ppm	simulate realisations of a fitted model
simulate.ppm	simulate realisations of a fitted model
envelope	compute simulation envelopes for a fitted model

Model fitting (determinantal point process models)

Code for fitting *determinantal point process models* has recently been added to **spatstat**.

For information, see the help file for dppm.

Model fitting (spatial logistic regression)

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

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

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

Fitting a spatial logistic regression

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

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

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

Manipulating a fitted spatial logistic regression

anova.slrm	Analysis of deviance

coef.slrm	Extract fitted coefficients
vcov.slrm	Variance-covariance matrix of fitted coefficients
fitted.slrm	Compute fitted probabilities or intensity
logLik.slrm	Evaluate loglikelihood of fitted model
plot.slrm	Plot fitted probabilities or intensity
predict.slrm	Compute predicted probabilities or intensity with new data
simulate.slrm	Simulate model

There are many other undocumented methods for this class, including methods for print, update, formula and terms. Stepwise model selection is possible using step or stepAIC. For variable selection, see sdr.

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.

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

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

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

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

Sensitivity diagnostics:

Classical measures of model sensitivity such as leverage and influence have been adapted to point process models.

leverage.ppm	Leverage for point process model
influence.ppm	Influence for point process model
dfbetas.ppm	Parameter influence
dffit.ppm	Effect change diagnostic

Diagnostics for covariate effect:

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

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

Residual diagnostics:

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

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

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

Resampling and randomisation procedures

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

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

Licence

This library and its documentation are usable under the terms of the "GNU General Public License", a copy of which is distributed with the package.

Acknowledgements

Kasper Klitgaard Berthelsen, Ottmar Cronie, 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 <r.turner@auckland.ac.nz> 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"))
```

Arguments

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

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 `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` which estimates the intensity using a variable-bandwidth kernel 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 <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk> and Mehdi Moradi.

See Also

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

Examples

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

---

addvar *Added Variable Plot for Point Process Model*

Description

Computes the coordinates for an Added Variable Plot for a fitted point process model.

Usage

```
addvar(model, covariate, ...,
                    subregion=NULL,
                    bw="nrd0", adjust=1,
                    from=NULL, to=NULL, n=512,
                    bw.input = c("points", "quad"),
                    bw.restrict = FALSE,
                    covname, crosscheck=FALSE)
```

Arguments

model	Fitted point process model (object of class "ppm").
covariate	The covariate to be added to the model. Either a pixel image, a function(x,y), or a character string giving the name of a covariate that was supplied when the model was fitted.
subregion	Optional. A window (object of class "owin") specifying a subset of the spatial domain of the data. The calculation will be confined to the data in this subregion.
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.
...	Additional arguments passed to density.default.
bw.input	Character string specifying the input data used for automatic bandwidth selection.
bw.restrict	Logical value, specifying whether bandwidth selection is performed using data from the entire spatial domain or from the subregion.
covname	Optional. Character string to use as the name of the covariate.
crosscheck	For developers only. Logical value indicating whether to perform cross-checks on the validity of the calculation.

Details

This command generates the plot coordinates for an Added Variable Plot for a spatial point process model.

Added Variable Plots (Cox, 1958, sec 4.5; Wang, 1985) are commonly used in linear models and generalized linear models, to decide whether a model with response y and predictors x would be improved by including another predictor z.

In a (generalised) linear model with response y and predictors x, the Added Variable Plot for a new covariate z is a plot of the smoothed Pearson residuals from the original model against the scaled residuals from a weighted linear regression of z on x. If this plot has nonzero slope, then the new covariate z is needed. For general advice see Cook and Weisberg(1999); Harrell (2001).

Essentially the same technique can be used for a spatial point process model (Baddeley et al, 2012).

The argument `model` should be a fitted spatial point process model (object of class `"ppm"`).

The argument `covariate` identifies the covariate that is to be considered for addition to the model. It should be either a pixel image (object of class `"im"`) or a `function(x,y)` giving the values of the covariate at any spatial location. Alternatively `covariate` may be a character string, giving the name of a covariate that was supplied (in the `covariates` argument to `ppm`) when the model was fitted, but was not used in the model.

The result of `addvar(model, covariate)` is an object belonging to the classes `"addvar"` and `"fv"`. Plot this object to generate the added variable plot.

Note that the plot method shows the pointwise significance bands for a test of the *null* model, i.e. the null hypothesis that the new covariate has no effect.

The smoothing bandwidth is controlled by the arguments `bw`, `adjust`, `bw.input` and `bw.restrict`. If `bw` is a numeric value, then the bandwidth is taken to be `adjust * bw`. If `bw` is a string representing a bandwidth selection rule (recognised by `density.default`) then the bandwidth is selected by this rule.

The data used for automatic bandwidth selection are specified by `bw.input` and `bw.restrict`. If `bw.input="points"` (the default) then bandwidth selection is based on the covariate values at the points of the original point pattern dataset to which the model was fitted. If `bw.input="quad"` then bandwidth selection is based on the covariate values at every quadrature point used to fit the model. If `bw.restrict=TRUE` then the bandwidth selection is performed using only data from inside the `subregion`.

Value

An object of class `"addvar"` containing the coordinates for the added variable plot. There is a `plot` method.

Slow computation

In a large dataset, computation can be very slow if the default settings are used, because the smoothing bandwidth is selected automatically. To avoid this, specify a numerical value for the bandwidth `bw`. One strategy is to use a coarser subset of the data to select `bw` automatically. The selected bandwidth can be read off the print output for `addvar`.

Internal data

The return value has an attribute "spatial" which contains the internal data: the computed values of the residuals, and of all relevant covariates, at each quadrature point of the model. It is an object of class "ppp" with a data frame of marks.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz>, Ya-Mei Chang and Yong Song.

References

Baddeley, A., Chang, Y.-M., Song, Y. and Turner, R. (2013) Residual diagnostics for covariate effects in spatial point process models. *Journal of Computational and Graphical Statistics*, **22**, 886–905.

Cook, R.D. and Weisberg, S. (1999) *Applied regression, including computing and graphics*. New York: Wiley.

Cox, D.R. (1958) *Planning of Experiments*. New York: Wiley.

Harrell, F. (2001) *Regression Modeling Strategies*. New York: Springer.

Wang, P. (1985) Adding a variable in generalized linear models. *Technometrics* **27**, 273–276.

See Also

parres, rhohat, rho2hat.

Examples

```
X <-  rpoispp(function(x,y){exp(3+3*x)})
model <- ppm(X, ~y)
adv <- addvar(model, "x")
plot(adv)
adv <- addvar(model, "x", subregion=square(0.5))
```

---

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.solist).

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 <r.turner@auckland.ac.nz>

See Also

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

Examples

```
data(swedishpines)
        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 <r.turner@auckland.ac.nz>.

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))

 # A setting where you might REALLY want to use dataname:
 # xxx <- alltypes(ppp(Melvin$x,Melvin$y,
 #              window=as.owin(c(5,20,15,50)),marks=clyde),
 #              fun="F",verb=TRUE,dataname="Melvin")

 # 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])
```

---

anova.mppm *ANOVA for Fitted Point Process Models for Replicated Patterns*

Description

Performs analysis of deviance for one or more point process models fitted to replicated point pattern data.

Usage

```
  ## S3 method for class 'mppm'
anova(object, ...,
                  test=NULL, adjust=TRUE,
                  fine=FALSE, warn=TRUE)
```

Arguments

object	Object of class "mppm" representing a point process model that was fitted to replicated point patterns.
...	Optional. Additional objects of class "mppm".
test	Type of hypothesis test to perform. A character string, partially matching one of "Chisq", "LRT", "Rao", "score", "F" or "Cp", or NULL indicating that no test should be performed.
adjust	Logical value indicating whether to correct the pseudolikelihood ratio when some of the models are not Poisson processes.
fine	Logical value passed to vcov.ppm indicating whether to use a quick estimate (fine=FALSE, the default) or a slower, more accurate estimate (fine=TRUE) of the variance of the fitted coefficients of each model. Relevant only when some of the models are not Poisson and adjust=TRUE.
warn	Logical value indicating whether to issue warnings if problems arise.

Details

This is a method for anova for comparing several fitted point process models of class "mppm", usually generated by the model-fitting function mppm).

If the fitted models are all Poisson point processes, then this function performs an Analysis of Deviance of the fitted models. The output shows the deviance differences (i.e. 2 times log likelihood ratio), the difference in degrees of freedom, and (if test="Chi") the two-sided p-values for the chi-squared tests. Their interpretation is very similar to that in anova.glm.

If some of the fitted models are *not* Poisson point processes, the 'deviance' differences in this table are 'pseudo-deviances' equal to 2 times the differences in the maximised values of the log pseudolikelihood (see ppm). It is not valid to compare these values to the chi-squared distribution. In this case, if adjust=TRUE (the default), the pseudo-deviances will be adjusted using the method of Pace et al (2011) and Baddeley, Turner and Rubak (2015) so that the chi-squared test is valid. It is strongly advisable to perform this adjustment.

The argument test determines which hypothesis test, if any, will be performed to compare the models. The argument test should be a character string, partially matching one of "Chisq", "F" or "Cp", or NULL. The first option "Chisq" gives the likelihood ratio test based on the asymptotic chi-squared distribution of the deviance difference. The meaning of the other options is explained in anova.glm.

Value

An object of class "anova", or NULL.

Random effects models are currently not supported

For models with random effects (i.e. where the call to mppm included the argument random),

analysis of deviance is currently not supported, due to changes in the **nlme** package. We will try to find a solution.

Error messages

An error message that reports *system is computationally singular* indicates that the determinant of the Fisher information matrix of one of the models was either too large or too small for reliable numerical calculation. See vcov.ppm for suggestions on how to handle this.

Author(s)

Adrian Baddeley, Ida-Maria Sintorn and Leanne Bischoff. Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.ed
Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

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

Baddeley, A., Turner, R. and Rubak, E. (2015) Adjusted composite likelihood ratio test for Gibbs point processes. *Journal of Statistical Computation and Simulation* **86** (5) 922–941. DOI: 10.1080/00949655.2015.1044530.

Pace, L., Salvan, A. and Sartori, N. (2011) Adjusting composite likelihood ratio statistics. *Statistica Sinica* **21**, 129–148.

See Also

mppm

Examples

```
H <- hyperframe(X=waterstriders)
#' test for loglinear trend in x coordinate
mod0 <- mppm(X~1, data=H, Poisson())
modx <- mppm(X~x, data=H, Poisson())
anova(mod0, modx, test="Chi")
# not significant
anova(modx, test="Chi")
# not significant

#' test for inhibition
mod0S <- mppm(X~1, data=H, Strauss(2))
anova(mod0, mod0S, test="Chi")
# significant!

#' test for trend after accounting for inhibition
modxS <- mppm(X~x, data=H, Strauss(2))
anova(mod0S, modxS, test="Chi")
# not significant
```

---

anova.ppm *ANOVA for Fitted Point Process Models*

Description

Performs analysis of deviance for one or more fitted point process models.

Usage

```
  ## S3 method for class 'ppm'
anova(object, ..., test=NULL,
                     adjust=TRUE, warn=TRUE, fine=FALSE)
```

Arguments

object	A fitted point process model (object of class "ppm").
...	Optional. Additional objects of class "ppm".
test	Character string, partially matching one of "Chisq", "LRT", "Rao", "score", "F" or "Cp", or NULL indicating that no test should be performed.
adjust	Logical value indicating whether to correct the pseudolikelihood ratio when some of the models are not Poisson processes.
warn	Logical value indicating whether to issue warnings if problems arise.
fine	Logical value, passed to vcov.ppm, indicating whether to use a quick estimate (fine=FALSE, the default) or a slower, more accurate estimate (fine=TRUE) of variance terms. Relevant only when some of the models are not Poisson and adjust=TRUE.

Details

This is a method for anova for fitted point process models (objects of class "ppm", usually generated by the model-fitting function ppm).

If the fitted models are all Poisson point processes, then by default, this function performs an Analysis of Deviance of the fitted models. The output shows the deviance differences (i.e. 2 times log likelihood ratio), the difference in degrees of freedom, and (if test="Chi" or test="LRT") the two-sided p-values for the chi-squared tests. Their interpretation is very similar to that in anova.glm. If test="Rao" or test="score", the *score test* (Rao, 1948) is performed instead.

If some of the fitted models are *not* Poisson point processes, the 'deviance' differences in this table are 'pseudo-deviances' equal to 2 times the differences in the maximised values of the log pseudolikelihood (see ppm). It is not valid to compare these values to the chi-squared distribution. In this case, if adjust=TRUE (the default), the pseudo-deviances will be adjusted using the method of Pace et al (2011) and Baddeley et al (2015) so that the chi-squared test is valid. It is strongly advisable to perform this adjustment.

Value

An object of class "anova", or NULL.

Errors and warnings

models not nested: There may be an error message that the models are not "nested". For an Analysis of Deviance the models must be nested, i.e. one model must be a special case of the other. For example the point process model with formula ~x is a special case of the model with formula ~x+y, so these models are nested. However the two point process models with formulae ~x and ~y are not nested.

If you get this error message and you believe that the models should be nested, the problem may be the inability of R to recognise that the two formulae are nested. Try modifying the formulae to make their relationship more obvious.

different sizes of dataset: There may be an error message from anova.glmlist that "models were not all fitted to the same size of dataset". This implies that the models were fitted using different quadrature schemes (see quadscheme) and/or with different edge corrections or different values of the border edge correction distance rbord.

To ensure that models are comparable, check the following:

- the models must all have been fitted to the same point pattern dataset, in the same window.
- all models must have been fitted by the same fitting method as specified by the argument method in ppm.
- If some of the models depend on covariates, then they should all have been fitted using the same list of covariates, and using allcovar=TRUE to ensure that the same quadrature scheme is used.
- all models must have been fitted using the same edge correction as specified by the arguments correction and rbord. If you did not specify the value of rbord, then it may have taken a different value for different models. The default value of rbord is equal to zero for a Poisson model, and otherwise equals the reach (interaction distance) of the interaction term (see reach). To ensure that the models are comparable, set rbord to equal the maximum reach of the interactions that you are fitting.

Error messages

An error message that reports *system is computationally singular* indicates that the determinant of the Fisher information matrix of one of the models was either too large or too small for reliable numerical calculation. See vcov.ppm for suggestions on how to handle this.

Author(s)

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

References

Baddeley, A., Turner, R. and Rubak, E. (2015) Adjusted composite likelihood ratio test for Gibbs point processes. *Journal of Statistical Computation and Simulation* **86** (5) 922–941. DOI: 10.1080/00949655.2015.1044530.

Pace, L., Salvan, A. and Sartori, N. (2011) Adjusting composite likelihood ratio statistics. *Statistica Sinica* **21**, 129–148.

Rao, C.R. (1948) Large sample tests of statistical hypotheses concerning several parameters with applications to problems of estimation. *Proceedings of the Cambridge Philosophical Society* **44**, 50–57.

See Also

ppm, vcov.ppm

Examples

```
mod0 <- ppm(swedishpines ~1)
modx <- ppm(swedishpines ~x)
# Likelihood ratio test
anova(mod0, modx, test="Chi")
# Score test
anova(mod0, modx, test="Rao")

# Single argument
modxy <- ppm(swedishpines ~x + y)
anova(modxy, test="Chi")

# Adjusted composite likelihood ratio test
modP <- ppm(swedishpines ~1, rbord=9)
modS <- ppm(swedishpines ~1, Strauss(9))
anova(modP, modS, test="Chi")
```

anova.slrm *Analysis of Deviance for Spatial Logistic Regression Models*

Description

Performs Analysis of Deviance for two or more fitted Spatial Logistic Regression models.

Usage

```
  ## S3 method for class 'slrm'
anova(object, ..., test = NULL)
```

Arguments

object	a fitted spatial logistic regression model. An object of class "slrm".
...	additional objects of the same type (optional).
test	a character string, (partially) matching one of "Chisq", "F" or "Cp", indicating the reference distribution that should be used to compute p-values.

Details

This is a method for anova for fitted spatial logistic regression models (objects of class "slrm", usually obtained from the function slrm).

The output shows the deviance differences (i.e. 2 times log likelihood ratio), the difference in degrees of freedom, and (if test="Chi") the two-sided p-values for the chi-squared tests. Their interpretation is very similar to that in anova.glm.

Value

An object of class "anova", inheriting from class "data.frame", representing the analysis of deviance table.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> <adrian@maths.uwa.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

slrm

Examples

```
X <- rpoispp(42)
fit0 <- slrm(X ~ 1)
fit1 <- slrm(X ~ x+y)
anova(fit0, fit1, test="Chi")
```

`AreaInter` *The Area Interaction Point Process Model*

Description

Creates an instance of the Area Interaction point process model (Widom-Rowlinson penetrable spheres model) which can then be fitted to point pattern data.

Usage

```
AreaInter(r)
```

Arguments

`r`	The radius of the discs in the area interaction process

Details

This function defines the interpoint interaction structure of a point process called the Widom-Rowlinson penetrable sphere model or area-interaction process. It can be used to fit this model to point pattern data.

The function `ppm()`, which fits point process models to point pattern data, requires an argument of class `"interact"` describing the interpoint interaction structure of the model to be fitted. The appropriate description of the area interaction structure is yielded by the function `AreaInter()`. See the examples below.

In **standard form**, the area-interaction process (Widom and Rowlinson, 1970; Baddeley and Van Lieshout, 1995) with disc radius r, intensity parameter κ and interaction parameter γ is a point process with probability density

$$f(x_1, \ldots, x_n) = \alpha \kappa^{n(x)} \gamma^{-A(x)}$$

for a point pattern x, where $x_1, \ldots, x_n$ represent the points of the pattern, $n(x)$ is the number of points in the pattern, and $A(x)$ is the area of the region formed by the union of discs of radius r centred at the points $x_1, \ldots, x_n$. Here α is a normalising constant.

The interaction parameter γ can be any positive number. If $\gamma = 1$ then the model reduces to a Poisson process with intensity κ. If $\gamma < 1$ then the process is regular, while if $\gamma > 1$ the process is clustered. Thus, an area interaction process can be used to model either clustered or regular point patterns. Two points interact if the distance between them is less than $2r$.

The standard form of the model, shown above, is a little complicated to interpret in practical applications. For example, each isolated point of the pattern x contributes a factor $\kappa\gamma^{-\pi r^2}$ to the probability density.

In **spatstat**, the model is parametrised in a different form, which is easier to interpret. In **canonical scale-free form**, the probability density is rewritten as

$$f(x_1, \ldots, x_n) = \alpha \beta^{n(x)} \eta^{-C(x)}$$

where β is the new intensity parameter, η is the new interaction parameter, and $C(x) = B(x) - n(x)$ is the interaction potential. Here

$$B(x) = \frac{A(x)}{\pi r^2}$$

is the normalised area (so that the discs have unit area). In this formulation, each isolated point of the pattern contributes a factor β to the probability density (so the first order trend is β). The quantity $C(x)$ is a true interaction potential, in the sense that $C(x) = 0$ if the point pattern x does not contain any points that lie close together (closer than $2r$ units apart).

When a new point u is added to an existing point pattern x, the rescaled potential $-C(x)$ increases by a value between 0 and 1. The increase is zero if u is not close to any point of x. The increase is 1 if the disc of radius r centred at u is completely contained in the union of discs of radius r centred at the data points x_i. Thus, the increase in potential is a measure of how close the new point u is to the existing pattern x. Addition of the point u contributes a factor $\beta\eta^\delta$ to the probability density, where δ is the increase in potential.

The old parameters κ, γ of the standard form are related to the new parameters β, η of the canonical scale-free form, by

$$\beta = \kappa\gamma^{-\pi r^2} = \kappa/\eta$$

and

$$\eta = \gamma^{\pi r^2}$$

provided γ and κ are positive and finite.

In the canonical scale-free form, the parameter η can take any nonnegative value. The value $\eta = 1$ again corresponds to a Poisson process, with intensity β. If $\eta < 1$ then the process is regular, while if $\eta > 1$ the process is clustered. The value $\eta = 0$ corresponds to a hard core process with hard core radius r (interaction distance $2r$).

The *nonstationary* area interaction process is similar except that the contribution of each individual point x_i is a function $\beta(x_i)$ of location, rather than a constant beta.

Note the only argument of `AreaInter()` is the disc radius `r`. When `r` is fixed, the model becomes an exponential family. The canonical parameters $\log(\beta)$ and $\log(\eta)$ are estimated by `ppm()`, not fixed in `AreaInter()`.

Value

An object of class `"interact"` describing the interpoint interaction structure of the area-interaction process with disc radius r.

Warnings

The interaction distance of this process is equal to `2 * r`. Two discs of radius `r` overlap if their centres are closer than `2 * r` units apart.

The estimate of the interaction parameter η is unreliable if the interaction radius `r` is too small or too large. In these situations the model is approximately Poisson so that η is unidentifiable. As a rule of thumb, one can inspect the empty space function of the data, computed by `Fest`. The value $F(r)$ of the empty space function at the interaction radius `r` should be between 0.2 and 0.8.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A.J. and Van Lieshout, M.N.M. (1995). Area-interaction point processes. *Annals of the Institute of Statistical Mathematics* **47** (1995) 601–619.

Widom, B. and Rowlinson, J.S. (1970). New model for the study of liquid-vapor phase transitions. *The Journal of Chemical Physics* **52** (1970) 1670–1684.

See Also

ppm, pairwise.family, ppm.object

ragsAreaInter and rmh for simulation of area-interaction models.

Examples

```
# prints a sensible description of itself
AreaInter(r=0.1)

# Note the reach is twice the radius
reach(AreaInter(r=1))

# Fit the stationary area interaction process to Swedish Pines data
data(swedishpines)
ppm(swedishpines ~1, AreaInter(r=7))

# Fit the stationary area interaction process to `cells'
ppm(cells ~1, AreaInter(r=0.06))
# eta=0 indicates hard core process.

# Fit a nonstationary area interaction with log-cubic polynomial trend
#   ppm(swedishpines ~polynom(x/10,y/10,3), AreaInter(r=7))
```

---

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 <r.turner@auckland.ac.nz> 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 <r.turner@auckland.ac.nz>

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.leverage.ppm

Convert Leverage Object to Function of Coordinates

Description

Converts an object of class "leverage.ppm" to a function of the x and y coordinates.

Usage

```
## S3 method for class 'leverage.ppm'
as.function(x, ...)
```

Arguments

x	Object of class "leverage.ppm" produced by leverage.ppm.
...	Ignored.

Details

An object of class "leverage.ppm" represents the leverage function of a fitted point process model. This command converts the object to a function(x,y) where the arguments x and y are (vectors of) spatial coordinates. This function returns the leverage values at the specified locations (calculated by referring to the nearest location where the leverage has been computed).

Value

A function in the R language, also belonging to the class "funxy".

Author(s)

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

See Also

as.im.leverage.ppm

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X ~x+y)
lev <- leverage(fit)
f <- as.function(lev)

f(0.2, 0.3)  # evaluate at (x,y) coordinates
y <- f(X)    # evaluate at a point pattern
```

---

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 <r.turner@auckland.ac.nz>

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 'minconfit'
as.fv(x)

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

## S3 method for class 'kppm'
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 <r.turner@auckland.ac.nz> 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.interact *Extract Interaction Structure*

Description

Extracts the interpoint interaction structure from a point pattern model.

Usage

```
as.interact(object)
## S3 method for class 'fii'
as.interact(object)
## S3 method for class 'interact'
as.interact(object)
## S3 method for class 'ppm'
as.interact(object)
```

Arguments

object	A fitted point process model (object of class "ppm") or an interpoint interaction structure (object of class "interact").

Details

The function as.interact extracts the interpoint interaction structure from a suitable object.

An object of class "interact" describes an interpoint interaction structure, before it has been fitted to point pattern data. The irregular parameters of the interaction (such as the interaction range) are fixed, but the regular parameters (such as interaction strength) are undetermined. Objects of this class are created by the functions Poisson, Strauss and so on. The main use of such objects is in a call to ppm.

The function as.interact is generic, with methods for the classes "ppm", "fii" and "interact". The result is an object of class "interact" which can be printed.

Value

An object of class "interact" representing the interpoint interaction. This object can be printed and plotted.

Note on parameters

This function does **not** extract the fitted coefficients of the interaction. To extract the fitted interaction including the fitted coefficients, use fitin.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

fitin, ppm.

Examples

```
data(cells)
model <- ppm(cells, ~1, Strauss(0.07))
f <- as.interact(model)
f
```

`as.layered.msr` *Convert Measure To Layered Object*

Description

Converts a measure into a layered object.

Usage

```
 ## S3 method for class 'msr'
as.layered(X)
```

Arguments

`X`	A measure (object of class `"msr"`).

Details

This function converts the object `X` into an object of class `"layered"`.

It is a method for the generic `as.layered` for the class of measures.

If `X` is a vector-valued measure, then `as.layered(X)` consists of several layers, each containing a scalar-valued measure.

Value

An object of class `"layered"` (see `layered`).

Author(s)

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

See Also

`as.layered`, `msr`.

Examples

```
P <- rpoispp(100)
fit <- ppm(P ~ x+y)
rs <- residuals(fit, type="score")
as.layered(rs)
```

`as.owin.ppm` *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 'ppm'
as.owin(W, ..., from=c("points", "covariates"), fatal=TRUE)

 ## S3 method for class 'kppm'
as.owin(W, ..., from=c("points", "covariates"), fatal=TRUE)

 ## S3 method for class 'dppm'
as.owin(W, ..., from=c("points", "covariates"), fatal=TRUE)

 ## S3 method for class 'slrm'
as.owin(W, ..., from=c("points", "covariates"))

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

 ## 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.
`from`	Character string. See Details.

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 class "SpatialPolygon", "SpatialPolygons" or "SpatialPolygonsDataFrame". To handle these data types, **the package maptools must be loaded**, because it provides the methods for as.owin for these classes. 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 <r.turner@auckland.ac.nz> 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 are provided in the **maptools** package: as.owin.SpatialPolygon, as.owin.SpatialPolygons, as.owin.SpatialPolygonsDataFrame.

Examples

```
fit <- ppm(cells ~ 1)
as.owin(fit)
```

---

as.ppm *Extract Fitted Point Process Model*

Description

Extracts the fitted point process model from some kind of fitted model.

Usage

```
as.ppm(object)

## S3 method for class 'ppm'
as.ppm(object)

## S3 method for class 'profilepl'
as.ppm(object)
```

```
## S3 method for class 'kppm'
as.ppm(object)

## S3 method for class 'dppm'
as.ppm(object)
```

Arguments

object	An object that includes a fitted Poisson or Gibbs point process model. An object of class "ppm", "profilepl", "kppm" or "dppm" or possibly other classes.

Details

The function as.ppm extracts the fitted point process model (of class "ppm") from a suitable object.

The function as.ppm is generic, with methods for the classes "ppm", "profilepl", "kppm" and "dppm", and possibly for other classes.

For the class "profilepl" of models fitted by maximum profile pseudolikelihood, the method as.ppm.profilepl extracts the fitted point process model (with the optimal values of the irregular parameters).

For the class "kppm" of models fitted by minimum contrast (or Palm or composite likelihood) using Waagepetersen's two-step estimation procedure (see kppm), the method as.ppm.kppm extracts the Poisson point process model that is fitted in the first stage of the procedure.

The behaviour for the class "dppm" is analogous to the "kppm" case above.

Value

An object of class "ppm".

Author(s)

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

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

ppm, profilepl.

Examples

```
# fit a model by profile maximum pseudolikelihood
rvals <- data.frame(r=(1:10)/100)
pfit <- profilepl(rvals, Strauss, cells, ~1)
# extract the fitted model
fit <- as.ppm(pfit)
```

auc *Area Under ROC Curve*

Description

Compute the AUC (area under the Receiver Operating Characteristic curve) for a fitted point process model.

Usage

```
auc(X, ...)

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

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

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

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

Arguments

`X`	Point pattern (object of class `"ppp"` or `"lpp"`) or fitted point process model (object of class `"ppm"`, `"kppm"`, `"slrm"` or `"lppm"`).
`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 `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.

Details

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

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.

For a fitted point process model X, the AUC measures the ability of the fitted model intensity to separate the spatial domain into areas of high and low density of points. Suppose $\lambda(u)$ is the intensity function of the model. The AUC is the probability that $\lambda(x_i) > \lambda(U)$. That is, AUC is the probability that a randomly-selected data point has higher predicted intensity than does a randomly-selected spatial location. The AUC is **not** a measure of the goodness-of-fit of the model (Lobo et al, 2007).

(For spatial logistic regression models (class "slrm") replace "intensity" by "probability of presence" in the text above.)

Value

Numeric. For auc.ppp and auc.lpp, the result is a single number giving the AUC value. For auc.ppm, auc.kppm and auc.lppm, the result is a numeric vector of length 2 giving the AUC value and the theoretically expected AUC value for this model.

Author(s)

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

References

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

Examples

```
fit <- ppm(swedishpines ~ x+y)
auc(fit)
auc(swedishpines, "x")
```

BadGey *Hybrid Geyer Point Process Model*

Description

Creates an instance of the Baddeley-Geyer point process model, defined as a hybrid of several Geyer interactions. The model can then be fitted to point pattern data.

Usage

```
BadGey(r, sat)
```

Arguments

r	vector of interaction radii
sat	vector of saturation parameters, or a single common value of saturation parameter

Details

This is Baddeley's generalisation of the Geyer saturation point process model, described in Geyer, to a process with multiple interaction distances.

The BadGey point process with interaction radii $r_1, \ldots, r_k$, saturation thresholds $s_1, \ldots, s_k$, intensity parameter β and interaction parameters $\gamma_1, \ldots, gamma_k$, is the point process in which each point x_i in the pattern X contributes a factor

$$\beta \gamma_1^{v_1(x_i,X)} \ldots gamma_k^{v_k(x_i,X)}$$

to the probability density of the point pattern, where

$$v_j(x_i, X) = \min(s_j, t_j(x_i, X))$$

where $t_j(x_i, X)$ denotes the number of points in the pattern X which lie within a distance r_j from the point x_i.

BadGey is used to fit this model to data. The function ppm(), which fits point process models to point pattern data, requires an argument of class "interact" describing the interpoint interaction structure of the model to be fitted. The appropriate description of the piecewise constant Saturated pairwise interaction is yielded by the function BadGey(). See the examples below.

The argument r specifies the vector of interaction distances. The entries of r must be strictly increasing, positive numbers.

The argument sat specifies the vector of saturation parameters that are applied to the point counts $t_j(x_i, X)$. It should be a vector of the same length as r, and its entries should be nonnegative numbers. Thus sat[1] is applied to the count of points within a distance r[1], and sat[2] to the count of points within a distance r[2], etc. Alternatively sat may be a single number, and this saturation value will be applied to every count.

Infinite values of the saturation parameters are also permitted; in this case $v_j(x_i, X) = t_j(x_i, X)$ and there is effectively no 'saturation' for the distance range in question. If all the saturation parameters are set to Inf then the model is effectively a pairwise interaction process, equivalent to PairPiece (however the interaction parameters γ obtained from BadGey have a complicated relationship to the interaction parameters γ obtained from PairPiece).

If r is a single number, this model is virtually equivalent to the Geyer process, see Geyer.

Value

An object of class "interact" describing the interpoint interaction structure of a point process.

Hybrids

A 'hybrid' interaction is one which is built by combining several different interactions (Baddeley et al, 2013). The BadGey interaction can be described as a hybrid of several Geyer interactions.

The Hybrid command can be used to build hybrids of any interactions. If the Hybrid operator is applied to several Geyer models, the result is equivalent to a BadGey model. This can be useful for incremental model selection.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz> in collaboration with Hao Wang and Jeff Picka

References

Baddeley, A., Turner, R., Mateu, J. and Bevan, A. (2013) Hybrids of Gibbs point process models and their implementation. *Journal of Statistical Software* **55**:11, 1–43. DOI: 10.18637/jss.v055.i11

See Also

ppm, pairsat.family, Geyer, PairPiece, SatPiece, Hybrid

Examples

```
BadGey(c(0.1,0.2), c(1,1))
# prints a sensible description of itself
BadGey(c(0.1,0.2), 1)

# fit a stationary Baddeley-Geyer model
ppm(cells ~1, BadGey(c(0.07, 0.1, 0.13), 2))

# nonstationary process with log-cubic polynomial trend
# ppm(cells ~polynom(x,y,3), BadGey(c(0.07, 0.1, 0.13), 2))
```

---

bc.ppm *Bias Correction for Fitted Model*

Description

Applies a first-order bias correction to a fitted model.

Usage

```
bc(fit, ...)

## S3 method for class 'ppm'
bc(fit, ..., nfine = 256)
```

Arguments

fit	A fitted point process model (object of class "ppm") or a model of some other class.
...	Additional arguments are currently ignored.
nfine	Grid dimensions for fine grid of locations. An integer, or a pair of integers. See Details.

Details

This command applies the first order Newton-Raphson bias correction method of Baddeley and Turner (2014, sec 4.2) to a fitted model. The function bc is generic, with a method for fitted point process models of class "ppm".

A fine grid of locations, of dimensions nfine * nfine or nfine[2] * nfine[1], is created over the original window of the data, and the intensity or conditional intensity of the fitted model is calculated on this grid. The result is used to update the fitted model parameters once by a Newton-Raphson update.

This is only useful if the quadrature points used to fit the original model fit are coarser than the grid of points specified by nfine.

Value

A numeric vector, of the same length as coef(fit), giving updated values for the fitted model coefficients.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>.

References

Baddeley, A. and Turner, R. (2014) Bias correction for parameter estimates of spatial point process models. *Journal of Statistical Computation and Simulation* **84**, 1621–1643. DOI: 10.1080/00949655.2012.755976

See Also

rex

Examples

```
fit <- ppm(cells ~ x, Strauss(0.07))
coef(fit)
if(!interactive()) {
  bc(fit, nfine=64)
} else {
  bc(fit)
}
```

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"), ...)

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

Arguments

X	A point pattern (object of class "ppp" or "lpp").
model	A fitted point process model (object of class "ppm" or "lppm").
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 and 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 <r.turner@auckland.ac.nz> 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
data(copper)
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"`.
`x`	An object of class `"fv"`.
`y`	Either a data frame or an object of class `"fv"`.
`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 two 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.

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 must be provided.

Value

An object of class "fv".

Author(s)

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

See Also

fv, with.fv.

Undocumented functions for modifying an "fv" object include fvnames, fvnames<-, 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")
```

`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 <r.turner@auckland.ac.nz> 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*, in press.

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 <r.turner@auckland.ac.nz>

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)
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(mfrow=c(1,1))
```

bw.abram *Abramson's Adaptive Bandwidths*

Description

Computes adaptive smoothing bandwidths according to the inverse-square-root rule of Abramson (1982).

Usage

```
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).

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 M. Davies. 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.

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.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].

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 bandwidth selection criterion as a function of sigma.

Value

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

Author(s)

Ottmar Cronie <ottmar.cronie@umu.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 <r.turner@auckland.ac.nz> 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.diggle, bw.scott, bw.ppl, bw.frac.

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

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 <r.turner@auckland.ac.nz> 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.ppl, bw.scott, bw.CvL, bw.frac.

Examples

```
data(lansing)
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 <r.turner@auckland.ac.nz> 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.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, lambda=NULL, divisor="r",
       kernel="epanechnikov", nr=10000, bias.correct=TRUE,
       cv.method=c("compLik", "leastSQ"), simple=TRUE, srange=NULL,
..., verbose=FALSE, warn=TRUE)
```

Arguments

X	A point pattern (object of class "ppp").
rmax	Numeric. Maximum value of the spatial lag distance r for which $g(r)$ should be evaluated.
lambda	Optional. Values of the estimated intensity function. A vector giving the intensity values at the points of the pattern X.
divisor	Choice of divisor in the estimation formula: either "r" (the default) or "d". See pcf.ppp.
kernel	Choice of smoothing kernel, passed to density; see pcf and pcfinhom.
nr	Integer. Number of subintervals for discretization of [0, rmax] to use in computing numerical integrals.
bias.correct	Logical. Whether to use bias corrected version of the kernel estimate. See Details.

cv.method	Choice of cross validation method: either "compLik" or "leastSQ" (partially matched).
simple	Logical. Whether to use simple removal of spatial lag distances. See Details.
srange	Optional. Numeric vector of length 2 giving the range of bandwidth values that should be searched to find the optimum bandwidth.
...	Other arguments, passed to pcf or pcfinhom.
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.ppp (homogeneous case) or pcfinhom (inhomogeneous case).

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).

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

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 <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

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.ppp, pcfinhom

Examples

```
b <- bw.pcf(redwood)
plot(pcf(redwood, 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, weights=NULL,
       shortcut=FALSE, 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.
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 $\mathrm{LCV}(\sigma)$ 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 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.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> 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, 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.

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

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>.

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

Examples

```
data(urkiola)

b <- bw.relrisk(urkiola)
b
plot(b)
```

```
b <- bw.relrisk(urkiola, hmax=20)
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 <r.turner@auckland.ac.nz> 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")
```

Arguments

X	A marked point pattern with numeric marks.
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").

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 $\sum_i (y_i - \hat{y}_i)^2$.

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

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

Smooth.ppp

Examples

```
data(longleaf)

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

---

bw.stoyan *Stoyan's Rule of Thumb for Bandwidth Selection*

Description

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

Usage

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

Arguments

X	A point pattern (object of class "ppp").
co	Coefficient appearing in the rule of thumb. See Details.

Details

Estimation of the pair correlation function and other 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.

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>

and Rolf Turner <r.turner@auckland.ac.nz>

References

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

```
data(shapley)
bw.stoyan(shapley)
```

---

cauchy.estK *Fit the Neyman-Scott cluster process with Cauchy kernel*

Description

Fits the Neyman-Scott Cluster point process with Cauchy kernel to a point pattern dataset by the Method of Minimum Contrast.

Usage

```
cauchy.estK(X, startpar=c(kappa=1,scale=1), lambda=NULL,
            q = 1/4, p = 2, rmin = NULL, rmax = NULL, ...)
```

Arguments

X	Data to which the model will be fitted. Either a point pattern or a summary statistic. See Details.
startpar	Vector of starting values for the parameters of the model.
lambda	Optional. An estimate of the intensity of the point process.
q,p	Optional. Exponents for the contrast criterion.
rmin, rmax	Optional. The interval of r values for the contrast criterion.
...	Optional arguments passed to optim to control the optimisation algorithm. See Details.

Details

This algorithm fits the Neyman-Scott cluster point process model with Cauchy kernel to a point pattern dataset by the Method of Minimum Contrast, using the K function.

The argument X can be either

a point pattern: An object of class "ppp" representing a point pattern dataset. The K function of the point pattern will be computed using Kest, and the method of minimum contrast will be applied to this.

a summary statistic: An object of class "fv" containing the values of a summary statistic, computed for a point pattern dataset. The summary statistic should be the K function, and this object should have been obtained by a call to Kest or one of its relatives.

The algorithm fits the Neyman-Scott cluster point process with Cauchy kernel to X, by finding the parameters of the Matérn Cluster model which give the closest match between the theoretical K function of the Matérn Cluster process and the observed K function. For a more detailed explanation of the Method of Minimum Contrast, see mincontrast.

The model is described in Jalilian et al (2013). It is a cluster process formed by taking a pattern of parent points, generated according to a Poisson process with intensity κ, and around each parent point, generating a random number of offspring points, such that the number of offspring of each parent is a Poisson random variable with mean μ, and the locations of the offspring points of one parent follow a common distribution described in Jalilian et al (2013).

If the argument lambda is provided, then this is used as the value of the point process intensity λ. Otherwise, if X is a point pattern, then λ will be estimated from X. If X is a summary statistic and lambda is missing, then the intensity λ cannot be estimated, and the parameter μ will be returned as NA.

The remaining arguments rmin,rmax,q,p control the method of minimum contrast; see mincontrast.

The corresponding model can be simulated using rCauchy.

For computational reasons, the optimisation procedure uses the parameter eta2, which is equivalent to 4 * scale^2 where scale is the scale parameter for the model as used in rCauchy.

Homogeneous or inhomogeneous Neyman-Scott/Cauchy models can also be fitted using the function kppm and the fitted models can be simulated using simulate.kppm.

The optimisation algorithm can be controlled through the additional arguments "..." which are passed to the optimisation function optim. For example, to constrain the parameter values to a certain range, use the argument method="L-BFGS-B" to select an optimisation algorithm that respects box constraints, and use the arguments lower and upper to specify (vectors of) minimum and maximum values for each parameter.

Value

An object of class "minconfit". There are methods for printing and plotting this object. It contains the following main components:

par	Vector of fitted parameter values.
fit	Function value table (object of class "fv") containing the observed values of the summary statistic (observed) and the theoretical values of the summary statistic computed from the fitted model parameters.

Author(s)

Abdollah Jalilian and Rasmus Waagepetersen. Adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu.a

References

Ghorbani, M. (2012) Cauchy cluster process. *Metrika*, to appear.

Jalilian, A., Guan, Y. and Waagepetersen, R. (2013) Decomposition of variance for spatial Cox processes. *Scandinavian Journal of Statistics* **40**, 119-137.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

kppm, cauchy.estpcf, lgcp.estK, thomas.estK, vargamma.estK, mincontrast, Kest, Kmodel.

rCauchy to simulate the model.

Examples

```
u <- cauchy.estK(redwood)
u
plot(u)
```

---

cauchy.estpcf *Fit the Neyman-Scott cluster process with Cauchy kernel*

Description

Fits the Neyman-Scott Cluster point process with Cauchy kernel to a point pattern dataset by the Method of Minimum Contrast, using the pair correlation function.

Usage

```
cauchy.estpcf(X, startpar=c(kappa=1,scale=1), lambda=NULL,
            q = 1/4, p = 2, rmin = NULL, rmax = NULL, ...,
            pcfargs = list())
```

Arguments

X	Data to which the model will be fitted. Either a point pattern or a summary statistic. See Details.
startpar	Vector of starting values for the parameters of the model.
lambda	Optional. An estimate of the intensity of the point process.
q,p	Optional. Exponents for the contrast criterion.
rmin, rmax	Optional. The interval of r values for the contrast criterion.
...	Optional arguments passed to optim to control the optimisation algorithm. See Details.
pcfargs	Optional list containing arguments passed to pcf.ppp to control the smoothing in the estimation of the pair correlation function.

Details

This algorithm fits the Neyman-Scott cluster point process model with Cauchy kernel to a point pattern dataset by the Method of Minimum Contrast, using the pair correlation function.

The argument X can be either

a point pattern: An object of class "ppp" representing a point pattern dataset. The pair correlation function of the point pattern will be computed using pcf, and the method of minimum contrast will be applied to this.

a summary statistic: An object of class "fv" containing the values of a summary statistic, computed for a point pattern dataset. The summary statistic should be the pair correlation function, and this object should have been obtained by a call to pcf or one of its relatives.

The algorithm fits the Neyman-Scott cluster point process with Cauchy kernel to X, by finding the parameters of the Matérn Cluster model which give the closest match between the theoretical pair correlation function of the Matérn Cluster process and the observed pair correlation function. For a more detailed explanation of the Method of Minimum Contrast, see mincontrast.

The model is described in Jalilian et al (2013). It is a cluster process formed by taking a pattern of parent points, generated according to a Poisson process with intensity κ, and around each parent point, generating a random number of offspring points, such that the number of offspring of each parent is a Poisson random variable with mean μ, and the locations of the offspring points of one parent follow a common distribution described in Jalilian et al (2013).

If the argument lambda is provided, then this is used as the value of the point process intensity λ. Otherwise, if X is a point pattern, then λ will be estimated from X. If X is a summary statistic and lambda is missing, then the intensity λ cannot be estimated, and the parameter μ will be returned as NA.

The remaining arguments rmin,rmax,q,p control the method of minimum contrast; see mincontrast.

The corresponding model can be simulated using rCauchy.

For computational reasons, the optimisation procedure internally uses the parameter eta2, which is equivalent to 4 * scale^2 where scale is the scale parameter for the model as used in rCauchy.

Homogeneous or inhomogeneous Neyman-Scott/Cauchy models can also be fitted using the function kppm and the fitted models can be simulated using simulate.kppm.

The optimisation algorithm can be controlled through the additional arguments "..." which are passed to the optimisation function optim. For example, to constrain the parameter values to a certain range, use the argument method="L-BFGS-B" to select an optimisation algorithm that respects box constraints, and use the arguments lower and upper to specify (vectors of) minimum and maximum values for each parameter.

Value

An object of class "minconfit". There are methods for printing and plotting this object. It contains the following main components:

par	Vector of fitted parameter values.
fit	Function value table (object of class "fv") containing the observed values of the summary statistic (observed) and the theoretical values of the summary statistic computed from the fitted model parameters.

Author(s)

Abdollah Jalilian and Rasmus Waagepetersen. Adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu.a

References

Ghorbani, M. (2012) Cauchy cluster process. *Metrika*, to appear.

Jalilian, A., Guan, Y. and Waagepetersen, R. (2013) Decomposition of variance for spatial Cox processes. *Scandinavian Journal of Statistics* **40**, 119-137.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

kppm, cauchy.estK, lgcp.estpcf, thomas.estpcf, vargamma.estpcf, mincontrast, pcf, pcfmodel.

rCauchy to simulate the model.

Examples

```
u <- cauchy.estpcf(redwood)
u
plot(u, legendpos="topright")
```

CDF *Cumulative Distribution Function From Kernel Density Estimate*

Description

Given a kernel estimate of a probability density, compute the corresponding cumulative distribution function.

Usage

```
CDF(f, ...)

## S3 method for class 'density'
CDF(f, ..., warn = TRUE)
```

Arguments

f	Density estimate (object of class "density").
...	Ignored.
warn	Logical value indicating whether to issue a warning if the density estimate f had to be renormalised because it was computed in a restricted interval.

Details

CDF is generic, with a method for class "density".

This calculates the cumulative distribution function whose probability density has been estimated and stored in the object f. The object f must belong to the class "density", and would typically have been obtained from a call to the function density.

Value

A function, which can be applied to any numeric value or vector of values.

Author(s)

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

See Also

density, quantile.density

Examples

```
b <- density(runif(10))
f <- CDF(b)
f(0.5)
plot(f)
```

---

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)

## S3 method for class 'ppm'
cdf.test(model, covariate,  test=c("ks", "cvm", "ad"), ...,
          interpolate=TRUE, jitter=TRUE, nsim=99, verbose=TRUE)
```

```
## S3 method for class 'slrm'
cdf.test(model, covariate,  test=c("ks", "cvm", "ad"), ...,
         modelname=NULL, covname=NULL)
```

Arguments

X	A point pattern (object of class "ppp" or "lpp").
model	A fitted point process model (object of class "ppm" or "lppm") or fitted spatial logistic regression (object of class "slrm").
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.
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.
modelname,covname	Character strings giving alternative names for model and covariate to be used in labelling plot axes.
nsim	Number of simulated realisations from the model to be used for the Monte Carlo test, when model is not a Poisson process.
verbose	Logical value indicating whether to print progress reports when performing a Monte Carlo 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 <r.turner@auckland.ac.nz>

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)

# fit inhomogeneous Poisson model and test
model <- ppm(nztrees ~x)
cdf.test(model, "x")
```

```
if(interactive()) {
  # synthetic data: nonuniform Poisson process
  X <- rpoispp(function(x,y) { 100 * exp(x) }, win=square(1))

  # fit uniform Poisson process
  fit0 <- ppm(X ~1)
  # fit correct nonuniform Poisson process
  fit1 <- ppm(X ~x)

  # test wrong model
  cdf.test(fit0, "x")
  # test right model
  cdf.test(fit1, "x")
}

# multitype point pattern
cdf.test(amacrine, "x")
yimage <- as.im(function(x,y){y}, W=Window(amacrine))
cdf.test(ppm(amacrine ~marks+y), yimage)

options(op)
```

`cdf.test.mppm` *Spatial Distribution Test for Multiple Point Process Model*

Description

Performs a spatial distribution test of a point process model fitted to multiple spatial point patterns. The test compares the observed and predicted distributions of the values of a spatial covariate, using either the Kolmogorov-Smirnov, Cramér-von Mises or Anderson-Darling test of goodness-of-fit.

Usage

```
## S3 method for class 'mppm'
cdf.test(model, covariate, test=c("ks", "cvm", "ad"), ...,
            nsim=19, verbose=TRUE, interpolate=FALSE, fast=TRUE, jitter=TRUE)
```

Arguments

`model`	An object of class `"mppm"` representing a point process model fitted to multiple spatial point patterns.
`covariate`	The spatial covariate on which the test will be based. A function, a pixel image, a list of functions, a list of pixel images, a hyperframe, a character string containing the name of one of the covariates in `model`, or one of the strings `"x"` or `"y"`.
`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 cdf.test to control the test.
nsim	Number of simulated realisations which should be generated, if a Monte Carlo test is required.
verbose	Logical flag indicating whether to print progress reports.
interpolate	Logical flag indicating whether to interpolate between pixel values when code-covariate is a pixel image. See *Details*.
fast	Logical flag. If TRUE, values of the covariate are only sampled at the original quadrature points used to fit the model. If FALSE, values of the covariate are sampled at all pixels, which can be slower by three orders of magnitude.
jitter	Logical flag. If TRUE, observed values of the covariate are perturbed by adding small random values, to avoid tied observations.

Details

This function is a method for the generic function cdf.test for the class mppm.

This function performs a goodness-of-fit test of a point process model that has been fitted to multiple point patterns. 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, Cramér-von Mises or Anderson-Darling test of goodness-of-fit. These are exact tests if the model is Poisson; otherwise, for a Gibbs model, a Monte Carlo p-value is computed by generating simulated realisations of the model and applying the selected goodness-of-fit test to each simulation.

The argument model should be a fitted point process model fitted to multiple point patterns (object of class "mppm").

The argument covariate contains the values of a spatial function. It can be

- a function(x,y)
- a pixel image (object of class "im"
- a list of function(x,y), one for each point pattern
- a list of pixel images, one for each point pattern
- a hyperframe (see hyperframe) of which the first column will be taken as containing the covariate
- a character string giving the name of one of the covariates in model
- one of the character strings "x" or "y", indicating the spatial 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.

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. A goodness-of-fit test of the uniform distribution is applied to these numbers using ks.test, cvm.test or ad.test.

The argument interpolate determines how pixel values will be handled when codecovariate is a pixel image. The value of the covariate at a data point is obtained by looking up the value of the nearest pixel if interpolate=FALSE, or by linearly interpolating between the values of the four nearest pixels if interpolate=TRUE. Linear interpolation is slower, but is sometimes necessary to avoid tied values of the covariate arising when the pixel grid is coarse.

If model is a Poisson point process, then the Kolmogorov-Smirnov, Cramér-von Mises and Anderson-Darling tests are theoretically exact. This test was apparently first described (in the context of spatial data, and for Kolmogorov-Smirnov) by Berman (1986). See also Baddeley et al (2005).

If model is not a Poisson point process, then the Kolmogorov-Smirnov, Cramér-von Mises and Anderson-Darling tests are biased. Instead they are used as the basis of a Monte Carlo test. First nsim simulated realisations of the model will be generated. Each simulated realisation consists of a list of simulated point patterns, one for each of the original data patterns. This can take a very long time. The model is then re-fitted to each simulation, and the refitted model is subjected to the goodness-of-fit test described above. A Monte Carlo p-value is then computed by comparing the p-value of the original test with the p-values obtained from the simulations.

Value

An object of class "cdftest" and "htest" containing the results of the test. See cdf.test for details.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Ida-Maria Sintorn and Leanne Bischoff. Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

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

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

cdf.test, quadrat.test, mppm

Examples

```
# three i.i.d. realisations of nonuniform Poisson process
lambda <- as.im(function(x,y) { 200 * exp(x) }, square(1))
dat <- hyperframe(X=list(rpoispp(lambda), rpoispp(lambda), rpoispp(lambda)))

# fit uniform Poisson process
fit0 <- mppm(X~1, dat)
# fit correct nonuniform Poisson process
fit1 <- mppm(X~x, dat)

# test wrong model
cdf.test(fit0, "x")
# test right model
cdf.test(fit1, "x")

# Gibbs model
fitGibbs <- update(fit0, interaction=Strauss(0.05))
ns <- if(interactive()) 19 else 2
cdf.test(fitGibbs, "x", nsim=ns)
```

`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 <r.turner@auckland.ac.nz> 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="none",
                clipregion=NULL,
                alternative=c("two.sided", "less", "greater",
                              "clustered", "regular"),
                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`
`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.
`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 index is computed by the function `clarkevans`. See the help for `clarkevans` for information about the Clark-Evans index R and about the arguments `correction` and `clipregion`.

This command performs a hypothesis test of clustering or ordering of the point pattern `X`. 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 computed for the data as described in `clarkevans`.

If `correction="none"` and `nsim` is missing, the p-value for the test is computed by standardising R as proposed by Clark and Evans (1954) and referring the statistic to the standard Normal distribution.

Otherwise, the p-value for the test is computed by Monte Carlo simulation of `nsim` realisations of Complete Spatial Randomness conditional on the observed number of points.

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", nsim=39)
```

`closepaircounts` *Count Close Pairs of Points*

Description

Low-level functions to count the number of close pairs of points.

Usage

```
closepaircounts(X, r)

crosspaircounts(X, Y, r)
```

Arguments

`X,Y`	Point patterns (objects of class `"ppp"`).
`r`	Maximum distance between pairs of points to be counted as close pairs.

Details

These are the efficient low-level functions used by **spatstat** to count close pairs of points in a point pattern or between two point patterns.

`closepaircounts(X,r)` counts the number of neighbours for each point in the pattern `X`. That is, for each point `X[i]`, it counts the number of other points `X[j]` with `j != i` such that `d(X[i],X[j]) <= r` where `d` denotes Euclidean distance. The result is an integer vector `v` such that `v[i]` is the number of neighbours of `X[i]`.

`crosspaircounts(X,Y,r)` counts, for each point in the pattern `X`, the number of neighbours in the pattern `Y`. That is, for each point `X[i]`, it counts the number of points `Y[j]` such that `d(X[i],X[j]) <= r`. The result is an integer vector `v` such that `v[i]` is the number of neighbours of `X[i]` in the pattern `Y`.

Value

An integer vector of length equal to the number of points in `X`.

Warning about accuracy

The results of these functions may not agree exactly with the correct answer (as calculated by a human) and may not be consistent between different computers and different installations of R. The discrepancies arise in marginal cases where the interpoint distance is equal to, or very close to, the threshold `rmax`.

Floating-point numbers in a computer are not mathematical Real Numbers: they are approximations using finite-precision binary arithmetic. The approximation is accurate to a tolerance of about `.Machine$double.eps`.

If the true interpoint distance d and the threshold `rmax` are equal, or if their difference is no more than `.Machine$double.eps`, the result may be incorrect.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

closepairs to identify all close pairs of points.

Examples

```
a <- closepaircounts(cells, 0.1)
sum(a)
Y <- split(amacrine)
b <- crosspaircounts(Y$on, Y$off, 0.1)
```

---

clusterfield *Field of clusters*

Description

Calculate the superposition of cluster kernels at the location of a point pattern.

Usage

```
  ## S3 method for class 'kppm'
clusterfield(model, locations = NULL, ...)
```

Arguments

model	Cluster model. Either a fitted cluster model (object of class "kppm"), a character string specifying the type of cluster model, or a function defining the cluster kernel. See Details.
locations	A point pattern giving the locations of the kernels. Defaults to the centroid of the observation window for the "kppm" method and to the center of a unit square otherwise.
...	Additional arguments passed to density.ppp or the cluster kernel. See Details.

Details

The actual calculations are performed by density.ppp and ... arguments are passed thereto for control over the pixel resolution etc. (These arguments are then passed on to pixellate.ppp and as.mask.)

For the method clusterfield.function, the given kernel function should accept vectors of x and y coordinates as its first two arguments. Any additional arguments may be passed through the ....

The function method also accepts the optional parameter mu (defaulting to 1) specifying the mean number of points per cluster (as a numeric) or the inhomogeneous reference cluster intensity (as

an "im" object or a function(x,y)). The interpretation of mu is as explained in the simulation functions referenced in the See Also section below.

For the method clusterfield.character, the argument model must be one of the following character strings: model="Thomas" for the Thomas process, model="MatClust" for the Matérn cluster process, model="Cauchy" for the Neyman-Scott cluster process with Cauchy kernel, or model="VarGamma" for the Neyman-Scott cluster process with Variance Gamma kernel. For all these models the parameter scale is required and passed through ... as well as the parameter nu when model="VarGamma". This method calls clusterfield.function so the parameter mu may also be passed through ... and will be interpreted as explained above.

The method clusterfield.kppm extracts the relevant information from the fitted model (including mu) and calls clusterfield.function.

Value

A pixel image (object of class "im").

Author(s)

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

See Also

density.ppp and kppm.

Simulation algorithms for cluster models: rCauchy rMatClust rThomas rVarGamma

Examples

```
# method for fitted model
fit <- kppm(redwood~1, "Thomas")
clusterfield(fit, eps = 0.01)

# method for functions
kernel <- function(x,y,scal) {
    r <- sqrt(x^2 + y^2)
    ifelse(r > 0,
           dgamma(r, shape=5, scale=scal)/(2 * pi * r),
           0)
}
X <- runifpoint(10)
clusterfield(kernel, X, scal=0.05)
```

clusterfit *Fit Cluster or Cox Point Process Model via Minimum Contrast*

Description

Fit a homogeneous or inhomogeneous cluster process or Cox point process model to a point pattern by the Method of Minimum Contrast.

Usage

```
clusterfit(X, clusters, lambda = NULL, startpar = NULL, ...,
           q = 1/4, p = 2, rmin = NULL, rmax = NULL,
           ctrl=list(q=q, p=p, rmin=rmin, rmax=rmax),
           statistic = NULL, statargs = NULL, algorithm="Nelder-Mead",
           verbose=FALSE, pspace=NULL)
```

Arguments

X	Data to which the cluster or Cox model will be fitted. Either a point pattern or a summary statistic. See Details.
clusters	Character string determining the cluster or Cox model. Partially matched. Options are "Thomas", "MatClust", "Cauchy", "VarGamma" and "LGCP".
lambda	Optional. An estimate of the intensity of the point process. Either a single numeric specifying a constant intensity, 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.
startpar	Vector of initial values of the parameters of the point process mode. If X is a point pattern sensible defaults are used. Otherwise rather arbitrary values are used.
q,p	Optional. Exponents for the contrast criterion. See mincontrast.
rmin, rmax	Optional. The interval of r values for the contrast criterion. See mincontrast.
ctrl	Optional. Named list containing values of the parameters q,p,rmin,rmax.
...	Additional arguments passed to mincontrast.
statistic	Optional. Name of the summary statistic to be used for minimum contrast estimation: either "K" or "pcf".
statargs	Optional list of arguments to be used when calculating the statistic. See Details.
algorithm	Character string determining the mathematical optimisation algorithm to be used by optim. See the argument method of optim.
verbose	Logical value indicating whether to print detailed progress reports for debugging purposes.
pspace	For internal use by package code only.

Details

This function fits the clustering parameters of a cluster or Cox point process model by the Method of Minimum Contrast, that is, by matching the theoretical K-function of the model to the empirical K-function of the data, as explained in `mincontrast`.

If `statistic="pcf"` (or `X` appears to be an estimated pair correlation function) then instead of using the K-function, the algorithm will use the pair correlation function.

If `X` is a point pattern of class `"ppp"` an estimate of the summary statistic specfied by `statistic` (defaults to `"K"`) is first computed before minimum contrast estimation is carried out as described above. In this case the argument `statargs` can be used for controlling the summary statistic estimation. The precise algorithm for computing the summary statistic depends on whether the intensity specification (`lambda`) is:

homogeneous: If `lambda` is `NULl` or a single numeric the pattern is considered homogeneous and either `Kest` or `pcf` is invoked. In this case `lambda` is **not** used for anything when estimating the summary statistic.

inhomogeneous: If `lambda` is a pixel image (object of class `"im"`), a fitted point process model (object of class `"ppm"` or `"kppm"`) or a `function(x,y)` the pattern is considered inhomogeneous. In this case either `Kinhom` or `pcfinhom` is invoked with `lambda` as an argument.

After the clustering parameters of the model have been estimated by minimum contrast `lambda` (if non-null) is used to compute the additional model parameter μ.

The algorithm parameters `q,p,rmax,rmin` are described in the help for `mincontrast`. They may be provided either as individually-named arguments, or as entries in the list `ctrl`. The individually-named arguments `q,p,rmax,rmin` override the entries in the list `ctrl`.

Value

An object of class `"minconfit"`. There are methods for printing and plotting this object. See `mincontrast`.

Author(s)

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

References

Diggle, P.J. and Gratton, R.J. (1984) Monte Carlo methods of inference for implicit statistical models. *Journal of the Royal Statistical Society, series B* **46**, 193 – 212.

Møller, J. and Waagepetersen, R. (2003). Statistical Inference and Simulation for Spatial Point Processes. Chapman and Hall/CRC, Boca Raton.

Waagepetersen, R. (2007). An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63** (2007) 252–258.

See Also

`kppm`

Examples

```
fit <- clusterfit(redwood, "Thomas")
fit
if(interactive()){
  plot(fit)
}
K <- Kest(redwood)
fit2 <- clusterfit(K, "MatClust")
```

`clusterkernel` *Extract Cluster Offspring Kernel*

Description

Given a cluster point process model, this command returns the probability density of the cluster offspring.

Usage

```
## S3 method for class 'kppm'
clusterkernel(model, ...)
```

Arguments

`model`	Cluster model. Either a fitted cluster or Cox model (object of class `"kppm"`), or a character string specifying the type of cluster model.
`...`	Parameter values for the model, when `model` is a character string.

Details

Given a specification of a cluster point process model, this command returns a `function(x,y)` giving the two-dimensional probability density of the cluster offspring points assuming a cluster parent located at the origin.

Value

A function in the R language with arguments `x,y,...`.

Author(s)

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

See Also

`clusterfield`, `kppm`

Examples

```
fit <- kppm(redwood ~ x, "MatClust")
f <- clusterkernel(fit)
f(0.05, 0.02)

f <- clusterkernel("Thomas", kappa=10, scale=0.5)
f(0.1, 0.2)
```

`clusterradius` *Compute or Extract Effective Range of Cluster Kernel*

Description

Given a cluster point process model, this command returns a value beyond which the the probability density of the cluster offspring is neglible.

Usage

```
## S3 method for class 'kppm'
clusterradius(model, ..., thresh = NULL, precision = FALSE)
```

Arguments

`model`	Cluster model. Either a fitted cluster or Cox model (object of class `"kppm"`), or a character string specifying the type of cluster model.
`...`	Parameter values for the model, when `model` is a character string.
`thresh`	Numerical threshold relative to the cluster kernel value at the origin (parent location) determining when the cluster kernel will be considered neglible. A sensible default is provided.
`precision`	Logical. If `precision=TRUE` the precision of the calculated range is returned as an attribute to the range. See details.

Details

Given a cluster model this function by default returns the effective range of the model with the given parameters as used in spatstat. For the Matérn cluster model (see e.g. `rMatClust`) this is simply the finite radius of the offsring density given by the paramter `scale` irrespective of other options given to this function. The remaining models in spatstat have infinite theoretical range, and an effective finite value is given as follows: For the Thomas model (see e.g. `rThomas` the default is `4*scale` where scale is the scale or standard deviation parameter of the model. If `thresh` is given the value is instead found as described for the other models below.

For the Cauchy model (see e.g. `rCauchy`) and the Variance Gamma (Bessel) model (see e.g. `rVarGamma`) the value of `thresh` defaults to 0.001, and then this is used to compute the range numerically as follows. If $k(x, y) = k_0(r)$ with $r = \sqrt{(x^2 + y^2)}$ denotes the isotropic cluster

kernel then $f(r) = 2\pi r k_0(r)$ is the density function of the offspring distance from the parent. The range is determined as the value of r where $f(r)$ falls below `thresh` times $k_0(r)$.

If `precision=TRUE` the precision related to the chosen range is returned as an attribute. Here the precision is defined as the polar integral of the kernel from distance 0 to the calculated range. Ideally this should be close to the value 1 which would be obtained for the true theretical infinite range.

Value

A positive numeric.

Additionally, the precision related to this range value is returned as an attribute `"prec"`, if `precision=TRUE`.

Author(s)

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

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

`clusterkernel`, `kppm`, `rMatClust`, `rThomas`, `rCauchy`, `rVarGamma`, `rNeymanScott`.

Examples

```
fit <- kppm(redwood ~ x, "MatClust")
clusterradius(fit)

clusterradius("Thomas", scale = .1)
clusterradius("Thomas", scale = .1, thresh = 0.001)
clusterradius("VarGamma", scale = .1, nu = 2, precision = TRUE)
```

---

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 <r.turner@auckland.ac.nz>

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)
```

---

coef.mppm *Coefficients of Point Process Model Fitted to Multiple Point Patterns*

Description

Given a point process model fitted to a list of point patterns, extract the coefficients of the fitted model. A method for coef.

Usage

```
  ## S3 method for class 'mppm'
coef(object, ...)
```

Arguments

object	The fitted point process model (an object of class "mppm")
...	Ignored.

Details

This function is a method for the generic function coef.

The argument object must be a fitted point process model (object of class "mppm") produced by the fitting algorithm mppm). This represents a point process model that has been fitted to a list of several point pattern datasets. See mppm for information.

This function extracts the vector of coefficients of the fitted model. This is the estimate of the parameter vector θ such that the conditional intensity of the model is of the form

$$\lambda(u, x) = \exp(\theta S(u, x))$$

where $S(u, x)$ is a (vector-valued) statistic.

For example, if the model object is the uniform Poisson process, then coef(object) will yield a single value (named "(Intercept)") which is the logarithm of the fitted intensity of the Poisson process.

If the fitted model includes random effects (i.e. if the argument random was specified in the call to mppm), then the fitted coefficients are different for each point pattern in the original data, so coef(object) is a data frame with one row for each point pattern, and one column for each parameter. Use fixef.mppm to extract the vector of fixed effect coefficients, and ranef.mppm to extract the random effect coefficients at each level.

Use print.mppm to print a more useful description of the fitted model.

Value

Either a vector containing the fitted coefficients, or a data frame containing the fitted coefficients for each point pattern.

Author(s)

Adrian Baddeley, Ida-Maria Sintorn and Leanne Bischoff. Implemented in **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

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

See Also

fixef.mppm and ranef.mppm for the fixed and random effect coefficients in a model that includes random effects.

print.mppm, mppm

Examples

```
H <- hyperframe(X=waterstriders)

fit.Poisson <- mppm(X ~ 1, H)
coef(fit.Poisson)

# The single entry "(Intercept)"
# is the log of the fitted intensity of the Poisson process

fit.Strauss <- mppm(X~1, H, Strauss(7))
coef(fit.Strauss)

# The two entries "(Intercept)" and "Interaction"
# are respectively log(beta) and log(gamma)
# in the usual notation for Strauss(beta, gamma, r)

# Tweak data to exaggerate differences
H$X[[1]] <- rthin(H$X[[1]], 0.3)
# Model with random effects
fitran <- mppm(X ~ 1, H, random=~1|id)
coef(fitran)
```

`coef.ppm` *Coefficients of Fitted Point Process Model*

Description

Given a point process model fitted to a point pattern, extract the coefficients of the fitted model. A method for `coef`.

Usage

```
  ## S3 method for class 'ppm'
coef(object, ...)
```

Arguments

`object`	The fitted point process model (an object of class `"ppm"`)
`...`	Ignored.

Details

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

The argument `object` must be a fitted point process model (object of class `"ppm"`). Such objects are produced by the maximum pseudolikelihood fitting algorithm `ppm`).

This function extracts the vector of coefficients of the fitted model. This is the estimate of the parameter vector θ such that the conditional intensity of the model is of the form

$$\lambda(u, x) = \exp(\theta S(u, x))$$

where $S(u, x)$ is a (vector-valued) statistic.

For example, if the model `object` is the uniform Poisson process, then `coef(object)` will yield a single value (named `"(Intercept)"`) which is the logarithm of the fitted intensity of the Poisson process.

Use `print.ppm` to print a more useful description of the fitted model.

Value

A vector containing the fitted coefficients.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`print.ppm`, `ppm.object`, `ppm`

Examples

```
data(cells)

poi <- ppm(cells, ~1, Poisson())
coef(poi)
# This is the log of the fitted intensity of the Poisson process

stra <- ppm(cells, ~1, Strauss(r=0.07))
coef(stra)

# The two entries "(Intercept)" and "Interaction"
# are respectively log(beta) and log(gamma)
# in the usual notation for Strauss(beta, gamma, r)
```

---

coef.slrm *Coefficients of Fitted Spatial Logistic Regression Model*

Description

Extracts the coefficients (parameters) from a fitted Spatial Logistic Regression model.

Usage

```
  ## S3 method for class 'slrm'
coef(object, ...)
```

Arguments

object	a fitted spatial logistic regression model. An object of class "slrm".
...	Ignored.

Details

This is a method for coef for fitted spatial logistic regression models (objects of class "slrm", usually obtained from the function slrm).

It extracts the fitted canonical parameters, i.e.\ the coefficients in the linear predictor of the spatial logistic regression.

Value

Numeric vector of coefficients.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> <adrian@maths.uwa.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

slrm

Examples

```
X <- rpoispp(42)
fit <- slrm(X ~ x+y)
coef(fit)
```

---

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 <r.turner@auckland.ac.nz>

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
```

---

compareFit *Residual Diagnostics for Multiple Fitted Models*

Description

Compares several fitted point process models using the same residual diagnostic.

Usage

```
compareFit(object, Fun, r = NULL, breaks = NULL, ...,
         trend = ~1, interaction = Poisson(), rbord = NULL,
         modelnames = NULL, same = NULL, different = NULL)
```

Arguments

object	Object or objects to be analysed. Either a fitted point process model (object of class "ppm"), a point pattern (object of class "ppp"), or a list of these objects.
Fun	Diagnostic function to be computed for each model. One of the functions Kcom, Kres, Gcom, Gres, psst, psstA or psstG or a string containing one of these names.
r	Optional. Vector of values of the argument r at which the diagnostic should be computed. This argument is usually not specified. There is a sensible default.
breaks	Optional alternative to r for advanced use.
...	Extra arguments passed to Fun.
trend,interaction,rbord	Optional. Arguments passed to ppm to fit a point process model to the data, if object is a point pattern or list of point patterns. See ppm for details. Each of these arguments can be a list, specifying different trend, interaction and/or rbord values to be used to generate different fitted models.
modelnames	Character vector. Short descriptive names for the different models.
same,different	Character strings or character vectors passed to collapse.fv to determine the format of the output.

Details

This is a convenient way to collect diagnostic information for several different point process models fitted to the same point pattern dataset, or for point process models of the same form fitted to several different datasets, etc.

The first argument, `object`, is usually a list of fitted point process models (objects of class `"ppm"`), obtained from the model-fitting function `ppm`.

For convenience, `object` can also be a list of point patterns (objects of class `"ppp"`). In that case, point process models will be fitted to each of the point pattern datasets, by calling `ppm` using the arguments `trend` (for the first order trend), `interaction` (for the interpoint interaction) and `rbord` (for the erosion distance in the border correction for the pseudolikelihood). See `ppm` for details of these arguments.

Alternatively `object` can be a single point pattern (object of class `"ppp"`) and one or more of the arguments `trend`, `interaction` or `rbord` can be a list. In this case, point process models will be fitted to the same point pattern dataset, using each of the model specifications listed.

The diagnostic function `Fun` will be applied to each of the point process models. The results will be collected into a single function value table. The `modelnames` are used to label the results from each fitted model.

Value

Function value table (object of class `"fv"`).

Author(s)

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

See Also

`ppm`, `Kcom`, `Kres`, `Gcom`, `Gres`, `psst`, `psstA`, `psstG`, `collapse.fv`

Examples

```
nd <- 40

ilist <- list(Poisson(), Geyer(7, 2), Strauss(7))
iname <- c("Poisson", "Geyer", "Strauss")

K <- compareFit(swedishpines, Kcom, interaction=ilist, rbord=9,
         correction="translate",
         same="trans", different="tcom", modelnames=iname, nd=nd)
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 <r.turner@auckland.ac.nz>

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 <r.turner@auckland.ac.nz>

See Also

eval.fv

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")

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

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.

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. 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.

Examples

```
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)
```

Concom *The Connected Component Process Model*

Description

Creates an instance of the Connected Component point process model which can then be fitted to point pattern data.

Usage

```
Concom(r)
```

Arguments

r	Threshold distance

Details

This function defines the interpoint interaction structure of a point process called the connected component process. It can be used to fit this model to point pattern data.

The function ppm(), which fits point process models to point pattern data, requires an argument of class "interact" describing the interpoint interaction structure of the model to be fitted. The appropriate description of the connected component interaction is yielded by the function Concom(). See the examples below.

In **standard form**, the connected component process (Baddeley and Møller, 1989) with disc radius r, intensity parameter κ and interaction parameter γ is a point process with probability density

$$f(x_1, \ldots, x_n) = \alpha \kappa^{n(x)} \gamma^{-C(x)}$$

for a point pattern x, where $x_1, \ldots, x_n$ represent the points of the pattern, $n(x)$ is the number of points in the pattern, and $C(x)$ is defined below. Here α is a normalising constant.

To define the term C(x), suppose that we construct a planar graph by drawing an edge between each pair of points x_i, x_j which are less than r units apart. Two points belong to the same connected component of this graph if they are joined by a path in the graph. Then $C(x)$ is the number of connected components of the graph.

The interaction parameter γ can be any positive number. If $\gamma = 1$ then the model reduces to a Poisson process with intensity κ. If $\gamma < 1$ then the process is regular, while if $\gamma > 1$ the process is clustered. Thus, a connected-component interaction process can be used to model either clustered or regular point patterns.

In **spatstat**, the model is parametrised in a different form, which is easier to interpret. In **canonical form**, the probability density is rewritten as

$$f(x_1, \ldots, x_n) = \alpha \beta^{n(x)} \gamma^{-U(x)}$$

where β is the new intensity parameter and $U(x) = C(x) - n(x)$ is the interaction potential. In this formulation, each isolated point of the pattern contributes a factor β to the probability density

(so the first order trend is β). The quantity $U(x)$ is a true interaction potential, in the sense that $U(x) = 0$ if the point pattern x does not contain any points that lie close together.

When a new point u is added to an existing point pattern x, the rescaled potential $-U(x)$ increases by zero or a positive integer. The increase is zero if u is not close to any point of x. The increase is a positive integer k if there are k different connected components of x that lie close to u. Addition of the point u contributes a factor $\beta\eta^\delta$ to the probability density, where δ is the increase in potential.

If desired, the original parameter κ can be recovered from the canonical parameter by $\kappa = \beta\gamma$.

The *nonstationary* connected component process is similar except that the contribution of each individual point x_i is a function $\beta(x_i)$ of location, rather than a constant beta.

Note the only argument of `Concom()` is the threshold distance `r`. When `r` is fixed, the model becomes an exponential family. The canonical parameters $\log(\beta)$ and $\log(\gamma)$ are estimated by `ppm()`, not fixed in `Concom()`.

Value

An object of class `"interact"` describing the interpoint interaction structure of the connected component process with disc radius r.

Edge correction

The interaction distance of this process is infinite. There are no well-established procedures for edge correction for fitting such models, and accordingly the model-fitting function `ppm` will give an error message saying that the user must specify an edge correction. A reasonable solution is to use the border correction at the same distance `r`, as shown in the Examples.

Author(s)

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

References

Baddeley, A.J. and Møller, J. (1989) Nearest-neighbour Markov point processes and random sets. *International Statistical Review* **57**, 89–121.

See Also

`ppm`, `pairwise.family`, `ppm.object`

Examples

```
# prints a sensible description of itself
Concom(r=0.1)

# Fit the stationary connected component process to redwood data
ppm(redwood, ~1, Concom(r=0.07), rbord=0.07)

# Fit the stationary connected component process to `cells' data
ppm(cells, ~1, Concom(r=0.06), rbord=0.06)
```

```
# eta=0 indicates hard core process.

# Fit a nonstationary connected component model
# with log-cubic polynomial trend
# ppm(swedishpines, ~polynom(x/10,y/10,3), Concom(r=7), rbord=7)
```

`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 = "everything", 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 image domains are intersected, and converted to a common pixel resolution. 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 `i`th and `j`th 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`).

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 <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

cor, cov

pairs.im

Examples

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

---

data.ppm *Extract Original Data from a Fitted Point Process Model*

Description

Given a fitted point process model, this function extracts the original point pattern dataset to which the model was fitted.

Usage

```
data.ppm(object)
```

Arguments

object	fitted point process model (an object of class "ppm").

Details

An object of class "ppm" represents a point process model that has been fitted to data. It is typically produced by the model-fitting algorithm ppm. The object contains complete information about the original data point pattern to which the model was fitted. This function extracts the original data pattern.

See ppm.object for a list of all operations that can be performed on objects of class "ppm".

Value

A point pattern (object of class "ppp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>.

See Also

ppm.object, ppp.object

Examples

```
fit <- ppm(cells, ~1, Strauss(r=0.1))
X <- data.ppm(fit)
# 'X' is identical to 'cells'
```

`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 <r.turner@auckland.ac.nz> 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, Boca Raton, FL.

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 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., Hardegen, A., Lawrence, T., Milne, R.K. and Nair, G. (2015) *Pushing the envelope*. In preparation.

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,
        kernel="gaussian",
        scalekernel=is.character(kernel),
        positive=FALSE, verbose=TRUE)
```

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.
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.

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).

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 <r.turner@auckland.ac.nz> 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.

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 <r.turner@auckland.ac.nz> 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"), or an expression.

For convenience, weights can also be a single expression or a single pixel image (object of class "im").

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 <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

ppp.object, im.object

Examples

```
Z <- density(split(amacrine), 0.05)
plot(Z)
```

`densityAdaptiveKernel` *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

```
densityAdaptiveKernel(X, ...)

## 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`.
`...`	Arguments passed to `bw.abram` 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 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` 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

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")
```

---

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 <r.turner@auckland.ac.nz> 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 (2021).

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 (2021) 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, 2021) 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, 2021).

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, 2021, 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. (2021) Diffusion smoothing for spatial point patterns. *Statistical Science*, in press.

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 <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk> and Mehdi Moradi.

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 <r.turner@auckland.ac.nz>

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)
```

`detpointprocfamilyfun` *Construct a New Determinantal Point Process Model Family Function*

Description

Function to ease the implementation of a new determinantal point process model family.

Usage

```
detpointprocfamilyfun(kernel = NULL,
    specden = NULL, basis = "fourierbasis",
    convkernel = NULL, Kfun = NULL, valid = NULL, intensity = NULL,
    dim = 2, name = "User-defined", isotropic = TRUE, range = NULL,
    parbounds = NULL, specdenrange = NULL, startpar = NULL, ...)
```

Arguments

`kernel`	function specifying the kernel. May be set to `NULL`. See Details.
`specden`	function specifying the spectral density. May be set to `NULL`. See Details.
`basis`	character string giving the name of the basis. Defaults to the Fourier basis. See Details.
`convkernel`	function specifying the k-fold auto-convolution of the kernel. May be set to `NULL`. See Details.
`Kfun`	function specifying the K-function. May be set to `NULL`. See Details.
`valid`	function determining whether a given set of parameter values yields a valid model. May be set to `NULL`. See Examples.
`intensity`	character string specifying which parameter is the intensity in the model family. Should be `NULL` if the model family has no intensity parameter.
`dim`	character strig specifying which parameter is the dimension of the state space in this model family (if any). Alternatively a positive integer specifying the dimension.
`name`	character string giving the name of the model family used for printing.
`isotropic`	logical value indicating whether or not the model is isotropic.
`range`	function determining the interaction range of the model. May be set to `NULL`. See Examples.
`parbounds`	function determining the bounds for each model parameter when all other parameters are fixed. May be set to `NULL`. See Examples.
`specdenrange`	function specifying the the range of the spectral density if it is finite (only the case for very few models). May be set to `NULL`.
`startpar`	function determining starting values for parameters in any estimation algorithm. May be set to `NULL`. See Examples.
`...`	Additional arguments for inclusion in the returned model object. These are not checked in any way.

Details

A determinantal point process family is specified either in terms of a kernel (a positive semi-definite function, i.e. a covariance function) or a spectral density, or preferably both. One of these can be NULL if it is unknown, but not both. When both are supplied they must have the same arguments. The first argument gives the values at which the function should be evaluated. In general the function should accept an n by d matrix or data.frame specifying $n(>= 0)$ points in dimension d. If the model is isotropic it only needs to accept a non-negative valued numeric of length n. (In fact there is currently almost no support for non-isotropic models, so it is recommended not to specify such a model.) The name of this argument could be chosen freely, but x is recommended. The remaining arguments are the parameters of the model. If one of these is an intensity parameter the name should be mentioned in the argument intensity. If one of these specifies the dimension of the model it should be mentioned in the argument dim.

The kernel and spectral density is with respect to a specific set of basis functions, which is typically the Fourier basis. However this can be changed to any user-supplied basis in the argument basis. If such an alternative is supplied it must be the name of a function expecting the same arguments as fourierbasis and returning the results in the same form as fourierbasis.

If supplied, the arguments of convkernel must obey the following: first argument should be like the first argument of kernel and/or specden (see above). The second argument (preferably called k) should be the positive integer specifying how many times the auto-convolution is done (i.e. the k in k-fold auto-convolution). The remaining arguments must agree with the arguments of kernel and/or specden (see above).

If supplied, the arguments of Kfun should be like the arguments of kernel and specden (see above).

Value

A function in the R language, belonging to the class "detpointprocfamilyfun". The function has formal arguments ... and returns a determinantal point process family (object of class "detpointprocfamily").

Author(s)

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

Examples

```
  ## Example of how to define the Gauss family
exGauss <- detpointprocfamilyfun(
    name="Gaussian",
    kernel=function(x, lambda, alpha, d){
        lambda*exp(-(x/alpha)^2)
    },
    specden=function(x, lambda, alpha, d){
        lambda * (sqrt(pi)*alpha)^d * exp(-(x*alpha*pi)^2)
    },
    convkernel=function(x, k, lambda, alpha, d){
        logres <- k*log(lambda*pi*alpha^2) - log(pi*k*alpha^2) - x^2/(k*alpha^2)
        return(exp(logres))
    },
```

```
  Kfun = function(x, lambda, alpha, d){
      pi*x^2 - pi*alpha^2/2*(1-exp(-2*x^2/alpha^2))
  },
  valid=function(lambda, alpha, d){
      lambda>0 && alpha>0 && d>=1 && lambda <= (sqrt(pi)*alpha)^(-d)
  },
  isotropic=TRUE,
  intensity="lambda",
  dim="d",
  range=function(alpha, bound = .99){
      if(missing(alpha))
          stop("The parameter alpha is missing.")
      if(!(is.numeric(bound)&&bound>0&&bound<1))
          stop("Argument bound must be a numeric between 0 and 1.")
      return(alpha*sqrt(-log(sqrt(1-bound))))
  },
  parbounds=function(name, lambda, alpha, d){
      switch(name,
             lambda = c(0, (sqrt(pi)*alpha)^(-d)),
             alpha = c(0, lambda^(-1/d)/sqrt(pi)),
             stop("Parameter name misspecified")
             )
  },
  startpar=function(model, X){
      rslt <- NULL
      if("lambda" %in% model$freepar){
          lambda <- intensity(X)
          rslt <- c(rslt, "lambda" = lambda)
          model <- update(model, lambda=lambda)
      }
      if("alpha" %in% model$freepar){
          alpha <- .8*dppparbounds(model, "alpha")[2]
          rslt <- c(rslt, "alpha" = alpha)
      }
      return(rslt)
  }
  )
exGauss
m <- exGauss(lambda=100, alpha=.05, d=2)
m
```

---

dfbetas.ppm *Parameter Influence Measure*

Description

Computes the deletion influence measure for each parameter in a fitted point process model.

Usage

```
## S3 method for class 'ppm'
dfbetas(model, ...,
       drop = FALSE, iScore=NULL, iHessian=NULL, iArgs=NULL)
```

Arguments

model	Fitted point process model (object of class "ppm").
...	Ignored, except for the arguments dimyx and eps which are passed to as.mask to control the spatial resolution of the image of the density component.
drop	Logical. Whether to include (drop=FALSE) or exclude (drop=TRUE) contributions from quadrature points that were not used to fit the model.
iScore,iHessian	Components of the score vector and Hessian matrix for the irregular parameters, if required. See Details.
iArgs	List of extra arguments for the functions iScore, iHessian if required.

Details

Given a fitted spatial point process model, this function computes the influence measure for each parameter, as described in Baddeley, Chang and Song (2013) and Baddeley, Rubak and Turner (2019).

This is a method for the generic function dfbetas.

The influence measure for each parameter θ is a signed measure in two-dimensional space. It consists of a discrete mass on each data point (i.e. each point in the point pattern to which the model was originally fitted) and a continuous density at all locations. The mass at a data point represents the change in the fitted value of the parameter θ that would occur if this data point were to be deleted. The density at other non-data locations represents the effect (on the fitted value of θ) of deleting these locations (and their associated covariate values) from the input to the fitting procedure.

If the point process model trend has irregular parameters that were fitted (using ippm) then the influence calculation requires the first and second derivatives of the log trend with respect to the irregular parameters. The argument iScore should be a list, with one entry for each irregular parameter, of R functions that compute the partial derivatives of the log trend (i.e. log intensity or log conditional intensity) with respect to each irregular parameter. The argument iHessian should be a list, with p^2 entries where p is the number of irregular parameters, of R functions that compute the second order partial derivatives of the log trend with respect to each pair of irregular parameters.

Value

An object of class "msr" representing a signed or vector-valued measure. This object can be printed and plotted.

Author(s)

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

References

Baddeley, A. and Chang, Y.M. and Song, Y. (2013) Leverage and influence diagnostics for spatial point process models. *Scandinavian Journal of Statistics* **40**, 86–104.

Baddeley, A., Rubak, E. and Turner, R. (2019) Leverage and influence diagnostics for Gibbs spatial point processes. *Spatial Statistics* **29**, 15–48.

See Also

`leverage.ppm`, `influence.ppm`, `ppmInfluence`.

See `msr` for information on how to use a measure.

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X ~x+y)

plot(dfbetas(fit))
plot(Smooth(dfbetas(fit)))
```

`dffit.ppm` *Case Deletion Effect Measure of Fitted Model*

Description

Computes the case deletion effect measure `DFFIT` for a fitted model.

Usage

```
dffit(object, ...)

## S3 method for class 'ppm'
dffit(object, ..., collapse = FALSE, dfb = NULL)
```

Arguments

`object`	A fitted model, such as a point process model (object of class `"ppm"`).
`...`	Additional arguments passed to `dfbetas.ppm`.
`collapse`	Logical value specifying whether to collapse the vector-valued measure to a scalar-valued measure by adding all the components.
`dfb`	Optional. The result of `dfbetas(object)`, if it has already been computed.

Details

The case deletion effect measure DFFIT is a model diagnostic traditionally used for regression models. In that context, DFFIT[i,j] is the negative change, in the value of the jth term in the linear predictor, that would occur if the ith data value was deleted. It is closely related to the diagnostic DFBETA.

For a spatial point process model, dffit computes the analogous spatial case deletion diagnostic, described in Baddeley, Rubak and Turner (2019).

Value

A measure (object of class "msr").

Author(s)

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

References

Baddeley, A., Rubak, E. and Turner, R. (2019) Leverage and influence diagnostics for Gibbs spatial point processes. *Spatial Statistics* **29**, 15–48.

See Also

dfbetas.ppm

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X ~x+y)

plot(dffit(fit))
plot(dffit(fit, collapse=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 <r.turner@auckland.ac.nz> 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, Boca Raton, FL.

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 <r.turner@auckland.ac.nz> 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, Boca Raton, FL.

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 <r.turner@auckland.ac.nz> 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, Boca Raton, FL.

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 <r.turner@auckland.ac.nz> 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*, in press.

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)
```

`diagnose.ppm` *Diagnostic Plots for Fitted Point Process Model*

Description

Given a point process model fitted to a point pattern, produce diagnostic plots based on residuals.

Usage

```
diagnose.ppm(object, ..., type="raw", which="all", sigma=NULL,
             rbord=reach(object), cumulative=TRUE,
             plot.it=TRUE, rv = NULL,
             compute.sd=is.poisson(object), compute.cts=TRUE,
             envelope=FALSE, nsim=39, nrank=1,
             typename, check=TRUE, repair=TRUE,
             oldstyle=FALSE, splineargs=list(spar=0.5))

## S3 method for class 'diagppm'
plot(x, ..., which,
             plot.neg=c("image", "discrete", "contour", "imagecontour"),
             plot.smooth=c("imagecontour", "image", "contour", "persp"),
             plot.sd, spacing=0.1, outer=3,
             srange=NULL, monochrome=FALSE, main=NULL)
```

Arguments

`object`	The fitted point process model (an object of class `"ppm"`) for which diagnostics should be produced. This object is usually obtained from `ppm`.
`type`	String indicating the type of residuals or weights to be used. Current options are `"eem"` for the Stoyan-Grabarnik exponential energy weights, `"raw"` for the raw residuals, `"inverse"` for the inverse-lambda residuals, and `"pearson"` for the Pearson residuals. A partial match is adequate.
`which`	Character string or vector indicating the choice(s) of plots to be generated. Options are `"all"`, `"marks"`, `"smooth"`, `"x"`, `"y"` and `"sum"`. Multiple choices may be given but must be matched exactly. See Details.
`sigma`	Bandwidth for kernel smoother in `"smooth"` option.
`rbord`	Width of border to avoid edge effects. The diagnostic calculations will be confined to those points of the data pattern which are at least `rbord` units away from the edge of the window. (An infinite value of `rbord` will be ignored.)
`cumulative`	Logical flag indicating whether the lurking variable plots for the x and y coordinates will be the plots of cumulative sums of marks (`cumulative=TRUE`) or the plots of marginal integrals of the smoothed residual field (`cumulative=FALSE`).
`plot.it`	Logical value indicating whether plots should be shown. If `plot.it=FALSE`, the computed diagnostic quantities are returned without plotting them.
`plot.neg`	String indicating how the density part of the residual measure should be plotted.
`plot.smooth`	String indicating how the smoothed residual field should be plotted.
`compute.sd,plot.sd`	Logical values indicating whether error bounds should be computed and added to the `"x"` and `"y"` plots. The default is `TRUE` for Poisson models and `FALSE` for non-Poisson models. See Details.
`envelope,nsim,nrank`	Arguments passed to `lurking` in order to plot simulation envelopes for the lurking variable plots.

rv	Usually absent. Advanced use only. If this argument is present, the values of the residuals will not be calculated from the fitted model object but will instead be taken directly from rv.
spacing	The spacing between plot panels (when a four-panel plot is generated) expressed as a fraction of the width of the window of the point pattern.
outer	The distance from the outermost line of text to the nearest plot panel, expressed as a multiple of the spacing between plot panels.
srange	Vector of length 2 that will be taken as giving the range of values of the smoothed residual field, when generating an image plot of this field. This is useful if you want to generate diagnostic plots for two different fitted models using the same colour map.
monochrome	Flag indicating whether images should be displayed in greyscale (suitable for publication) or in colour (suitable for the screen). The default is to display in colour.
check	Logical value indicating whether to check the internal format of object. If there is any possibility that this object has been restored from a dump file, or has otherwise lost track of the environment where it was originally computed, set check=TRUE.
repair	Logical value indicating whether to repair the internal format of object, if it is found to be damaged.
oldstyle	Logical flag indicating whether error bounds should be plotted using the approximation given in the original paper (oldstyle=TRUE), or using the correct asymptotic formula (oldstyle=FALSE).
splineargs	Argument passed to lurking to control the smoothing in the lurking variable plot.
x	The value returned from a previous call to diagnose.ppm. An object of class "diagppm".
typename	String to be used as the name of the residuals.
main	Main title for the plot.
...	Extra arguments, controlling either the resolution of the smoothed image (passed from diagnose.ppm to density.ppp) or the appearance of the plots (passed from diagnose.ppm to plot.diagppm and from plot.diagppm to plot.default).
compute.cts	Advanced use only.

Details

The function diagnose.ppm generates several diagnostic plots for a fitted point process model. The plots display the residuals from the fitted model (Baddeley et al, 2005) or alternatively the 'exponential energy marks' (Stoyan and Grabarnik, 1991). These plots can be used to assess goodness-of-fit, to identify outliers in the data, and to reveal departures from the fitted model. See also the companion function qqplot.ppm.

The argument object must be a fitted point process model (object of class "ppm") typically produced by the maximum pseudolikelihood fitting algorithm ppm).

The argument type selects the type of residual or weight that will be computed. Current options are:

`"eem"`**:** exponential energy marks (Stoyan and Grabarnik, 1991) computed by `eem`. These are positive weights attached to the data points (i.e. the points of the point pattern dataset to which the model was fitted). If the fitted model is correct, then the sum of these weights for all data points in a spatial region B has expected value equal to the area of B. See `eem` for further explanation.

`"raw"`**,** `"inverse"` **or** `"pearson"`**:** point process residuals (Baddeley et al, 2005) computed by the function `residuals.ppm`. These are residuals attached both to the data points and to some other points in the window of observation (namely, to the dummy points of the quadrature scheme used to fit the model). If the fitted model is correct, then the sum of the residuals in a spatial region B has mean zero. The options are

- `"raw"`: the raw residuals;
- `"inverse"`: the 'inverse-lambda' residuals, a counterpart of the exponential energy weights;
- `"pearson"`: the Pearson residuals.

See `residuals.ppm` for further explanation.

The argument `which` selects the type of plot that is produced. Options are:

`"marks"`**:** plot the residual measure. For the exponential energy weights (`type="eem"`) this displays circles centred at the points of the data pattern, with radii proportional to the exponential energy weights. For the residuals (`type="raw"`, `type="inverse"` or `type="pearson"`) this again displays circles centred at the points of the data pattern with radii proportional to the (positive) residuals, while the plotting of the negative residuals depends on the argument `plot.neg`. If `plot.neg="image"` then the negative part of the residual measure, which is a density, is plotted as a colour image. If `plot.neg="discrete"` then the discretised negative residuals (obtained by approximately integrating the negative density using the quadrature scheme of the fitted model) are plotted as squares centred at the dummy points with side lengths proportional to the (negative) residuals. [To control the size of the circles and squares, use the argument `maxsize`.]

`"smooth"`**:** plot a kernel-smoothed version of the residual measure. Each data or dummy point is taken to have a 'mass' equal to its residual or exponential energy weight. (Note that residuals can be negative). This point mass is then replaced by a bivariate isotropic Gaussian density with standard deviation `sigma`. The value of the smoothed residual field at any point in the window is the sum of these weighted densities. If the fitted model is correct, this smoothed field should be flat, and its height should be close to 0 (for the residuals) or 1 (for the exponential energy weights). The field is plotted either as an image, contour plot or perspective view of a surface, according to the argument `plot.smooth`. The range of values of the smoothed field is printed if the option `which="sum"` is also selected.

`"x"`**:** produce a 'lurking variable' plot for the x coordinate. This is a plot of $h(x)$ against x (solid lines) and of $E(h(x))$ against x (dashed lines), where $h(x)$ is defined below, and $E(h(x))$ denotes the expectation of $h(x)$ assuming the fitted model is true.

- if `cumulative=TRUE` then $h(x)$ is the cumulative sum of the weights or residuals for all points which have X coordinate less than or equal to x. For the residuals $E(h(x)) = 0$, and for the exponential energy weights $E(h(x))$ = area of the subset of the window to the left of the line $X = x$.
- if `cumulative=FALSE` then $h(x)$ is the marginal integral of the smoothed residual field (see the case `which="smooth"` described above) on the x axis. This is approximately the derivative of the plot for `cumulative=TRUE`. The value of $h(x)$ is computed by summing

the values of the smoothed residual field over all pixels with the given x coordinate. For the residuals $E(h(x)) = 0$, and for the exponential energy weights $E(h(x)) =$ length of the intersection between the observation window and the line $X = x$.

If `plot.sd = TRUE`, then superimposed on the lurking variable plot are the pointwise two-standard-deviation error limits for $h(x)$ calculated for the inhomogeneous Poisson process. The default is `plot.sd = TRUE` for Poisson models and `plot.sd = FALSE` for non-Poisson models.

`"y"`: produce a similar lurking variable plot for the y coordinate.

`"sum"`: print the sum of the weights or residuals for all points in the window (clipped by a margin `rbord` if required) and the area of the same window. If the fitted model is correct the sum of the exponential energy weights should equal the area of the window, while the sum of the residuals should equal zero. Also print the range of values of the smoothed field displayed in the `"smooth"` case.

`"all"`: All four of the diagnostic plots listed above are plotted together in a two-by-two display. Top left panel is `"marks"` plot. Bottom right panel is `"smooth"` plot. Bottom left panel is `"x"` plot. Top right panel is `"y"` plot, rotated 90 degrees.

The argument `rbord` ensures there are no edge effects in the computation of the residuals. The diagnostic calculations will be confined to those points of the data pattern which are at least `rbord` units away from the edge of the window. The value of `rbord` should be greater than or equal to the range of interaction permitted in the model.

By default, the two-standard-deviation limits are calculated from the exact formula for the asymptotic variance of the residuals under the asymptotic normal approximation, equation (37) of Baddeley et al (2006). However, for compatibility with the original paper of Baddeley et al (2005), if `oldstyle=TRUE`, the two-standard-deviation limits are calculated using the innovation variance, an over-estimate of the true variance of the residuals. (However, see the section about Replicated Data).

The argument `rv` would normally be used only by experts. It enables the user to substitute arbitrary values for the residuals or marks, overriding the usual calculations. If `rv` is present, then instead of calculating the residuals from the fitted model, the algorithm takes the residuals from the object `rv`, and plots them in the manner appropriate to the type of residual or mark selected by `type`. If `type ="eem"` then `rv` should be similar to the return value of `eem`, namely, a numeric vector of length equal to the number of points in the original data point pattern. Otherwise, `rv` should be similar to the return value of `residuals.ppm`, that is, it should be an object of class `"msr"` (see `msr`) representing a signed measure.

The return value of `diagnose.ppm` is an object of class `"diagppm"`. The `plot` method for this class is documented here. There is also a `print` method. See the Examples.

In `plot.diagppm`, if a four-panel diagnostic plot is produced (the default), then the extra arguments `xlab`, `ylab`, `rlab` determine the text labels for the x and y coordinates and the residuals, respectively. The undocumented arguments `col.neg` and `col.smooth` control the colour maps used in the top left and bottom right panels respectively.

See also the companion functions `qqplot.ppm`, which produces a Q-Q plot of the residuals, and `lurking`, which produces lurking variable plots for any spatial covariate.

Value

An object of class "diagppm" which contains the coordinates needed to reproduce the selected plots. This object can be plotted using plot.diagppm and printed using print.diagppm.

Replicated Data

Note that if object is a model that was obtained by first fitting a model to replicated point pattern data using mppm and then using subfits to extract a model for one of the individual point patterns, then the variance calculations are only implemented for the innovation variance (oldstyle=TRUE) and this is the default in such cases.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

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.

Baddeley, A., Møller, J. and Pakes, A.G. (2008) Properties of residuals for spatial point processes. *Annals of the Institute of Statistical Mathematics* **60**, 627–649.

Stoyan, D. and Grabarnik, P. (1991) Second-order characteristics for stochastic structures connected with Gibbs point processes. *Mathematische Nachrichten*, 151:95–100.

See Also

residuals.ppm, eem, ppm.object, qqplot.ppm, lurking, ppm

Examples

```
fit <- ppm(cells ~x, Strauss(r=0.15))
diagnose.ppm(fit)

diagnose.ppm(fit, type="pearson")


diagnose.ppm(fit, which="marks")

diagnose.ppm(fit, type="raw", plot.neg="discrete")

diagnose.ppm(fit, type="pearson", which="smooth")

# save the diagnostics and plot them later
u <- diagnose.ppm(fit, rbord=0.15, plot.it=FALSE)
if(interactive()) {
plot(u)
plot(u, which="marks")
}
```

DiggleGatesStibbard *Diggle-Gates-Stibbard Point Process Model*

Description

Creates an instance of the Diggle-Gates-Stibbard point process model which can then be fitted to point pattern data.

Usage

```
DiggleGatesStibbard(rho)
```

Arguments

rho	Interaction range

Details

Diggle, Gates and Stibbard (1987) proposed a pairwise interaction point process in which each pair of points separated by a distance d contributes a factor $e(d)$ to the probability density, where

$$e(d) = \sin^2\left(\frac{\pi d}{2\rho}\right)$$

for $d < \rho$, and $e(d)$ is equal to 1 for $d \geq \rho$.

The function ppm(), which fits point process models to point pattern data, requires an argument of class "interact" describing the interpoint interaction structure of the model to be fitted. The appropriate description of the Diggle-Gates-Stibbard pairwise interaction is yielded by the function DiggleGatesStibbard(). See the examples below.

Note that this model does not have any regular parameters (as explained in the section on Interaction Parameters in the help file for ppm). The parameter ρ is not estimated by ppm.

Value

An object of class "interact" describing the interpoint interaction structure of the Diggle-Gates-Stibbard process with interaction range rho.

Author(s)

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

References

Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.

Ripley, B.D. (1981) *Spatial statistics*. John Wiley and Sons.

Diggle, P.J., Gates, D.J., and Stibbard, A. (1987) A nonparametric estimator for pairwise-interaction point processes. Biometrika **74**, 763 – 770. *Scandinavian Journal of Statistics* **21**, 359–373.

See Also

ppm, pairwise.family, DiggleGratton, rDGS, ppm.object

Examples

```
DiggleGatesStibbard(0.02)
# prints a sensible description of itself

ppm(cells ~1, DiggleGatesStibbard(0.05))
# fit the stationary D-G-S process to `cells'

# ppm(cells ~ polynom(x,y,3), DiggleGatesStibbard(0.05))
# fit a nonstationary D-G-S process
# with log-cubic polynomial trend
```

DiggleGratton *Diggle-Gratton model*

Description

Creates an instance of the Diggle-Gratton pairwise interaction point process model, which can then be fitted to point pattern data.

Usage

```
DiggleGratton(delta=NA, rho)
```

Arguments

delta	lower threshold δ
rho	upper threshold ρ

Details

Diggle and Gratton (1984, pages 208-210) introduced the pairwise interaction point process with pair potential $h(t)$ of the form

$$h(t) = \left(\frac{t-\delta}{\rho-\delta}\right)^{\kappa} \qquad \text{if } \delta \le t \le \rho$$

with $h(t) = 0$ for $t < \delta$ and $h(t) = 1$ for $t > \rho$. Here δ, ρ and κ are parameters.

Note that we use the symbol κ where Diggle and Gratton (1984) and Diggle, Gates and Stibbard (1987) use β, since in **spatstat** we reserve the symbol β for an intensity parameter.

The parameters must all be nonnegative, and must satisfy $\delta \le \rho$.

The potential is inhibitory, i.e.\ this model is only appropriate for regular point patterns. The strength of inhibition increases with κ. For $\kappa = 0$ the model is a hard core process with hard core radius δ. For $\kappa = \infty$ the model is a hard core process with hard core radius ρ.

The irregular parameters δ, ρ must be given in the call to DiggleGratton, while the regular parameter κ will be estimated.

If the lower threshold delta is missing or NA, it will be estimated from the data when ppm is called. The estimated value of delta is the minimum nearest neighbour distance multiplied by $n/(n+1)$, where n is the number of data points.

Value

An object of class "interact" describing the interpoint interaction structure of a point process.

Author(s)

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

References

Diggle, P.J., Gates, D.J. and Stibbard, A. (1987) A nonparametric estimator for pairwise-interaction point processes. *Biometrika* **74**, 763 – 770.

Diggle, P.J. and Gratton, R.J. (1984) Monte Carlo methods of inference for implicit statistical models. *Journal of the Royal Statistical Society, series B* **46**, 193 – 212.

See Also

ppm, ppm.object, Pairwise

Examples

```
ppm(cells ~1, DiggleGratton(0.05, 0.1))
```

---

dim.detpointprocfamily

Dimension of Determinantal Point Process Model

Description

Extracts the dimension of a determinantal point process model.

Usage

```
  ## S3 method for class 'detpointprocfamily'
dim(x)
```

Arguments

x	object of class "detpointprocfamily".

Value

A numeric (or NULL if the dimension of the model is unspecified).

Author(s)

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

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

`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 $\lambda_k / \lambda_{k-1}$ 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 <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

setcov, as.mask.

Examples

```
# The unit disc
B <- disc()
plot(distcdf(B))
```

---

dkernel *Kernel distributions and random generation*

Description

Density, distribution function, quantile function and random generation for several distributions used in kernel estimation for numerical data.

Usage

```
dkernel(x, kernel = "gaussian", mean = 0, sd = 1)
pkernel(q, kernel = "gaussian", mean = 0, sd = 1, lower.tail = TRUE)
qkernel(p, kernel = "gaussian", mean = 0, sd = 1, lower.tail = TRUE)
rkernel(n, kernel = "gaussian", mean = 0, sd = 1)
```

Arguments

x, q	Vector of quantiles.
p	Vector of probabilities.
kernel	String name of the kernel. Options are "gaussian", "rectangular", "triangular", "epanechnikov", "biweight", "cosine" and "optcosine". (Partial matching is used).
n	Number of observations.
mean	Mean of distribution.
sd	Standard deviation of distribution.
lower.tail	logical; if TRUE (the default), then probabilities are $P(X \leq x)$, otherwise, $P(X > x)$.

Details

These functions give the probability density, cumulative distribution function, quantile function and random generation for several distributions used in kernel estimation for one-dimensional (numerical) data.

The available kernels are those used in density.default, namely "gaussian", "rectangular", "triangular", "epanechnikov", "biweight", "cosine" and "optcosine". For more information about these kernels, see density.default.

dkernel gives the probability density, pkernel gives the cumulative distribution function, qkernel gives the quantile function, and rkernel generates random deviates.

Value

A numeric vector. For dkernel, a vector of the same length as x containing the corresponding values of the probability density. For pkernel, a vector of the same length as x containing the corresponding values of the cumulative distribution function. For qkernel, a vector of the same length as p containing the corresponding quantiles. For rkernel, a vector of length n containing randomly generated values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> <adrian@maths.uwa.edu.au>

and Martin Hazelton

See Also

density.default, kernel.factor

Examples

```
x <- seq(-3,3,length=100)
plot(x, dkernel(x, "epa"), type="l",
         main=c("Epanechnikov kernel", "probability density"))
plot(x, pkernel(x, "opt"), type="l",
         main=c("OptCosine kernel", "cumulative distribution function"))
p <- seq(0,1, length=256)
plot(p, qkernel(p, "biw"), type="l",
         main=c("Biweight kernel", "cumulative distribution function"))
y <- rkernel(100, "tri")
hist(y, main="Random variates from triangular density")
rug(y)
```

---

domain.ppm *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 'ppm'
domain(X, ..., from=c("points", "covariates"))

 ## S3 method for class 'kppm'
domain(X, ..., from=c("points", "covariates"))

 ## S3 method for class 'dppm'
domain(X, ..., from=c("points", "covariates"))

 ## S3 method for class 'slrm'
domain(X, ..., from=c("points", "covariates"))

 ## S3 method for class 'msr'
domain(X, ...)

 ## 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.
from	Character string. See Details.

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.

The argument from applies when X is a fitted point process model (object of class "ppm", "kppm" or "dppm"). If from="data" (the default), domain extracts the window of the original point pattern data to which the model was fitted. If from="covariates" then domain returns the window in which the spatial covariates of the model were provided.

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 <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

domain, domain.rmhmodel, domain.lpp. Window, Frame.

Examples

```
domain(ppm(redwood ~ 1))
domain(quadrat.test(redwood, 2, 2))
```

---

dppapproxkernel *Approximate Determinantal Point Process Kernel*

Description

Returns an approximation to the kernel of a determinantal point process, as a function of one argument x.

Usage

```
dppapproxkernel(model, trunc = 0.99, W = NULL)
```

Arguments

model	Object of class "detpointprocfamily".
trunc	Numeric specifying how the model truncation is performed. See Details section of simulate.detpointprocfamily.
W	Optional window – undocumented at the moment.

Value

A function

Author(s)

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

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

---

dppapproxpcf — *Approximate Pair Correlation Function of Determinantal Point Process Model*

Description

Returns an approximation to the theoretical pair correlation function of a determinantal point process model, as a function of one argument x.

Usage

```
dppapproxpcf(model, trunc = 0.99, W = NULL)
```

Arguments

model	Object of class "detpointprocfamily".
trunc	Numeric value specifying how the model truncation is performed. See Details section of simulate.detpointprocfamily.
W	Optional window – undocumented at the moment.

Details

This function is usually NOT needed for anything. It only exists for investigative purposes.

Value

A function in the R language, with one numeric argument x, that returns the value of the approximate pair correlation at distances x.

Author(s)

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

Examples

```
f <- dppapproxpcf(dppMatern(lambda = 100, alpha=.028, nu=1, d=2))
plot(f, xlim = c(0,0.1))
```

---

dppBessel *Bessel Type Determinantal Point Process Model*

Description

Function generating an instance of the Bessel-type determinantal point process model.

Usage

```
dppBessel(...)
```

Arguments

...	arguments of the form tag=value specifying the model parameters. See Details.

Details

The possible parameters are:

- the intensity lambda as a positive numeric
- the scale parameter alpha as a positive numeric
- the shape parameter sigma as a non-negative numeric
- the dimension d as a positive integer

Value

An object of class "detpointprocfamily".

Author(s)

Frederic Lavancier and Christophe Biscio. Modified by Ege Rubak <rubak@math.aau.dk> , Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

dppCauchy, dppGauss, dppMatern, dppPowerExp

Examples

```
m <- dppBessel(lambda=100, alpha=.05, sigma=0, d=2)
```

---

dppCauchy *Generalized Cauchy Determinantal Point Process Model*

Description

Function generating an instance of the (generalized) Cauchy determinantal point process model.

Usage

```
dppCauchy(...)
```

Arguments

...	arguments of the form tag=value specifying the parameters. See Details.

Details

The (generalized) Cauchy DPP is defined in (Lavancier, Møller and Rubak, 2015) The possible parameters are:

- the intensity lambda as a positive numeric
- the scale parameter alpha as a positive numeric
- the shape parameter nu as a positive numeric (artificially required to be less than 20 in the code for numerical stability)
- the dimension d as a positive integer

Value

An object of class "detpointprocfamily".

Author(s)

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

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Lavancier, F. Møller, J. and Rubak, E. (2015) Determinantal point process models and statistical inference *Journal of the Royal Statistical Society, Series B* **77**, 853–977.

See Also

dppBessel, dppGauss, dppMatern, dppPowerExp

Examples

```
m <- dppCauchy(lambda=100, alpha=.05, nu=1, d=2)
```

---

dppeigen *Internal function calculating eig and index*

Description

This function is mainly for internal package use and is usually not called by the user.

Usage

```
dppeigen(model, trunc, Wscale, stationary = FALSE)
```

Arguments

model	object of class "detpointprocfamily"
trunc	numeric giving the truncation
Wscale	numeric giving the scale of the window relative to a unit box
stationary	logical indicating whether the stationarity of the model should be used (only works in dimension 2).

Value

A list

Author(s)

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

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

dppGauss *Gaussian Determinantal Point Process Model*

Description

Function generating an instance of the Gaussian determinantal point process model.

Usage

```
dppGauss(...)
```

Arguments

...	arguments of the form `tag=value` specifying the parameters. See Details.

Details

The Gaussian DPP is defined in (Lavancier, Møller and Rubak, 2015) The possible parameters are:

- the intensity `lambda` as a positive numeric
- the scale parameter `alpha` as a positive numeric
- the dimension `d` as a positive integer

Value

An object of class `"detpointprocfamily"`.

Author(s)

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

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Lavancier, F. Møller, J. and Rubak, E. (2015) Determinantal point process models and statistical inference *Journal of the Royal Statistical Society, Series B* **77**, 853–977.

See Also

`dppBessel`, `dppCauchy`, `dppMatern`, `dppPowerExp`

Examples

```
m <- dppGauss(lambda=100, alpha=.05, d=2)
```

dppkernel *Extract Kernel from Determinantal Point Process Model Object*

Description

Returns the kernel of a determinantal point process model as a function of one argument `x`.

Usage

```
dppkernel(model, ...)
```

Arguments

`model`	Model of class `"detpointprocfamily"`.
`...`	Arguments passed to `dppapproxkernel` if the exact kernel is unknown

Value

A function

Author(s)

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

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

Examples

```
kernelMatern <- dppkernel(dppMatern(lambda = 100, alpha=.01, nu=1, d=2))
plot(kernelMatern, xlim = c(0,0.1))
```

dppm *Fit Determinantal Point Process Model*

Description

Fit a determinantal point process model to a point pattern.

Usage

```
dppm(formula, family, data=NULL,
     ...,
     startpar = NULL,
     method = c("mincon", "clik2", "palm", "adapcl"),
     weightfun = NULL,
     control =  list(),
     algorithm,
     statistic = "K",
     statargs = list(),
     rmax = NULL,
     epsilon = 0.01,
     covfunargs = NULL,
     use.gam = FALSE,
     nd = NULL, eps = NULL)
```

Arguments

formula	A formula in the R language specifying the data (on the left side) and the form of the model to be fitted (on the right side). For a stationary model it suffices to provide a point pattern without a formula. See Details.
family	Information specifying the family of point processes to be used in the model. Typically one of the family functions dppGauss, dppMatern, dppCauchy, dppBessel or dppPowerExp. Alternatively a character string giving the name of a family function, or the result of calling one of the family functions. See Details.
data	The values of spatial covariates (other than the Cartesian coordinates) required by the model. A named list of pixel images, functions, windows, tessellations or numeric constants.
...	Additional arguments. See Details.
startpar	Named vector of starting parameter values for the optimization.
method	The fitting method. Either "mincon" for minimum contrast, "clik2" for second order composite likelihood, "adapcl" for adaptive second order composite likelihood, or "palm" for Palm likelihood. Partially matched.
weightfun	Optional weighting function w in the composite likelihoods or Palm likelihood. A function in the R language. See Details.
control	List of control parameters passed to the optimization function optim.
algorithm	Character string determining the mathematical algorithm to be used to solve the fitting problem. If method="mincon", "clik2" or "palm" this argument is passed to the generic optimization function optim (renamed as the argument method) with default "Nelder-Mead". If method="adapcl") the argument is passed to the equation solver nleqslv, with default "Bryden".
statistic	Name of the summary statistic to be used for minimum contrast estimation: either "K" or "pcf".
statargs	Optional list of arguments to be used when calculating the statistic. See Details.

rmax	Maximum value of interpoint distance to use in the composite likelihood.
epsilon	Tuning parameter for the adaptive composite likelihood.
covfunargs,use.gam,nd,eps	Arguments passed to ppm when fitting the intensity.

Details

This function fits a determinantal point process model to a point pattern dataset as described in Lavancier et al. (2015).

The model to be fitted is specified by the arguments formula and family.

The argument formula should normally be a formula in the R language. The left hand side of the formula specifies the point pattern dataset to which the model should be fitted. This should be a single argument which may be a point pattern (object of class "ppp") or a quadrature scheme (object of class "quad"). The right hand side of the formula is called the trend and specifies the form of the *logarithm of the intensity* of the process. Alternatively the argument formula may be a point pattern or quadrature scheme, and the trend formula is taken to be ~1.

The argument family specifies the family of point processes to be used in the model. It is typically one of the family functions dppGauss, dppMatern, dppCauchy, dppBessel or dppPowerExp. Alternatively it may be a character string giving the name of a family function, or the result of calling one of the family functions. A family function belongs to class "detpointprocfamilyfun". The result of calling a family function is a point process family, which belongs to class "detpointprocfamily".

The algorithm first estimates the intensity function of the point process using ppm. If the trend formula is ~1 (the default if a point pattern or quadrature scheme is given rather than a "formula") then the model is *homogeneous*. The algorithm begins by estimating the intensity as the number of points divided by the area of the window. Otherwise, the model is *inhomogeneous*. The algorithm begins by fitting a Poisson process with log intensity of the form specified by the formula trend. (See ppm for further explanation).

The interaction parameters of the model are then fitted either by minimum contrast estimation, or by a composite likelihood method (maximum composite likelihood, maximum Palm likelihood, or by solving the adaptive composite likelihood estimating equation).

Minimum contrast: If method = "mincon" (the default) interaction parameters of the model will be fitted by minimum contrast estimation, that is, by matching the theoretical K-function of the model to the empirical K-function of the data, as explained in mincontrast.

For a homogeneous model (trend = ~1) the empirical K-function of the data is computed using Kest, and the interaction parameters of the model are estimated by the method of minimum contrast.

For an inhomogeneous model, the inhomogeneous K function is estimated by Kinhom using the fitted intensity. Then the interaction parameters of the model are estimated by the method of minimum contrast using the inhomogeneous K function. This two-step estimation procedure is heavily inspired by Waagepetersen (2007).

If statistic="pcf" then instead of using the K-function, the algorithm will use the pair correlation function pcf for homogeneous models and the inhomogeneous pair correlation function pcfinhom for inhomogeneous models. In this case, the smoothing parameters of the pair correlation can be controlled using the argument statargs, as shown in the Examples.

Additional arguments ... will be passed to clusterfit to control the minimum contrast fitting algorithm.

Composite likelihood: If `method = "clik2"` the interaction parameters of the model will be fitted by maximising the second-order composite likelihood (Guan, 2006). The log composite likelihood is

$$\sum_{i,j} w(d_{ij}) \log \rho(d_{ij};\theta) - \left(\sum_{i,j} w(d_{ij}) \right) \log \int_D \int_D w(\|u-v\|)\rho(\|u-v\|;\theta)\, du\, dv$$

where the sums are taken over all pairs of data points x_i, x_j separated by a distance $d_{ij} = \|x_i - x_j\|$ less than `rmax`, and the double integral is taken over all pairs of locations u, v in the spatial window of the data. Here $\rho(d;\theta)$ is the pair correlation function of the model with interaction parameters θ.

The function w in the composite likelihood is a weighting function and may be chosen arbitrarily. It is specified by the argument `weightfun`. If this is missing or `NULL` then the default is a threshold weight function, $w(d) = 1(d \le R)$, where R is `rmax/2`.

Palm likelihood: If `method = "palm"` the interaction parameters of the model will be fitted by maximising the Palm loglikelihood (Tanaka et al, 2008)

$$\sum_{i,j} w(x_i, x_j) \log \lambda_P(x_j \mid x_i;\theta) - \int_D w(x_i,u)\lambda_P(u \mid x_i;\theta) \mathrm{d}u$$

with the same notation as above. Here $\lambda_P(u|v;\theta$ is the Palm intensity of the model at location u given there is a point at v.

Adaptive Composite likelihood: If `method = "cladap"` the clustering parameters of the model will be fitted by solving the adaptive second order composite likelihood estimating equation (Lavancier et al, 2021). The estimating function is

$$\sum_{u,v} w(\epsilon \frac{|g(0;\theta)-1|}{g(\|u-v\|;\theta)-1}) \frac{\nabla_\theta g(\|u-v\|;\theta)}{g(\|u-v\|;\theta)} - \int_D \int_D w(\epsilon \frac{M(u,v;\theta)}{\nabla_\theta} g(\|u-v\|;\theta)\rho(u)\rho(v)\, du\, dv$$

where the sum is taken over all distinct pairs of points. Here $g(d;\theta)$ is the pair correlation function with parameters θ. The partial derivative with respect to θ is $g'(d;\theta)$, and $\rho(u)$ denotes the fitted intensity function of the model.

The tuning parameter ϵ is independent of the data. It can be specified by the argument `epsilon` and has default value 0.01.

The function w in the estimating function is a weighting function of bounded support $[-1, 1]$. It is specified by the argument `weightfun`. If this is missing or `NULL` then the default is $w(d) = 1(\|d\| \le 1)\exp(1/(r^2-1))$. The estimating equation is solved using the nonlinear equation solver `nleqslv` from the package **nleqslv**. The package **nleqslv** must be installed in order to use this option.

It is also possible to fix any parameters desired before the optimisation by specifying them as `name=value` in the call to the family function. See Examples.

Value

An object of class `"dppm"` representing the fitted model. There are methods for printing, plotting, predicting and simulating objects of this class.

Optimization algorithm

The following details allow greater control over the fitting procedure.

For the first three fitting methods (`method="mincon"`, `"clik2"` and `"palm"`), the optimisation is performed by the generic optimisation algorithm `optim`. The behaviour of this algorithm can be modified using the arguments `control` and `algorithm`. Useful control arguments include `trace`, `maxit` and `abstol` (documented in the help for `optim`).

For `method="adapcl"`, the estimating equation is solved using the nonlinear equation solver `nleqslv` from the package **nleqslv**. Arguments available for controlling the solver are documented in the help for `nleqslv`; they include `control`, `globStrat`, `startparm` for the initial estimates and `algorithm` for the method. The package **nleqslv** must be installed in order to use this option.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>. Adaptive composite likelihood method contributed by Chiara Fend and modified by Adrian Baddeley.

References

Guan, Y. (2006) A composite likelihood approach in fitting spatial point process models. *Journal of the American Statistical Association* **101**, 1502–1512.

Lavancier, F., Møller, J. and Rubak, E. (2015) Determinantal point process models and statistical inference. *Journal of the Royal Statistical Society, Series B* **77**, 853–977.

Lavancier, F., Poinas, A., and Waagepetersen, R. (2021) Adaptive estimating function inference for nonstationary determinantal point processes. *Scandinavian Journal of Statistics*, **48** (1), 87–107.

Tanaka, U., Ogata, Y. and Stoyan, D. (2008) Parameter estimation and model selection for Neyman-Scott point processes. *Biometrical Journal* **50**, 43–57.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

methods for dppm objects: `plot.dppm`, `fitted.dppm`, `predict.dppm`, `simulate.dppm`, `methods.dppm`, `as.ppm.dppm`, `Kmodel.dppm`, `pcfmodel.dppm`.

Minimum contrast fitting algorithm: higher level interface `clusterfit`; low-level algorithm `mincontrast`.

Deterimantal point process models: `dppGauss`, `dppMatern`, `dppCauchy`, `dppBessel`, `dppPowerExp`,

Summary statistics: `Kest`, `Kinhom`, `pcf`, `pcfinhom`.

See also `ppm`

Examples

```
jpines <- residualspaper$Fig1


dppm(jpines ~ 1, dppGauss)
```

```
dppm(jpines ~ 1, dppGauss, method="c")
dppm(jpines ~ 1, dppGauss, method="p")
dppm(jpines ~ 1, dppGauss, method="a")

if(interactive()) {
   # Fixing the intensity at lambda=2 rather than the Poisson MLE 2.04:
   dppm(jpines ~ 1, dppGauss(lambda=2))

   # The following is quite slow (using K-function)
   dppm(jpines ~ x, dppMatern)
}

 # much faster using pair correlation function
dppm(jpines ~ x, dppMatern, statistic="pcf", statargs=list(stoyan=0.2))

# Fixing the Matern shape parameter at nu=2 rather than estimating it:
dppm(jpines ~ x, dppMatern(nu=2))
```

`dppMatern` *Whittle-Matern Determinantal Point Process Model*

Description

Function generating an instance of the Whittle-Matérn determinantal point process model

Usage

```
dppMatern(...)
```

Arguments

`...`	arguments of the form `tag=value` specifying the parameters. See Details.

Details

The Whittle-Matérn DPP is defined in (Lavancier, Møller and Rubak, 2015) The possible parameters are:

- the intensity `lambda` as a positive numeric
- the scale parameter `alpha` as a positive numeric
- the shape parameter `nu` as a positive numeric (artificially required to be less than 20 in the code for numerical stability)
- the dimension `d` as a positive integer

Value

An object of class `"detpointprocfamily"`.

Author(s)

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

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Lavancier, F. Møller, J. and Rubak, E. (2015) Determinantal point process models and statistical inference *Journal of the Royal Statistical Society, Series B* **77**, 853–977.

See Also

dppBessel, dppCauchy, dppGauss, dppPowerExp

Examples

```
m <- dppMatern(lambda=100, alpha=.02, nu=1, d=2)
```

---

dppparbounds *Parameter Bound for a Determinantal Point Process Model*

Description

Returns the lower and upper bound for a specific parameter of a determinantal point process model when all other parameters are fixed.

Usage

```
dppparbounds(model, name, ...)
```

Arguments

model	Model of class "detpointprocfamily".
name	name of the parameter for which the bound should be computed.
...	Additional arguments passed to the parbounds function of the given model

Value

A data.frame containing lower and upper bounds.

Author(s)

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

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

Examples

```
model <- dppMatern(lambda=100, alpha=.01, nu=1, d=2)
dppparbounds(model, "lambda")
```

`dppPowerExp` *Power Exponential Spectral Determinantal Point Process Model*

Description

Function generating an instance of the Power Exponential Spectral determinantal point process model.

Usage

```
dppPowerExp(...)
```

Arguments

`...`	arguments of the form `tag=value` specifying the parameters. See Details.

Details

The Power Exponential Spectral DPP is defined in (Lavancier, Møller and Rubak, 2015) The possible parameters are:

- the intensity `lambda` as a positive numeric
- the scale parameter `alpha` as a positive numeric
- the shape parameter `nu` as a positive numeric (artificially required to be less than 20 in the code for numerical stability)
- the dimension `d` as a positive integer

Value

An object of class `"detpointprocfamily"`.

Author(s)

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

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Lavancier, F. Møller, J. and Rubak, E. (2015) Determinantal point process models and statistical inference *Journal of the Royal Statistical Society, Series B* **77**, 853–977.

See Also

dppBessel, dppCauchy, dppGauss, dppMatern

Examples

```
m <- dppPowerExp(lambda=100, alpha=.01, nu=1, d=2)
```

---

dppspecden *Extract Spectral Density from Determinantal Point Process Model Object*

Description

Returns the spectral density of a determinantal point process model as a function of one argument x.

Usage

```
dppspecden(model)
```

Arguments

model	Model of class "detpointprocfamily".

Value

A function

Author(s)

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

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

dppspecdenrange

Examples

```
model <- dppMatern(lambda = 100, alpha=.01, nu=1, d=2)
dppspecden(model)
```

dppspecdenrange *Range of Spectral Density of a Determinantal Point Process Model*

Description

Computes the range of the spectral density of a determinantal point process model.

Usage

```
dppspecdenrange(model)
```

Arguments

model	Model of class "detpointprocfamily".

Value

Numeric value (possibly Inf).

Author(s)

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

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

dppspecden

Examples

```
m <- dppBessel(lambda=100, alpha=0.05, sigma=1, d=2)
dppspecdenrange(m)
```

dummify *Convert Data to Numeric Values by Constructing Dummy Variables*

Description

Converts data of any kind to numeric values. A factor is expanded to a set of dummy variables.

Usage

```
dummify(x)
```

Arguments

x	Vector, factor, matrix or data frame to be converted.

Details

This function converts data (such as a factor) to numeric values in order that the user may calculate, for example, the mean, variance, covariance and correlation of the data.

If x is a numeric vector or integer vector, it is returned unchanged.

If x is a logical vector, it is converted to a 0-1 matrix with 2 columns. The first column contains a 1 if the logical value is FALSE, and the second column contains a 1 if the logical value is TRUE.

If x is a complex vector, it is converted to a matrix with 2 columns, containing the real and imaginary parts.

If x is a factor, the result is a matrix of 0-1 dummy variables. The matrix has one column for each possible level of the factor. The (i,j) entry is equal to 1 when the ith factor value equals the jth level, and is equal to 0 otherwise.

If x is a matrix or data frame, the appropriate conversion is applied to each column of x.

Note that, unlike model.matrix, this command converts a factor into a full set of dummy variables (one column for each level of the factor).

Value

A numeric matrix.

Author(s)

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

Examples

```
chara <- sample(letters[1:3], 8, replace=TRUE)
logi <- (runif(8) < 0.3)
comp <- round(4*runif(8) + 3*runif(8) * 1i, 1)
nume <- 8:1 + 0.1
df <- data.frame(nume, chara, logi, comp)
df
dummify(df)
```

---

dummy.ppm *Extract Dummy Points Used to Fit a Point Process Model*

Description

Given a fitted point process model, this function extracts the 'dummy points' of the quadrature scheme used to fit the model.

Usage

```
dummy.ppm(object, drop=FALSE)
```

Arguments

object	fitted point process model (an object of class "ppm").
drop	Logical value determining whether to delete dummy points that were not used to fit the model.

Details

An object of class "ppm" represents a point process model that has been fitted to data. It is typically produced by the model-fitting algorithm ppm.

The maximum pseudolikelihood algorithm in ppm approximates the pseudolikelihood integral by a sum over a finite set of quadrature points, which is constructed by augmenting the original data point pattern by a set of "dummy" points. The fitted model object returned by ppm contains complete information about this quadrature scheme. See ppm or ppm.object for further information.

This function dummy.ppm extracts the dummy points of the quadrature scheme. A typical use of this function would be to count the number of dummy points, to gauge the accuracy of the approximation to the exact pseudolikelihood.

It may happen that some dummy points are not actually used in fitting the model (typically because the value of a covariate is NA at these points). The argument drop specifies whether these unused dummy points shall be deleted (drop=TRUE) or retained (drop=FALSE) in the return value.

See ppm.object for a list of all operations that can be performed on objects of class "ppm".

Value

A point pattern (object of class "ppp").

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppm.object, ppp.object, ppm

Examples

```
data(cells)
fit <- ppm(cells, ~1, Strauss(r=0.1))
X <- dummy.ppm(fit)
npoints(X)
# this is the number of dummy points in the quadrature scheme
```

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

$$e(u, r) = \frac{2\pi r}{\mathrm{length}(c(u, r) \cap W)}$$

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 <r.turner@auckland.ac.nz>

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 <r.turner@auckland.ac.nz>.

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))
```

---

eem *Exponential Energy Marks*

Description

Given a point process model fitted to a point pattern, compute the Stoyan-Grabarnik diagnostic "exponential energy marks" for the data points.

Usage

```
  eem(fit, ...)

  ## S3 method for class 'ppm'
eem(fit, check=TRUE, ...)

  ## S3 method for class 'slrm'
eem(fit, check=TRUE, ...)
```

Arguments

fit	The fitted point process model. An object of class "ppm".
check	Logical value indicating whether to check the internal format of fit. If there is any possibility that this object has been restored from a dump file, or has otherwise lost track of the environment where it was originally computed, set check=TRUE.
...	Ignored.

Details

Stoyan and Grabarnik (1991) proposed a diagnostic tool for point process models fitted to spatial point pattern data. Each point x_i of the data pattern X is given a 'mark' or 'weight'

$$m_i = \frac{1}{\hat{\lambda}(x_i, X)}$$

where $\hat{\lambda}(x_i, X)$ is the conditional intensity of the fitted model. If the fitted model is correct, then the sum of these marks for all points in a region B has expected value equal to the area of B.

The argument fit must be a fitted point process model (object of class "ppm" or "slrm"). Such objects are produced by the fitting algorithms ppm) and slrm. This fitted model object contains complete information about the original data pattern and the model that was fitted to it.

The value returned by eem is the vector of weights $m[i]$ associated with the points $x[i]$ of the original data pattern. The original data pattern (in corresponding order) can be extracted from fit using response.

The function diagnose.ppm produces a set of sensible diagnostic plots based on these weights.

Value

A vector containing the values of the exponential energy mark for each point in the pattern.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

References

Stoyan, D. and Grabarnik, P. (1991) Second-order characteristics for stochastic structures connected with Gibbs point processes. *Mathematische Nachrichten*, 151:95–100.

See Also

diagnose.ppm, ppm.object, data.ppm, residuals.ppm, ppm

Examples

```
data(cells)
fit <- ppm(cells ~x, Strauss(r=0.15))
ee <- eem(fit)
sum(ee)/area(Window(cells)) # should be about 1 if model is correct
Y <- setmarks(cells, ee)
plot(Y, main="Cells data\n Exponential energy marks")
```

`effectfun` *Compute Fitted Effect of a Spatial Covariate in a Point Process Model*

Description

Compute the trend or intensity of a fitted point process model as a function of one of its covariates.

Usage

```
effectfun(model, covname, ..., se.fit=FALSE, nvalues=256)
```

Arguments

`model`	A fitted point process model (object of class `"ppm"`, `"kppm"`, `"lppm"`, `"dppm"`, `"rppm"` or `"profilepl"`).
`covname`	The name of the covariate. A character string. (Needed only if the model has more than one covariate.)
`...`	The fixed values of other covariates (in the form `name=value`) if required.
`se.fit`	Logical. If `TRUE`, asymptotic standard errors of the estimates will be computed, together with a 95% confidence interval.
`nvalues`	Integer. The number of values of the covariate (if it is numeric) for which the effect function should be evaluated. We recommend at least 256.

Details

The object `model` should be an object of class `"ppm"`, `"kppm"`, `"lppm"`, `"dppm"`, `"rppm"` or `"profilepl"` representing a point process model fitted to point pattern data.

The model's trend formula should involve a spatial covariate named `covname`. This could be `"x"` or `"y"` representing one of the Cartesian coordinates. More commonly the covariate is another, external variable that was supplied when fitting the model.

The command `effectfun` computes the fitted trend of the point process `model` as a function of the covariate named `covname`. The return value can be plotted immediately, giving a plot of the fitted trend against the value of the covariate.

If the model also involves covariates other than `covname`, then these covariates will be held fixed. Values for these other covariates must be provided as arguments to `effectfun` in the form `name=value`.

If `se.fit=TRUE`, the algorithm also calculates the asymptotic standard error of the fitted trend, and a (pointwise) asymptotic 95% confidence interval for the true trend.

This command is just a wrapper for the prediction method `predict.ppm`. For more complicated computations about the fitted intensity, use `predict.ppm`.

Value

A data frame containing a column of values of the covariate and a column of values of the fitted trend. If `se.fit=TRUE`, there are 3 additional columns containing the standard error and the upper and lower limits of a confidence interval.

If the covariate named `covname` is numeric (rather than a factor or logical variable), the return value is also of class `"fv"` so that it can be plotted immediately.

Trend and intensity

For a Poisson point process model, the trend is the same as the intensity of the point process. For a more general Gibbs model, the trend is the first order potential in the model (the first order term in the Gibbs representation). In Poisson or Gibbs models fitted by `ppm`, the trend is the only part of the model that depends on the covariates.

Determinantal point process models with fixed intensity

The function `dppm` which fits a determinantal point process model allows the user to specify the intensity `lambda`. In such cases the effect function is undefined, and `effectfun` stops with an error message.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>.

See Also

`ppm`, `predict.ppm`, `fv.object`

Examples

```
X <- copper$SouthPoints
D <- distfun(copper$SouthLines)
fit <- ppm(X ~ polynom(D, 5))
effectfun(fit)
plot(effectfun(fit, se.fit=TRUE))

fitx <- ppm(X ~ x + polynom(D, 5))
plot(effectfun(fitx, "D", x=20))
```

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 <r.turner@auckland.ac.nz>

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)
```

---

emend *Force Model to be Valid*

Description

Check whether a model is valid, and if not, find the nearest model which is valid.

Usage

```
emend(object, ...)
```

Arguments

`object`	A statistical model, belonging to some class.
`...`	Arguments passed to methods.

Details

The function `emend` is generic, and has methods for several classes of statistical models in the **spatstat** package (mostly point process models). Its purpose is to check whether a given model is valid (for example, that none of the model parameters are `NA`) and, if not, to find the nearest model which is valid.

See the methods for more information.

Value

Another model of the same kind.

Author(s)

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

See Also

`emend.ppm`, `valid`.

emend.ppm *Force Point Process Model to be Valid*

Description

Ensures that a fitted point process model satisfies the integrability conditions for existence of the point process.

Usage

```
project.ppm(object, ..., fatal=FALSE, trace=FALSE)

## S3 method for class 'ppm'
emend(object, ..., fatal=FALSE, trace=FALSE)
```

Arguments

object	Fitted point process model (object of class "ppm").
...	Ignored.
fatal	Logical value indicating whether to generate an error if the model cannot be projected to a valid model.
trace	Logical value indicating whether to print a trace of the decision process.

Details

The functions emend.ppm and project.ppm are identical: emend.ppm is a method for the generic emend, while project.ppm is an older name for the same function.

The purpose of the function is to ensure that a fitted model is valid.

The model-fitting function ppm fits Gibbs point process models to point pattern data. By default, the fitted model returned by ppm may not actually exist as a point process.

First, some of the fitted coefficients of the model may be NA or infinite values. This usually occurs when the data are insufficient to estimate all the parameters. The model is said to be *unidentifiable* or *confounded*.

Second, unlike a regression model, which is well-defined for any finite values of the fitted regression coefficients, a Gibbs point process model is only well-defined if the fitted interaction parameters satisfy some constraints. A famous example is the Strauss process (see Strauss) which exists only when the interaction parameter γ is less than or equal to 1. For values $\gamma > 1$, the probability density is not integrable and the process does not exist (and cannot be simulated).

By default, ppm does not enforce the constraint that a fitted Strauss process (for example) must satisfy $\gamma \leq 1$. This is because a fitted parameter value of $\gamma > 1$ could be useful information for data analysis, as it indicates that the Strauss model is not appropriate, and suggests a clustered model should be fitted.

The function emend.ppm or project.ppm modifies the model object so that the model is valid. It identifies the terms in the model object that are associated with illegal parameter values (i.e.

parameter values which are either NA, infinite, or outside their permitted range). It considers all possible sub-models of object obtained by deleting one or more of these terms. It identifies which of these submodels are valid, and chooses the valid submodel with the largest pseudolikelihood. The result of emend.ppm or project.ppm is the true maximum pseudolikelihood fit to the data.

For large datasets or complex models, the algorithm used in emend.ppm or project.ppm may be time-consuming, because it takes time to compute all the sub-models. A faster, approximate algorithm can be applied by setting spatstat.options(project.fast=TRUE). This produces a valid submodel, which may not be the maximum pseudolikelihood submodel.

Use the function valid.ppm to check whether a fitted model object specifies a well-defined point process.

Use the expression all(is.finite(coef(object))) to determine whether all parameters are identifiable.

Value

Another point process model (object of class "ppm").

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppm, valid.ppm, emend, spatstat.options

Examples

```
fit <- ppm(redwood ~1, Strauss(0.1))
coef(fit)
fit2 <- emend(fit)
coef(fit2)
```

---

emend.slrm *Force Spatial Logistic Regression Model to be Valid*

Description

Ensures that a fitted spatial logistic regression specifies a well-defined model.

Usage

```
## S3 method for class 'slrm'
emend(object, ..., fatal=FALSE, trace=FALSE)
```

Arguments

object	Fitted point process model (object of class "slrm").
...	Ignored.
fatal	Logical value indicating whether to generate an error if the model cannot be projected to a valid model.
trace	Logical value indicating whether to print a trace of the decision process.

Details

emend.slrm is a method for the generic emend,

The purpose of the function is to ensure that a fitted model is valid.

The model-fitting function slrm fits spatial logistic regression models to point pattern data.

In some circumstances, the fitted model returned by slrm may not specify a well-defined model, because some of the fitted coefficients of the model may be NA or infinite values. This usually occurs when the data are insufficient to estimate all the parameters. The model is said to be *unidentifiable* or *confounded*.

The function emend.slrm modifies the model object so that the model is valid. It identifies the terms in the model object that are associated with illegal parameter values (i.e. parameter values which are either NA, infinite, or outside their permitted range). It considers all possible sub-models of object obtained by deleting one or more of these terms. It identifies which of these submodels are valid, and chooses the valid submodel with the largest pseudolikelihood. The result of emend.slrm or project.slrm is the true maximum pseudolikelihood fit to the data.

For large datasets or complex models, the algorithm used in emend.slrm may be time-consuming, because it takes time to compute all the sub-models. A faster, approximate algorithm can be applied by setting spatstat.options(project.fast=TRUE). This produces a valid submodel, which may not be the maximum likelihood submodel.

Use the function valid.slrm to check whether a fitted model object specifies a well-defined model.

Value

Another point process model (object of class "slrm").

Author(s)

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

See Also

slrm, valid.slrm, emend, spatstat.options

Examples

```
fit <- slrm(redwood ~ x + I(x))
coef(fit)
fit2 <- emend(fit)
coef(fit2)
```

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)

  ## S3 method for class 'ppm'
envelope(Y, fun=Kest, nsim=99, nrank=1, ...,
  funargs=list(), funYargs=funargs,
  simulate=NULL, fix.n=FALSE, fix.marks=FALSE,
  verbose=TRUE, clipdata=TRUE,
  start=NULL, control=update(default.rmhcontrol(Y), nrep=nrep), nrep=1e5,
  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)

  ## S3 method for class 'kppm'
envelope(Y, fun=Kest, nsim=99, nrank=1, ...,
  funargs=list(), funYargs=funargs,
  simulate=NULL,
  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)

  ## S3 method for class 'slrm'
envelope(Y, fun=Kest, nsim=99, nrank=1, ...,
  funargs=list(), funYargs=funargs,
  simulate=NULL,
  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.
start,control	Optional. These specify the arguments start and control of rmh, giving complete control over the simulation algorithm. Applicable only when Y is a fitted model of class "ppm".
nrep	Number of iterations in the Metropolis-Hastings simulation algorithm. Applicable only when Y is a fitted model of class "ppm".
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 <r.turner@auckland.ac.nz> 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 K function for simulations from Gibbs model
if(online) {
  fit <- ppm(cells ~1, Strauss(0.05))
  plot(envelope(fit))
  plot(envelope(fit, global=TRUE))
} else {
 fit <- ppm(cells ~1, Strauss(0.05), nd=20)
 E <- envelope(fit, nsim=Nsim, correction="border", nrep=100)
 E <- envelope(fit, nsim=Nsim, correction="border", global=TRUE, nrep=100)
}

# Envelope of K function for simulations from cluster model
fit <- kppm(redwood ~1, "Thomas")
if(online) {
  plot(envelope(fit, Gest))
  plot(envelope(fit, Gest, global=TRUE))
} else {
 E <- envelope(fit, Gest, correction="rs", nsim=Nsim, global=TRUE, nrep=100)
```

```
}

# 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"))))
}

# Envelope of INHOMOGENEOUS K-function with fitted trend

# The following is valid.
# Setting lambda=fit means that the fitted model is re-fitted to
# each simulated pattern to obtain the intensity estimates for Kinhom.
# (lambda=NULL would also be valid)

fit <- kppm(redwood ~1, clusters="MatClust")
if(online) {
   plot(envelope(fit, Kinhom, lambda=fit, nsim=19))
} else {
   envelope(fit, Kinhom, lambda=fit, nsim=Nsim)
}

# 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, savepatterns=TRUE)
# EG <- envelope(X, Gest, simulate=EK)

 if(online) {
   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 <r.turner@auckland.ac.nz>

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 <r.turner@auckland.ac.nz>

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 <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

envelope, alltypes.

Examples

```
Nsim <- if(interactive()) 19 else 3
A <- envelopeArray(finpines, markcrosscorr, nsim=Nsim)
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 <r.turner@auckland.ac.nz>

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 <r.turner@auckland.ac.nz>

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"))
```

`exactMPLEstrauss` *Exact Maximum Pseudolikelihood Estimate for Stationary Strauss Process*

Description

Computes, to very high accuracy, the Maximum Pseudolikelihood Estimates of the parameters of a stationary Strauss point process.

Usage

```
exactMPLEstrauss(X, R, ngrid = 2048, plotit = FALSE, project=TRUE)
```

Arguments

`X`	Data to which the Strauss process will be fitted. A point pattern dataset (object of class `"ppp"`).
`R`	Interaction radius of the Strauss process. A non-negative number.
`ngrid`	Grid size for calculation of integrals. An integer, giving the number of grid points in the x and y directions.
`plotit`	Logical. If `TRUE`, the log pseudolikelihood is plotted on the current device.
`project`	Logical. If `TRUE` (the default), the parameter γ is constrained to lie in the interval $[0, 1]$. If `FALSE`, this constraint is not applied.

Details

This function is intended mainly for technical investigation of algorithm performance. Its practical use is quite limited.

It fits the stationary Strauss point process model to the point pattern dataset `X` by maximum pseudolikelihood (with the border edge correction) using an algorithm with very high accuracy. This algorithm is more accurate than the *default* behaviour of the model-fitting function `ppm` because the discretisation is much finer.

Ripley (1988) and Baddeley and Turner (2000) derived the log pseudolikelihood for the stationary Strauss process, and eliminated the parameter β, obtaining an exact formula for the partial log pseudolikelihood as a function of the interaction parameter γ only. The algorithm evaluates this expression to a high degree of accuracy, using numerical integration on a `ngrid * ngrid` lattice, uses `optim` to maximise the log pseudolikelihood with respect to γ, and finally recovers β.

The result is a vector of length 2, containing the fitted coefficients $\log \beta$ and $\log \gamma$. These values correspond to the entries that would be obtained with `coef(ppm(X, ~1, Strauss(R)))`. The fitted coefficients are typically accurate to within 10^{-6} as shown in Baddeley and Turner (2013).

Note however that (by default) `exactMPLEstrauss` constrains the parameter γ to lie in the interval $[0, 1]$ in which the point process is well defined (Kelly and Ripley, 1976) whereas `ppm` does not constrain the value of γ (by default). This behaviour is controlled by the argument `project` to `ppm` and `exactMPLEstrauss`. The default for `ppm` is `project=FALSE`, while the default for `exactMPLEstrauss` is `project=TRUE`.

Value

Vector of length 2.

Author(s)

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

and Rolf Turner `<r.turner@auckland.ac.nz>`

References

Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.

Baddeley, A. and Turner, R. (2013) Bias correction for parameter estimates of spatial point process models. *Journal of Statistical Computation and Simulation* **2012**. `DOI: 10.1080/00949655.2012.755976`

Kelly, F.P. and Ripley, B.D. (1976) On Strauss's model for clustering. *Biometrika* **63**, 357–360.

Ripley, B.D. (1988) *Statistical inference for spatial processes*. Cambridge University Press.

See Also

`ppm`

Examples

```
if(interactive()) {
  rc <- 0.09
  exactMPLEstrauss(cells, rc, plotit=TRUE)
  coef(ppm(cells ~1, Strauss(rc)))
  coef(ppm(cells ~1, Strauss(rc), nd=128))
  rr <- 0.04
  exactMPLEstrauss(redwood, rr)
  exactMPLEstrauss(redwood, rr, project=FALSE)
  coef(ppm(redwood ~1, Strauss(rr)))
} else {
  rc <- 0.09
  exactMPLEstrauss(cells, rc, ngrid=64, plotit=TRUE)
  exactMPLEstrauss(cells, rc, ngrid=64, project=FALSE)
}
```

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 <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>

See Also

fasp.object

Examples

```
# Lansing woods data - multitype points with 6 types
a <- alltypes(lansing, 'K')

# extract first three marks only
b <- a[1:3,1:3]
if(interactive()) {plot(b)}
# subset of array pertaining to hickories
h <- a["hickory", ]
if(interactive()) {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 <r.turner@auckland.ac.nz>

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
```

`Extract.influence.ppm` *Extract Subset of Influence Object*

Description

Extract a subset of an influence object, or extract the influence values at specified locations.

Usage

```
  ## S3 method for class 'influence.ppm'
x[i, ...]
```

Arguments

`x`	A influence object (of class `"influence.ppm"`) computed by `influence.ppm`.
`i`	Subset index (passed to `[.ppp`). Either a spatial window (object of class `"owin"`) or an integer index.
`...`	Ignored.

Details

An object of class `"influence.ppm"` contains the values of the likelihood influence for a point process model, computed by `influence.ppm`. This is effectively a marked point pattern obtained by marking each of the original data points with its likelihood influence.

This function extracts a designated subset of the influence values, either as another influence object, or as a vector of numeric values.

The function `[.influence.ppm` is a method for `[` for the class `"influence.ppm"`. The argument `i` should be an index applicable to a point pattern. It may be either a spatial window (object of class `"owin"`) or a sequence index. The result will be another influence object (of class `influence.ppm`).

To extract the influence values as a numeric vector, use `marks(as.ppp(x))`.

Value

Another object of class `"influence.ppm"`.

Author(s)

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

See Also

`influence.ppm`.

Examples

```
fit <- ppm(cells, ~x)
infl <- influence(fit)
b <- owin(c(0.1, 0.3), c(0.2, 0.4))
infl[b]
infl[1:5]
marks(as.ppp(infl))[1:3]
```

`Extract.leverage.ppm` *Extract Subset of Leverage Object*

Description

Extract a subset of a leverage map, or extract the leverage values at specified locations.

Usage

```
  ## S3 method for class 'leverage.ppm'
x[i, ..., update=TRUE]
```

Arguments

`x`	A leverage object (of class `"leverage.ppm"`) computed by `leverage.ppm`.
`i`	Subset index (passed to `[.im`). Either a spatial window (object of class `"owin"`) or a spatial point pattern (object of class `"ppp"`).
`...`	Further arguments passed to `[.im`, especially the argument `drop`.
`update`	Logical value indicating whether to update the internally-stored value of the mean leverage, by averaging over the specified subset.

Details

An object of class `"leverage.ppm"` contains the values of the leverage function for a point process model, computed by `leverage.ppm`.

This function extracts a designated subset of the leverage values, either as another leverage object, or as a vector of numeric values.

The function `[.leverage.ppm` is a method for `[` for the class `"leverage.ppm"`. The argument `i` should be either

- a spatial window (object of class `"owin"`) determining a region where the leverage map is required. The result will typically be another leverage map (object of class `leverage.ppm`).
- a spatial point pattern (object of class `"ppp"`) specifying locations at which the leverage values are required. The result will be a numeric vector.

The subset operator for images, `[.im`, is applied to the leverage map. If this yields a pixel image, then the result of `[.leverage.ppm` is another leverage object. Otherwise, a vector containing the numeric values of leverage is returned.

Value

Another object of class "leverage.ppm", or a vector of numeric values of leverage.

Author(s)

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

See Also

leverage.ppm.

Examples

```
fit <- ppm(cells ~x)
lev <- leverage(fit)
b <- owin(c(0.1, 0.3), c(0.2, 0.4))
lev[b]
lev[cells]
```

Extract.msr *Extract Subset of Signed or Vector Measure*

Description

Extract a subset of a signed measure or vector-valued measure.

Usage

```
## S3 method for class 'msr'
x[i, j, ...]
```

Arguments

x	A signed or vector measure. An object of class "msr" (see msr).
i	Object defining the subregion or subset to be extracted. Either a spatial window (an object of class "owin"), or a pixel image with logical values, or any type of index that applies to a matrix.
j	Subset index selecting the vector coordinates to be extracted, if x is a vector-valued measure.
...	Ignored.

Details

This operator extracts a subset of the data which determines the signed measure or vector-valued measure x. The result is another measure.

Value

An object of class "msr".

Author(s)

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

See Also

msr

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X ~x+y)
rp <- residuals(fit, type="pearson")
rs <- residuals(fit, type="score")

rp[square(0.5)]
rs[ , 2:3]
```

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

$$F_3(r) = P(d(0, \Phi) \leq r)$$

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 <r.turner@auckland.ac.nz>

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 <r.turner@auckland.ac.nz>

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)))
}
```

`Fiksel` *The Fiksel Interaction*

Description

Creates an instance of Fiksel's double exponential pairwise interaction point process model, which can then be fitted to point pattern data.

Usage

```
Fiksel(r, hc=NA, kappa)
```

Arguments

`r`	The interaction radius of the Fiksel model
`hc`	The hard core distance
`kappa`	The rate parameter

Details

Fiksel (1984) introduced a pairwise interaction point process with the following interaction function c. For two points u and v separated by a distance $d = ||u - v||$, the interaction $c(u, v)$ is equal to 0 if $d < h$, equal to 1 if $d > r$, and equal to

$$\exp(a \exp(-\kappa d))$$

if $h \leq d \leq r$, where h, r, κ, a are parameters.

A graph of this interaction function is shown in the Examples. The interpretation of the parameters is as follows.

- h is the hard core distance: distinct points are not permitted to come closer than a distance h apart.
- r is the interaction range: points further than this distance do not interact.
- κ is the rate or slope parameter, controlling the decay of the interaction as distance increases.
- a is the interaction strength parameter, controlling the strength and type of interaction. If a is zero, the process is Poisson. If `a` is positive, the process is clustered. If `a` is negative, the process is inhibited (regular).

The function ppm(), which fits point process models to point pattern data, requires an argument of class "interact" describing the interpoint interaction structure of the model to be fitted. The appropriate description of the Fiksel pairwise interaction is yielded by the function Fiksel(). See the examples below.

The parameters h, r and κ must be fixed and given in the call to Fiksel, while the canonical parameter a is estimated by ppm().

To estimate h, r andκ it is possible to use profilepl. The maximum likelihood estimator of$h$ is the minimum interpoint distance.

If the hard core distance argument hc is missing or NA, it will be estimated from the data when ppm is called. The estimated value of hc is the minimum nearest neighbour distance multiplied by $n/(n+1)$, where n is the number of data points.

See also Stoyan, Kendall and Mecke (1987) page 161.

Value

An object of class "interact" describing the interpoint interaction structure of the Fiksel process with interaction radius r, hard core distance hc and rate parameter kappa.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.

Fiksel, T. (1984) Estimation of parameterized pair potentials of marked and non-marked Gibbsian point processes. *Electronische Informationsverabeitung und Kybernetika* **20**, 270–278.

Stoyan, D, Kendall, W.S. and Mecke, J. (1987) *Stochastic geometry and its applications*. Wiley.

See Also

ppm, pairwise.family, ppm.object, StraussHard

Examples

```
Fiksel(r=1,hc=0.02, kappa=2)
# prints a sensible description of itself

data(spruces)
X <- unmark(spruces)

fit <- ppm(X ~ 1, Fiksel(r=3.5, kappa=1))
plot(fitin(fit))
```

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

```
plot(Finhom(swedishpines, sigma=10))

# plot(Finhom(swedishpines, sigma=bw.diggle, adjust=2))
```

`fitin.ppm` *Extract the Interaction from a Fitted Point Process Model*

Description

Given a point process model that has been fitted to point pattern data, this function extracts the interpoint interaction part of the model as a separate object.

Usage

```
fitin(object)

## S3 method for class 'ppm'
fitin(object)

## S3 method for class 'profilepl'
fitin(object)
```

Arguments

`object`	A fitted point process model (object of class `"ppm"` or `"profilepl"`).

Details

An object of class `"ppm"` describes a fitted point process model. It contains information about the original data to which the model was fitted, the spatial trend that was fitted, the interpoint interaction that was fitted, and other data. See `ppm.object`) for details of this class.

The function `fitin` extracts from this model the information about the fitted interpoint interaction only. The information is organised as an object of class `"fii"` (fitted interpoint interaction). This object can be printed or plotted.

Users may find this a convenient way to plot the fitted interpoint interaction term, as shown in the Examples.

For a pairwise interaction, the plot of the fitted interaction shows the pair interaction function (the contribution to the probability density from a pair of points as a function of the distance between them). For a higher-order interaction, the plot shows the strongest interaction (the value most different from 1) that could ever arise at the given distance.

The fitted interaction coefficients can also be extracted from this object using `coef`.

Value

An object of class `"fii"` representing the fitted interpoint interaction. This object can be printed and plotted.

Author(s)

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

See Also

Methods for handling fitted interactions: `methods.fii`, `reach.fii`, `as.interact.fii`.

Background: `ppm`, `ppm.object`.

Examples

```
   # unmarked
   model <- ppm(swedishpines ~1, PairPiece(seq(3,19,by=4)))
   f <- fitin(model)
   f
   plot(f)

# extract fitted interaction coefficients
   coef(f)

   # multitype
   # fit the stationary multitype Strauss process to `amacrine'
   r <- 0.02 * matrix(c(1,2,2,1), nrow=2,ncol=2)
   model <- ppm(amacrine ~1, MultiStrauss(r))
   f <- fitin(model)
   f
   plot(f)
```

`fitted.mppm` *Fitted Conditional Intensity for Multiple Point Process Model*

Description

Given a point process model fitted to multiple point patterns, compute the fitted conditional intensity of the model at the points of each data pattern, or at the points of the quadrature schemes used to fit the model.

Usage

```
 ## S3 method for class 'mppm'
fitted(object, ..., type = "lambda", dataonly = FALSE)
```

Arguments

`object`	The fitted model. An object of class `"mppm"` obtained from `mppm`.
`...`	Ignored.
`type`	Type of fitted values: either `"trend"` for the spatial trend, or `"lambda"` or `"cif"` for the conditional intensity.
`dataonly`	If `TRUE`, fitted values are computed only for the points of the data point patterns. If `FALSE`, fitted values are computed for the points of the quadrature schemes used to fit the model.

Details

This function evaluates the conditional intensity $\hat{\lambda}(u, x)$ or spatial trend $b(\hat{u})$ of the fitted point process model for certain locations u, for each of the original point patterns x to which the model was fitted.

The locations u at which the fitted conditional intensity/trend is evaluated, are the points of the quadrature schemes used to fit the model in `mppm`. They include the data points (the points of the original point pattern datasets) and other "dummy" points in the window of observation.

Use `predict.mppm` to compute the fitted conditional intensity at other locations or with other values of the explanatory variables.

Value

A list of vectors (one for each row of the original hyperframe, i.e. one vector for each of the original point patterns) containing the values of the fitted conditional intensity or (if `type="trend"`) the fitted spatial trend.

Entries in these vector correspond to the quadrature points (data or dummy points) used to fit the model. The quadrature points can be extracted from `object` by `quad.mppm(object)`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Ida-Maria Sintorn and Leanne Bischoff. Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

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

See Also

mppm, predict.mppm

Examples

```
model <- mppm(Bugs ~ x, data=hyperframe(Bugs=waterstriders),
                        interaction=Strauss(7))
cifs <- fitted(model)
```

`fitted.ppm` *Fitted Conditional Intensity for Point Process Model*

Description

Given a point process model fitted to a point pattern, compute the fitted conditional intensity or fitted trend of the model at the points of the pattern, or at the points of the quadrature scheme used to fit the model.

Usage

```
  ## S3 method for class 'ppm'
fitted(object, ..., type="lambda",
                        dataonly=FALSE, new.coef=NULL, leaveoneout=FALSE,
       drop=FALSE, check=TRUE, repair=TRUE,
       ignore.hardcore=FALSE, dropcoef=FALSE)
```

Arguments

`object`	The fitted point process model (an object of class `"ppm"`)
`...`	Ignored.
`type`	String (partially matched) indicating whether the fitted value is the conditional intensity (`"lambda"` or `"cif"`) or the first order trend (`"trend"`) or the logarithm of conditional intensity (`"link"`).

dataonly	Logical. If TRUE, then values will only be computed at the points of the data point pattern. If FALSE, then values will be computed at all the points of the quadrature scheme used to fit the model, including the points of the data point pattern.
new.coef	Numeric vector of parameter values to replace the fitted model parameters coef(object).
leaveoneout	Logical. If TRUE the fitted value at each data point will be computed using a leave-one-out method. See Details.
drop	Logical value determining whether to delete quadrature points that were not used to fit the model.
check	Logical value indicating whether to check the internal format of object. If there is any possibility that this object has been restored from a dump file, or has otherwise lost track of the environment where it was originally computed, set check=TRUE.
repair	Logical value indicating whether to repair the internal format of object, if it is found to be damaged.
ignore.hardcore	Advanced use only. Logical value specifying whether to compute only the finite part of the interaction potential (effectively removing any hard core interaction terms).
dropcoef	Internal use only.

Details

The argument object must be a fitted point process model (object of class "ppm"). Such objects are produced by the model-fitting algorithm ppm).

This function evaluates the conditional intensity $\hat{\lambda}(u, x)$ or spatial trend $\hat{b}(u)$ of the fitted point process model for certain locations u, where x is the original point pattern dataset to which the model was fitted.

The locations u at which the fitted conditional intensity/trend is evaluated, are the points of the quadrature scheme used to fit the model in ppm. They include the data points (the points of the original point pattern dataset x) and other "dummy" points in the window of observation.

If leaveoneout=TRUE, fitted values will be computed for the data points only, using a 'leave-one-out' rule: the fitted value at X[i] is effectively computed by deleting this point from the data and re-fitting the model to the reduced pattern X[-i], then predicting the value at X[i]. (Instead of literally performing this calculation, we apply a Taylor approximation using the influence function computed in dfbetas.ppm.

The argument drop is explained in quad.ppm.

Use predict.ppm to compute the fitted conditional intensity at other locations or with other values of the explanatory variables.

Value

A vector containing the values of the fitted conditional intensity, fitted spatial trend, or logarithm of the fitted conditional intensity.

Entries in this vector correspond to the quadrature points (data or dummy points) used to fit the model. The quadrature points can be extracted from object by union.quad(quad.ppm(object)).

Author(s)

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

References

Baddeley, A., Turner, R., Møller, J. and Hazelton, M. (2005). Residual analysis for spatial point processes (with discussion). *Journal of the Royal Statistical Society, Series B* **67**, 617–666.

See Also

ppm.object, ppm, predict.ppm

Examples

```
str <- ppm(cells ~x, Strauss(r=0.1))
lambda <- fitted(str)

# extract quadrature points in corresponding order
quadpoints <- union.quad(quad.ppm(str))

# plot conditional intensity values
# as circles centred on the quadrature points
quadmarked <- setmarks(quadpoints, lambda)
plot(quadmarked)

if(!interactive()) str <- ppm(cells ~ x)

lambdaX <- fitted(str, leaveoneout=TRUE)
```

---

fitted.slrm *Fitted Probabilities for Spatial Logistic Regression*

Description

Given a fitted Spatial Logistic Regression model, this function computes the fitted probabilities for each pixel.

Usage

```
  ## S3 method for class 'slrm'
fitted(object, ...)
```

Arguments

object	a fitted spatial logistic regression model. An object of class "slrm".
...	Ignored.

Details

This is a method for the generic function fitted for spatial logistic regression models (objects of class "slrm", usually obtained from the function slrm).

The algorithm computes the fitted probabilities of the presence of a random point in each pixel.

Value

A pixel image (object of class "im") containing the fitted probability for each pixel.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> <adrian@maths.uwa.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

slrm, fitted

Examples

```
X <- rpoispp(42)
fit <- slrm(X ~ x+y)
plot(fitted(fit))
```

---

fixef.mppm *Extract Fixed Effects from Point Process Model*

Description

Given a point process model fitted to a list of point patterns, extract the fixed effects of the model. A method for fixef.

Usage

```
 ## S3 method for class 'mppm'
fixef(object, ...)
```

Arguments

object	A fitted point process model (an object of class "mppm").
...	Ignored.

Details

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

The argument `object` must be a fitted point process model (object of class `"mppm"`) produced by the fitting algorithm `mppm`). This represents a point process model that has been fitted to a list of several point pattern datasets. See `mppm` for information.

This function extracts the coefficients of the fixed effects of the model.

Value

A numeric vector of coefficients.

Author(s)

Adrian Baddeley, Ida-Maria Sintorn and Leanne Bischoff. Implemented in **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

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

See Also

`coef.mppm`

Examples

```
H <- hyperframe(Y = waterstriders)
# Tweak data to exaggerate differences
H$Y[[1]] <- rthin(H$Y[[1]], 0.3)
m1 <- mppm(Y ~ id,  data=H, Strauss(7))
fixef(m1)
m2 <- mppm(Y ~ 1,  random=~1|id, data=H, Strauss(7))
fixef(m2)
```

`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

```
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].
...	Ignored.
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).

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")
mod <- ppm(X ~ marks * x)
lam <- fitted(mod, dataonly=TRUE)
lmin <- min(predict(mod)[["off"]]) * 0.9
plot(FmultiInhom(X, J, lambda=lam, lambdamin=lmin))
```

`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 <r.turner@auckland.ac.nz>

See Also

fv, plot.fv, formula.

Examples

```
K <- Kest(cells)
formula(K)
formula(K) <- (iso ~ r)
```

---

formula.ppm *Model Formulae for Gibbs Point Process Models*

Description

Extract the trend formula, or the terms in the trend formula, in a fitted Gibbs point process model.

Usage

```
## S3 method for class 'ppm'
formula(x, ...)
## S3 method for class 'ppm'
terms(x, ...)
```

Arguments

x	An object of class "ppm", representing a fitted point process model.
...	Arguments passed to other methods.

Details

These functions are methods for the generic commands formula and terms for the class "ppm".

An object of class "ppm" represents a fitted Poisson or Gibbs point process model. It is obtained from the model-fitting function ppm.

The method formula.ppm extracts the trend formula from the fitted model x (the formula originally specified as the argument trend to ppm). The method terms.ppm extracts the individual terms in the trend formula.

Value

See the help files for the corresponding generic functions.

Author(s)

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

See Also

ppm, as.owin, coef.ppm, extractAIC.ppm, fitted.ppm, logLik.ppm, model.frame.ppm, model.matrix.ppm, plot.ppm, predict.ppm, residuals.ppm, simulate.ppm, summary.ppm, update.ppm, vcov.ppm.

Examples

```
data(cells)
fit <- ppm(cells, ~x)
formula(fit)
terms(fit)
```

---

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 `<r.turner@auckland.ac.nz>`

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 <r.turner@auckland.ac.nz>

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.

Use range.fv to compute the range of y values for a function, and 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 <r.turner@auckland.ac.nz>

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 <r.turner@auckland.ac.nz>

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)
```

Gcom *Model Compensator of Nearest Neighbour Function*

Description

Given a point process model fitted to a point pattern dataset, this function computes the *compensator* of the nearest neighbour distance distribution function G based on the fitted model (as well as the usual nonparametric estimates of G based on the data alone). Comparison between the nonparametric and model-compensated G functions serves as a diagnostic for the model.

Usage

```
Gcom(object, r = NULL, breaks = NULL, ...,
     correction = c("border", "Hanisch"),
     conditional = !is.poisson(object),
     restrict=FALSE,
     model=NULL,
     trend = ~1, interaction = Poisson(),
     rbord = reach(interaction),
     ppmcorrection="border",
     truecoef = NULL, hi.res = NULL)
```

Arguments

`object`	Object to be analysed. Either a fitted point process model (object of class `"ppm"`) or a point pattern (object of class `"ppp"`) or quadrature scheme (object of class `"quad"`).
`r`	Optional. Vector of values of the argument r at which the function $G(r)$ should be computed. This argument is usually not specified. There is a sensible default.
`breaks`	This argument is for internal use only.
`correction`	Edge correction(s) to be employed in calculating the compensator. Options are `"border"`, `"Hanisch"` and `"best"`. Alternatively `correction="all"` selects all options.
`conditional`	Optional. Logical value indicating whether to compute the estimates for the conditional case. See Details.
`restrict`	Logical value indicating whether to compute the restriction estimator (`restrict=TRUE`) or the reweighting estimator (`restrict=FALSE`, the default). Applies only if `conditional=TRUE`. See Details.
`model`	Optional. A fitted point process model (object of class `"ppm"`) to be re-fitted to the data using `update.ppm`, if `object` is a point pattern. Overrides the arguments `trend,interaction,rbord,ppmcorrection`.

`trend,interaction,rbord`
: Optional. Arguments passed to ppm to fit a point process model to the data, if `object` is a point pattern. See ppm for details.

`...`
: Extra arguments passed to ppm.

`ppmcorrection`
: The `correction` argument to ppm.

`truecoef`
: Optional. Numeric vector. If present, this will be treated as if it were the true coefficient vector of the point process model, in calculating the diagnostic. Incompatible with `hi.res`.

`hi.res`
: Optional. List of parameters passed to quadscheme. If this argument is present, the model will be re-fitted at high resolution as specified by these parameters. The coefficients of the resulting fitted model will be taken as the true coefficients. Then the diagnostic will be computed for the default quadrature scheme, but using the high resolution coefficients.

Details

This command provides a diagnostic for the goodness-of-fit of a point process model fitted to a point pattern dataset. It computes different estimates of the nearest neighbour distance distribution function G of the dataset, which should be approximately equal if the model is a good fit to the data.

The first argument, `object`, is usually a fitted point process model (object of class `"ppm"`), obtained from the model-fitting function ppm.

For convenience, `object` can also be a point pattern (object of class `"ppp"`). In that case, a point process model will be fitted to it, by calling ppm using the arguments `trend` (for the first order trend), `interaction` (for the interpoint interaction) and `rbord` (for the erosion distance in the border correction for the pseudolikelihood). See ppm for details of these arguments.

The algorithm first extracts the original point pattern dataset (to which the model was fitted) and computes the standard nonparametric estimates of the G function. It then also computes the *model-compensated* G function. The different functions are returned as columns in a data frame (of class `"fv"`). The interpretation of the columns is as follows (ignoring edge corrections):

`bord`: the nonparametric border-correction estimate of $G(r)$,

$$\hat{G}(r) = \frac{\sum_i I\{d_i \le r\} I\{b_i > r\}}{\sum_i I\{b_i > r\}}$$

where d_i is the distance from the i-th data point to its nearest neighbour, and b_i is the distance from the i-th data point to the boundary of the window W.

`bcom`: the model compensator of the border-correction estimate

$$\mathbf{C}\,\hat{G}(r) = \frac{\int \lambda(u,x) I\{b(u) > r\} I\{d(u,x) \le r\}}{1 + \sum_i I\{b_i > r\}}$$

where $\lambda(u,x)$ denotes the conditional intensity of the model at the location u, and $d(u,x)$ denotes the distance from u to the nearest point in x, while $b(u)$ denotes the distance from u to the boundary of the window$W$.

`han`: the nonparametric Hanisch estimate of $G(r)$

$$\hat{G}(r) = \frac{D(r)}{D(\infty)}$$

where

$$D(r) = \sum_i \frac{I\{x_i \in W_{\ominus d_i}\} I\{d_i \le r\}}{\mathrm{area}(W_{\ominus d_i})}$$

in which $W_{\ominus r}$ denotes the erosion of the window W by a distance r.

`hcom`: the corresponding model-compensated function

$$\mathbf{C}\, G(r) = \int_W \frac{\lambda(u,x) I(u \in W_{\ominus d(u)}) I(d(u) \le r)}{\hat{D}(\infty)\mathrm{area}(W_{\ominus d(u)}) + 1}$$

where $d(u) = d(u,x)$ is the ('empty space') distance from location u to the nearest point of x.

If the fitted model is a Poisson point process, then the formulae above are exactly what is computed. If the fitted model is not Poisson, the formulae above are modified slightly to handle edge effects.

The modification is determined by the arguments `conditional` and `restrict`. The value of `conditional` defaults to `FALSE` for Poisson models and `TRUE` for non-Poisson models. If `conditional=FALSE` then the formulae above are not modified. If `conditional=TRUE`, then the algorithm calculates the *restriction estimator* if `restrict=TRUE`, and calculates the *reweighting estimator* if `restrict=FALSE`. See Appendix E of Baddeley, Rubak and Møller (2011). See also `spatstat.options('eroded.intensity')`. Thus, by default, the reweighting estimator is computed for non-Poisson models.

The border-corrected and Hanisch-corrected estimates of $G(r)$ are approximately unbiased estimates of the G-function, assuming the point process is stationary. The model-compensated functions are unbiased estimates *of the mean value of the corresponding nonparametric estimate*, assuming the model is true. Thus, if the model is a good fit, the mean value of the difference between the nonparametric and model-compensated estimates is approximately zero.

To compute the difference between the nonparametric and model-compensated functions, use `Gres`.

Value

A function value table (object of class `"fv"`), essentially a data frame of function values. There is a plot method for this class. See `fv.object`.

Author(s)

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

References

Baddeley, A., Rubak, E. and Møller, J. (2011) Score, pseudo-score and residual diagnostics for spatial point process models. *Statistical Science* **26**, 613–646.

See Also

Related functions: `Gest`, `Gres`.

Alternative functions: `Kcom`, `psstA`, `psstG`, `psst`.

Model fitting: `ppm`.

Examples

```
    data(cells)
    fit0 <- ppm(cells, ~1) # uniform Poisson
    G0 <- Gcom(fit0)
    G0
    plot(G0)
# uniform Poisson is clearly not correct

# Hanisch estimates only
    plot(Gcom(fit0), cbind(han, hcom) ~ r)

    fit1 <- ppm(cells, ~1, Strauss(0.08))
    plot(Gcom(fit1), cbind(han, hcom) ~ r)

# Try adjusting interaction distance

    fit2 <- update(fit1, Strauss(0.10))
    plot(Gcom(fit2), cbind(han, hcom) ~ r)

    G3 <- Gcom(cells, interaction=Strauss(0.12))
    plot(G3, cbind(han, hcom) ~ r)
```

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 `<r.turner@auckland.ac.nz>`

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
```

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 <r.turner@auckland.ac.nz>

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
```

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 <r.turner@auckland.ac.nz>

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

```
data(cells)
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)))
}
```

`Geyer` *Geyer's Saturation Point Process Model*

Description

Creates an instance of Geyer's saturation point process model which can then be fitted to point pattern data.

Usage

```
Geyer(r,sat)
```

Arguments

`r`	Interaction radius. A positive real number.
`sat`	Saturation threshold. A non-negative real number.

Details

Geyer (1999) introduced the "saturation process", a modification of the Strauss process (see `Strauss`) in which the total contribution to the potential from each point (from its pairwise interaction with all other points) is trimmed to a maximum value s. The interaction structure of this model is implemented in the function `Geyer()`.

The saturation point process with interaction radius r, saturation threshold s, and parameters β and γ, is the point process in which each point x_i in the pattern X contributes a factor

$$\beta\gamma^{\min(s,t(x_i,X))}$$

to the probability density of the point pattern, where $t(x_i, X)$ denotes the number of 'close neighbours' of x_i in the pattern X. A close neighbour of x_i is a point x_j with $j \neq i$ such that the distance between x_i and x_j is less than or equal to r.

If the saturation threshold s is set to infinity, this model reduces to the Strauss process (see `Strauss`) with interaction parameter γ^2. If $s = 0$, the model reduces to the Poisson point process. If s is a finite positive number, then the interaction parameter γ may take any positive value (unlike the case of the Strauss process), with values $\gamma < 1$ describing an 'ordered' or 'inhibitive' pattern, and values $\gamma > 1$ describing a 'clustered' or 'attractive' pattern.

The nonstationary saturation process is similar except that the value β is replaced by a function $\beta(x_i)$ of location.

The function `ppm()`, which fits point process models to point pattern data, requires an argument of class `"interact"` describing the interpoint interaction structure of the model to be fitted. The appropriate description of the saturation process interaction is yielded by `Geyer(r, sat)` where the arguments `r` and `sat` specify the Strauss interaction radius r and the saturation threshold s, respectively. See the examples below.

Note the only arguments are the interaction radius `r` and the saturation threshold `sat`. When `r` and `sat` are fixed, the model becomes an exponential family. The canonical parameters $\log(\beta)$ and $\log(\gamma)$ are estimated by `ppm()`, not fixed in `Geyer()`.

Value

An object of class "interact" describing the interpoint interaction structure of Geyer's saturation point process with interaction radius r and saturation threshold sat.

Zero saturation

The value sat=0 is permitted by Geyer, but this is not very useful. For technical reasons, when ppm fits a Geyer model with sat=0, the default behaviour is to return an "invalid" fitted model in which the estimate of γ is NA. In order to get a Poisson process model returned when sat=0, you would need to set emend=TRUE in the call to ppm.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

References

Geyer, C.J. (1999) Likelihood Inference for Spatial Point Processes. Chapter 3 in O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. Van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*, Chapman and Hall / CRC, Monographs on Statistics and Applied Probability, number 80. Pages 79–140.

See Also

ppm, pairwise.family, ppm.object, Strauss.

To make an interaction object like Geyer but having multiple interaction radii, see BadGey or Hybrid.

Examples

```
ppm(cells, ~1, Geyer(r=0.07, sat=2))
# fit the stationary saturation process to `cells'
```

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:

$$G(r) = P(R \le r)$$

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

$$H(r) = P(S \le r)$$

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

```
data(copper)
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 <r.turner@auckland.ac.nz> 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

```
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).

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")
mod <- ppm(X ~ marks * x)
lam <- fitted(mod, dataonly=TRUE)
lmin <- min(predict(mod)[["off"]]) * 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="")
```

Gres *Residual G Function*

Description

Given a point process model fitted to a point pattern dataset, this function computes the residual G function, which serves as a diagnostic for goodness-of-fit of the model.

Usage

```
Gres(object, ...)
```

Arguments

object	Object to be analysed. Either a fitted point process model (object of class "ppm"), a point pattern (object of class "ppp"), a quadrature scheme (object of class "quad"), or the value returned by a previous call to Gcom.
...	Arguments passed to Gcom.

Details

This command provides a diagnostic for the goodness-of-fit of a point process model fitted to a point pattern dataset. It computes a residual version of the G function of the dataset, which should be approximately zero if the model is a good fit to the data.

In normal use, object is a fitted point process model or a point pattern. Then Gres first calls Gcom to compute both the nonparametric estimate of the G function and its model compensator. Then Gres computes the difference between them, which is the residual G-function.

Alternatively, object may be a function value table (object of class "fv") that was returned by a previous call to Gcom. Then Gres computes the residual from this object.

Value

A function value table (object of class "fv"), essentially a data frame of function values. There is a plot method for this class. See fv.object.

Author(s)

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

References

Baddeley, A., Rubak, E. and Møller, J. (2011) Score, pseudo-score and residual diagnostics for spatial point process models. *Statistical Science* **26**, 613–646.

See Also

Related functions: Gcom, Gest.

Alternative functions: Kres, psstA, psstG, psst.

Model-fitting: ppm.

Examples

```
    data(cells)
    fit0 <- ppm(cells, ~1) # uniform Poisson
    G0 <- Gres(fit0)
    plot(G0)
# Hanisch correction estimate
    plot(G0, hres ~ r)
# uniform Poisson is clearly not correct

    fit1 <- ppm(cells, ~1, Strauss(0.08))
    plot(Gres(fit1), hres ~ r)
# fit looks approximately OK; try adjusting interaction distance

    plot(Gres(cells, interaction=Strauss(0.12)))

# How to make envelopes
    if(interactive()) {
      E <- envelope(fit1, Gres, model=fit1, nsim=39)
      plot(E)
    }
# For computational efficiency
    Gc <- Gcom(fit1)
    G1 <- Gres(Gc)
```

`Hardcore` *The Hard Core Point Process Model*

Description

Creates an instance of the hard core point process model which can then be fitted to point pattern data.

Usage

```
Hardcore(hc=NA)
```

Arguments

`hc`	The hard core distance

Details

A hard core process with hard core distance h and abundance parameter β is a pairwise interaction point process in which distinct points are not allowed to come closer than a distance h apart.

The probability density is zero if any pair of points is closer than h units apart, and otherwise equals

$$f(x_1, \ldots, x_n) = \alpha \beta^{n(x)}$$

where $x_1, \ldots, x_n$ represent the points of the pattern, $n(x)$ is the number of points in the pattern, and α is the normalising constant.

The function `ppm()`, which fits point process models to point pattern data, requires an argument of class `"interact"` describing the interpoint interaction structure of the model to be fitted. The appropriate description of the hard core process pairwise interaction is yielded by the function `Hardcore()`. See the examples below.

If the hard core distance argument `hc` is missing or `NA`, it will be estimated from the data when `ppm` is called. The estimated value of `hc` is the minimum nearest neighbour distance multiplied by $n/(n+1)$, where n is the number of data points.

Value

An object of class `"interact"` describing the interpoint interaction structure of the hard core process with hard core distance `hc`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.

Ripley, B.D. (1981) *Spatial statistics*. John Wiley and Sons.

See Also

Strauss, StraussHard, MultiHard, ppm, pairwise.family, ppm.object

Examples

```
Hardcore(0.02)
# prints a sensible description of itself

ppm(cells ~1, Hardcore(0.05))
# fit the stationary hard core process to `cells'

# estimate hard core radius from data
ppm(cells, ~1, Hardcore())
# ppm(cells ~1, Hardcore)

# ppm(cells ~ polynom(x,y,3), Hardcore(0.05))
# fit a nonstationary hard core process
# with log-cubic polynomial trend
```

---

hardcoredist *Extract the Hard Core Distance of a Point Process Model*

Description

Extract or compute the hard core distance of a point process model.

Usage

```
hardcoredist(x, ...)

## S3 method for class 'fii'
hardcoredist(x, ..., epsilon = 0)

## S3 method for class 'ppm'
hardcoredist(x, ..., epsilon = 0)
```

Arguments

x	An object representing a point process model (class "ppm") or the interaction structure of a point process (class "fii") or similar.
...	Additional arguments passed to methods.
epsilon	Tolerance for defining the hard core.

Details

A point process model has a hard core distance `h` if it is impossible for two random points to lie closer than the distance `h` apart.

The function `hardcoredist` is generic, with methods for objects of class `"ppm"` (point process models) and `"fii"` (fitted point process interactions). It extracts or computes the hard core distance.

If `epsilon` is specified, then the code calculates the largest distance at which the interaction factor is smaller than `epsilon`, implying that points are unlikely to occur closer than this distance.

The result is zero if the model does not have a hard core distance.

Value

A single numeric value, or for multitype point processes, a numeric matrix giving the hard core distances for each pair of types of points.

Author(s)

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

Examples

```
m <- ppm(cells~1, Hardcore())
hardcoredist(m)
```

---

harmonic *Basis for Harmonic Functions*

Description

Evaluates a basis for the harmonic polynomials in x and y of degree less than or equal to n.

Usage

```
harmonic(x, y, n)
```

Arguments

`x`	Vector of x coordinates
`y`	Vector of y coordinates
`n`	Maximum degree of polynomial

Details

This function computes a basis for the harmonic polynomials in two variables x and y up to a given degree n and evaluates them at given x, y locations. It can be used in model formulas (for example in the model-fitting functions `lm`, `glm`, `gam` and `ppm`) to specify a linear predictor which is a harmonic function.

A function $f(x, y)$ is harmonic if

$$\frac{\partial^2}{\partial x^2} f + \frac{\partial^2}{\partial y^2} f = 0.$$

The harmonic polynomials of degree less than or equal to n have a basis consisting of $2n$ functions.

This function was implemented on a suggestion of P. McCullagh for fitting nonstationary spatial trend to point process models.

Value

A data frame with `2 * n` columns giving the values of the basis functions at the coordinates. Each column is labelled by an algebraic expression for the corresponding basis function.

Author(s)

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

See Also

`ppm`, `polynom`

Examples

```
# inhomogeneous point pattern
X <- unmark(longleaf)


# fit Poisson point process with log-cubic intensity
fit.3 <- ppm(X ~ polynom(x,y,3), Poisson())

# fit Poisson process with log-cubic-harmonic intensity
fit.h <- ppm(X ~ harmonic(x,y,3), Poisson())

# Likelihood ratio test
lrts <- 2 * (logLik(fit.3) - logLik(fit.h))
df <- with(coords(X),
           ncol(polynom(x,y,3)) - ncol(harmonic(x,y,3)))
pval <- 1 - pchisq(lrts, df=df)
```

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 <r.turner@auckland.ac.nz>

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
```

---

harmonise.msr *Make Measures Compatible*

Description

Convert several measures to a common quadrature scheme

Usage

```
## S3 method for class 'msr'
harmonise(...)
```

Arguments

...	Any number of measures (objects of class "msr").

Details

This function makes any number of measures compatible, by converting them all to a common quadrature scheme.

The command harmonise is generic. This is the method for objects of class "msr".

Value

A list, of length equal to the number of arguments ..., whose entries are measures.

Author(s)

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

See Also

harmonise, msr

Examples

```
fit1 <- ppm(cells ~ x)
fit2 <- ppm(rpoispp(ex=cells) ~ x)
m1 <- residuals(fit1)
m2 <- residuals(fit2)
harmonise(m1, m2)
s1 <- residuals(fit1, type="score")
s2 <- residuals(fit2, type="score")
harmonise(s1, s2)
```

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

$$H(r) = P(D \le r \mid D > 0)$$

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 $H^*(r) = P(D \le r)$ 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 <r.turner@auckland.ac.nz> 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)
```

HierHard *The Hierarchical Hard Core Point Process Model*

Description

Creates an instance of the hierarchical hard core point process model which can then be fitted to point pattern data.

Usage

```
HierHard(hradii=NULL, types=NULL, archy=NULL)
```

Arguments

hradii	Optional matrix of hard core distances
types	Optional; vector of all possible types (i.e. the possible levels of the marks variable in the data)
archy	Optional: the hierarchical order. See Details.

Details

This is a hierarchical point process model for a multitype point pattern (Högmander and Särkkä, 1999; Grabarnik and Särkkä, 2009). It is appropriate for analysing multitype point pattern data in which the types are ordered so that the points of type j depend on the points of type $1, 2, \ldots, j-1$.

The hierarchical version of the (stationary) hard core process with m types, with hard core distances h_{ij} and parameters β_j, is a point process in which each point of type j contributes a factor β_j to the probability density of the point pattern. If any pair of points of types i and j lies closer than h_{ij} units apart, the configuration of points is impossible (probability density zero).

The nonstationary hierarchical hard core process is similar except that the contribution of each individual point x_i is a function $\beta(x_i)$ of location and type, rather than a constant beta.

The function ppm(), which fits point process models to point pattern data, requires an argument of class "interact" describing the interpoint interaction structure of the model to be fitted. The appropriate description of the hierarchical hard core process pairwise interaction is yielded by the function HierHard(). See the examples below.

The argument types need not be specified in normal use. It will be determined automatically from the point pattern data set to which the HierHard interaction is applied, when the user calls ppm. However, the user should be confident that the ordering of types in the dataset corresponds to the ordering of rows and columns in the matrix radii.

The argument archy can be used to specify a hierarchical ordering of the types. It can be either a vector of integers or a character vector matching the possible types. The default is the sequence $1, 2, \ldots, m$ meaning that type j depends on types $1, 2, \ldots, j-1$.

The matrix iradii must be square, with entries which are either positive numbers, or zero or NA. A value of zero or NA indicates that no hard core interaction term should be included for this combination of types.

Note that only the hard core distances are specified in HierHard. The canonical parameters $\log(\beta_j)$ are estimated by ppm(), not fixed in HierHard().

Value

An object of class "interact" describing the interpoint interaction structure of the hierarchical hard core process with hard core distances $hradii[i, j]$.

Author(s)

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

, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>.

References

Grabarnik, P. and Särkkä, A. (2009) Modelling the spatial structure of forest stands by multivariate point processes with hierarchical interactions. *Ecological Modelling* **220**, 1232–1240.

Högmander, H. and Särkkä, A. (1999) Multitype spatial point patterns with hierarchical interactions. *Biometrics* **55**, 1051–1058.

See Also

`MultiHard` for the corresponding symmetrical interaction.

`HierStrauss`, `HierStraussHard`.

Examples

```
h <- matrix(c(4, NA, 10, 15), 2, 2)
HierHard(h)
# prints a sensible description of itself
ppm(ants ~1, HierHard(h))
# fit the stationary hierarchical hard core process to ants data
```

---

hierpair.family *Hierarchical Pairwise Interaction Process Family*

Description

An object describing the family of all hierarchical pairwise interaction Gibbs point processes.

Details

Advanced Use Only!

This structure would not normally be touched by the user. It describes the hierarchical pairwise interaction family of point process models.

Value

Object of class `"isf"`, see `isf.object`.

Author(s)

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

See Also

Other families: pairwise.family, pairsat.family, ord.family, inforder.family.

Hierarchical Strauss interaction: HierStrauss.

HierStrauss *The Hierarchical Strauss Point Process Model*

Description

Creates an instance of the hierarchical Strauss point process model which can then be fitted to point pattern data.

Usage

```
HierStrauss(radii, types=NULL, archy=NULL)
```

Arguments

radii	Matrix of interaction radii
types	Optional; vector of all possible types (i.e. the possible levels of the marks variable in the data)
archy	Optional: the hierarchical order. See Details.

Details

This is a hierarchical point process model for a multitype point pattern (Högmander and Särkkä, 1999; Grabarnik and Särkkä, 2009). It is appropriate for analysing multitype point pattern data in which the types are ordered so that the points of type j depend on the points of type $1, 2, \ldots, j-1$.

The hierarchical version of the (stationary) Strauss process with m types, with interaction radii r_{ij} and parameters β_j and γ_{ij} is a point process in which each point of type j contributes a factor β_j to the probability density of the point pattern, and a pair of points of types i and j closer than r_{ij} units apart contributes a factor γ_{ij} to the density **provided** $i \leq j$.

The nonstationary hierarchical Strauss process is similar except that the contribution of each individual point x_i is a function $\beta(x_i)$ of location and type, rather than a constant beta.

The function ppm(), which fits point process models to point pattern data, requires an argument of class "interact" describing the interpoint interaction structure of the model to be fitted. The appropriate description of the hierarchical Strauss process pairwise interaction is yielded by the function HierStrauss(). See the examples below.

The argument types need not be specified in normal use. It will be determined automatically from the point pattern data set to which the HierStrauss interaction is applied, when the user calls ppm. However, the user should be confident that the ordering of types in the dataset corresponds to the ordering of rows and columns in the matrix radii.

The argument archy can be used to specify a hierarchical ordering of the types. It can be either a vector of integers or a character vector matching the possible types. The default is the sequence $1, 2, \ldots, m$ meaning that type j depends on types $1, 2, \ldots, j-1$.

The matrix radii must be symmetric, with entries which are either positive numbers or NA. A value of NA indicates that no interaction term should be included for this combination of types.

Note that only the interaction radii are specified in HierStrauss. The canonical parameters $\log(\beta_j)$ and $\log(\gamma_{ij})$ are estimated by ppm(), not fixed in HierStrauss().

Value

An object of class "interact" describing the interpoint interaction structure of the hierarchical Strauss process with interaction radii $radii[i, j]$.

Author(s)

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

, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>.

References

Grabarnik, P. and Särkkä, A. (2009) Modelling the spatial structure of forest stands by multivariate point processes with hierarchical interactions. *Ecological Modelling* **220**, 1232–1240.

Högmander, H. and Särkkä, A. (1999) Multitype spatial point patterns with hierarchical interactions. *Biometrics* **55**, 1051–1058.

See Also

MultiStrauss for the corresponding symmetrical interaction.

HierHard, HierStraussHard.

Examples

```
r <- matrix(10 * c(3,4,4,3), nrow=2,ncol=2)
HierStrauss(r)
# prints a sensible description of itself
ppm(ants ~1, HierStrauss(r, , c("Messor", "Cataglyphis")))
# fit the stationary hierarchical Strauss process to ants data
```

HierStraussHard *The Hierarchical Strauss Hard Core Point Process Model*

Description

Creates an instance of the hierarchical Strauss-hard core point process model which can then be fitted to point pattern data.

Usage

```
HierStraussHard(iradii, hradii=NULL, types=NULL, archy=NULL)
```

Arguments

iradii	Matrix of interaction radii
hradii	Optional matrix of hard core distances
types	Optional; vector of all possible types (i.e. the possible levels of the marks variable in the data)
archy	Optional: the hierarchical order. See Details.

Details

This is a hierarchical point process model for a multitype point pattern (Högmander and Särkkä, 1999; Grabarnik and Särkkä, 2009). It is appropriate for analysing multitype point pattern data in which the types are ordered so that the points of type j depend on the points of type $1, 2, \ldots, j-1$.

The hierarchical version of the (stationary) Strauss hard core process with m types, with interaction radii r_{ij}, hard core distances h_{ij} and parameters β_j and γ_{ij} is a point process in which each point of type j contributes a factor β_j to the probability density of the point pattern, and a pair of points of types i and j closer than r_{ij} units apart contributes a factor γ_{ij} to the density **provided** $i \leq j$. If any pair of points of types i and j lies closer than h_{ij} units apart, the configuration of points is impossible (probability density zero).

The nonstationary hierarchical Strauss hard core process is similar except that the contribution of each individual point x_i is a function $\beta(x_i)$ of location and type, rather than a constant beta.

The function ppm(), which fits point process models to point pattern data, requires an argument of class "interact" describing the interpoint interaction structure of the model to be fitted. The appropriate description of the hierarchical Strauss hard core process pairwise interaction is yielded by the function HierStraussHard(). See the examples below.

The argument types need not be specified in normal use. It will be determined automatically from the point pattern data set to which the HierStraussHard interaction is applied, when the user calls ppm. However, the user should be confident that the ordering of types in the dataset corresponds to the ordering of rows and columns in the matrix radii.

The argument archy can be used to specify a hierarchical ordering of the types. It can be either a vector of integers or a character vector matching the possible types. The default is the sequence $1, 2, \ldots, m$ meaning that type j depends on types $1, 2, \ldots, j-1$.

The matrices iradii and hradii must be square, with entries which are either positive numbers or zero or NA. A value of zero or NA indicates that no interaction term should be included for this combination of types.

Note that only the interaction radii and hard core distances are specified in HierStraussHard. The canonical parameters $\log(\beta_j)$ and $\log(\gamma_{ij})$ are estimated by ppm(), not fixed in HierStraussHard().

Value

An object of class "interact" describing the interpoint interaction structure of the hierarchical Strauss-hard core process with interaction radii $iradii[i, j]$ and hard core distances $hradii[i, j]$.

Author(s)

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

, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>.

References

Grabarnik, P. and Särkkä, A. (2009) Modelling the spatial structure of forest stands by multivariate point processes with hierarchical interactions. *Ecological Modelling* **220**, 1232–1240.

Högmander, H. and Särkkä, A. (1999) Multitype spatial point patterns with hierarchical interactions. *Biometrics* **55**, 1051–1058.

See Also

MultiStraussHard for the corresponding symmetrical interaction.

HierHard, HierStrauss.

Examples

```
r <- matrix(c(30, NA, 40, 30), nrow=2,ncol=2)
h <- matrix(c(4, NA, 10, 15), 2, 2)
HierStraussHard(r, h)
# prints a sensible description of itself
ppm(ants ~1, HierStraussHard(r, h))
# fit the stationary hierarchical Strauss-hard core process to ants data
```

---

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 <r.turner@auckland.ac.nz> 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)
```

`Hybrid` *Hybrid Interaction Point Process Model*

Description

Creates an instance of a hybrid point process model which can then be fitted to point pattern data.

Usage

```
Hybrid(...)
```

Arguments

`...`	Two or more interactions (objects of class `"interact"`) or objects which can be converted to interactions. See Details.

Details

A *hybrid* (Baddeley, Turner, Mateu and Bevan, 2013) is a point process model created by combining two or more point process models, or an interpoint interaction created by combining two or more interpoint interactions.

The *hybrid* of two point processes, with probability densities $f(x)$ and $g(x)$ respectively, is the point process with probability density

$$h(x) = c\, f(x)\, g(x)$$

where c is a normalising constant.

Equivalently, the hybrid of two point processes with conditional intensities $\lambda(u, x)$ and $\kappa(u, x)$ is the point process with conditional intensity

$$\phi(u, x) = \lambda(u, x)\, \kappa(u, x).$$

The hybrid of $m > 3$ point processes is defined in a similar way.

The function `ppm`, which fits point process models to point pattern data, requires an argument of class `"interact"` describing the interpoint interaction structure of the model to be fitted. The appropriate description of a hybrid interaction is yielded by the function `Hybrid()`.

The arguments `...` will be interpreted as interpoint interactions (objects of class `"interact"`) and the result will be the hybrid of these interactions. Each argument must either be an interpoint interaction (object of class `"interact"`), or a point process model (object of class `"ppm"`) from which the interpoint interaction will be extracted.

The arguments `...` may also be given in the form `name=value`. This is purely cosmetic: it can be used to attach simple mnemonic names to the component interactions, and makes the printed output from `print.ppm` neater.

Value

An object of class `"interact"` describing an interpoint interaction structure.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A., Turner, R., Mateu, J. and Bevan, A. (2013) Hybrids of Gibbs point process models and their implementation. *Journal of Statistical Software* **55**:11, 1–43. DOI: 10.18637/jss.v055.i11

See Also

`ppm`

Examples

```
Hybrid(Strauss(0.1), Geyer(0.2, 3))

Hybrid(Ha=Hardcore(0.05), St=Strauss(0.1), Ge=Geyer(0.2, 3))

fit <- ppm(redwood, ~1, Hybrid(A=Strauss(0.02), B=Geyer(0.1, 2)))
fit

ctr <- rmhcontrol(nrep=5e4, expand=1)
plot(simulate(fit, control=ctr))

# hybrid components can be models (including hybrid models)
Hybrid(fit, S=Softcore(0.5))

# plot.fii only works if every component is a pairwise interaction
data(swedishpines)
fit2 <- ppm(swedishpines, ~1, Hybrid(DG=DiggleGratton(2,10), S=Strauss(5)))
plot(fitin(fit2))
plot(fitin(fit2), separate=TRUE, mar.panel=rep(4,4))
```

`hybrid.family` *Hybrid Interaction Family*

Description

An object describing the family of all hybrid interactions.

Details

Advanced Use Only!

This structure would not normally be touched by the user. It describes the family of all hybrid point process models.

If you need to create a specific hybrid interaction model for use in modelling, use the function `Hybrid`.

Value

Object of class `"isf"`, see `isf.object`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>.

See Also

Use `Hybrid` to make hybrid interactions.

Other families: `pairwise.family`, `pairsat.family`, `ord.family`, `inforder.family`.

`ic.kppm` *Model selection criteria for the intensity function of a point process*

Description

Information criteria for selecting the intensity function model of a Poisson, cluster or Cox point process.

Usage

```
  ic(object)

  ## S3 method for class 'ppm'
ic(object)

  ## S3 method for class 'kppm'
ic(object)
```

Arguments

`object`	Fitted point process model (object of class `"ppm"` or `"kppm"`).

Details

This function returns information criteria for selecting the intensity function model of a Poisson, Cox or cluster point process fitted by first order composite likelihood (i.e. using the Poisson likelihood function).

Degrees of freedom df for the information criteria are given by the trace of $S^{-1}\Sigma$ where S is the sensitivity matrix and Σ is the variance matrix for the log composite likelihood score function. In case of a Poisson process, df is the number of parameters in the model for the intensity function.

The composite Bayesian information criterion (cbic) is $-2\ell + \log(n)df$ where ℓ is the maximal log first-order composite likelihood (Poisson loglikelihood for the intensity function) and n is the observed number of points. It reduces to the BIC criterion in case of a Poisson process.

The composite information criterion (cic) is $-2\ell + 2df$ and reduces to the AIC in case of a Poisson process.

NOTE: the information criteria are for selecting the intensity function model (a set of covariates) within a given model class. They cannot be used to choose among different types of cluster or Cox point process models (e.g. can not be used to choose between Thomas and LGCP models).

Value

A list with entries `loglike`, `cbic`, `cic` and `df`. Here `loglike` is the fitted log first-order composite likelihood, `cbic` is composite Bayesian information criterion, `cic` is is the composite likelihood criterion and `df` is the adjusted degrees of freedom for the fitted intensity function model.

Author(s)

Achmad Choiruddin, Jean-Francois Coeurjolly and Rasmus Waagepetersen.

References

Choiruddin, A., Coeurjolly, J.F. and Waagepetersen, R. (2020) Information criteria for inhomogeneous spatial point processes. *Australian and New Zealand Journal of Statistics*. To appear.

See Also

kppm

Examples

```
if(interactive()) {

  # model with one covariate
  fit1 <- kppm(bei~elev,data=bei.extra)
  ic1  <- ic(fit1)

  # model with two covariates
  fit2 <- kppm(bei~elev+grad,data=bei.extra)
  ic2  <- ic(fit2)

  # smallest cbic for fit1 but smallest cic for fit2
}
```

---

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 <r.turner@auckland.ac.nz> 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 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 <r.turner@auckland.ac.nz>

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

```
data(amacrine)
Ic <- Iest(amacrine)
plot(Ic, main="Amacrine Cells data")
# values are below I= 0, suggesting negative association
# between 'on' and 'off' cells.
```

---

improve.kppm *Improve Intensity Estimate of Fitted Cluster Point Process Model*

Description

Update the fitted intensity of a fitted cluster point process model.

Usage

```
improve.kppm(object, type=c("quasi", "wclik1", "clik1"), rmax = NULL,
             eps.rmax = 0.01, dimyx = 50, maxIter = 100, tolerance = 1e-06,
             fast = TRUE, vcov = FALSE, fast.vcov = FALSE, verbose = FALSE,
                          save.internals = FALSE)
```

Arguments

object	Fitted cluster point process model (object of class "kppm").
type	A character string indicating the method of estimation. Current options are "clik1", "wclik1" and "quasi" for, respectively, first order composite (Poisson) likelihood, weighted first order composite likelihood and quasi-likelihood.
rmax	Optional. The dependence range. Not usually specified by the user.
eps.rmax	Numeric. A small positive number which is used to determine rmax from the tail behaviour of the pair correlation function. Namely rmax is the smallest value of r at which $(g(r) - 1)/(g(0) - 1)$ falls below eps.rmax. Ignored if rmax is provided.
dimyx	Pixel array dimensions. See Details.
maxIter	Integer. Maximum number of iterations of iterative weighted least squares (Fisher scoring).
tolerance	Numeric. Tolerance value specifying when to stop iterative weighted least squares (Fisher scoring).
fast	Logical value indicating whether tapering should be used to make the computations faster (requires the package **Matrix**).
vcov	Logical value indicating whether to calculate the asymptotic variance covariance/matrix.
fast.vcov	Logical value indicating whether tapering should be used for the variance/covariance matrix to make the computations faster (requires the package **Matrix**). Caution: This is expected to underestimate the true asymptotic variances/covariances.
verbose	A logical indicating whether the details of computations should be printed.
save.internals	A logical indicating whether internal quantities should be saved in the returned object (mostly for development purposes).

Details

This function reestimates the intensity parameters in a fitted "kppm" object. If type="clik1" estimates are based on the first order composite (Poisson) likelihood, which ignores dependence between the points. Note that type="clik1" is mainly included for testing purposes and is not recommended for the typical user; instead the more efficient kppm with improve.type="none" should be used.

When type="quasi" or type="wclik1" the dependence structure between the points is incorporated in the estimation procedure by using the estimated pair correlation function in the estimating equation.

In all cases the estimating equation is based on dividing the observation window into small subregions and count the number of points in each subregion. To do this the observation window is first converted into a digital mask by as.mask where the resolution is controlled by the argument dimyx. The computational time grows with the cube of the number of subregions, so fine grids may take very long to compute (or even run out of memory).

Value

A fitted cluster point process model of class "kppm".

Author(s)

Abdollah Jalilian <jalilian@razi.ac.ir>

and Rasmus Waagepetersen <rw@math.aau.dk> adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu.

and Ege Rubak <rubak@math.aau.dk>

References

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes, *Biometrics*, **63**, 252-258.

Guan, Y. and Shen, Y. (2010) A weighted estimating equation approach to inference for inhomogeneous spatial point processes, *Biometrika*, **97**, 867-880.

Guan, Y., Jalilian, A. and Waagepetersen, R. (2015) Quasi-likelihood for spatial point processes. *Journal of the Royal Statistical Society, Series B* **77**, 677–697.

See Also

ppm, kppm, improve.kppm

Examples

```
# fit a Thomas process using minimum contrast estimation method
# to model interaction between points of the pattern
fit0 <- kppm(bei ~ elev + grad, data = bei.extra)

# fit the log-linear intensity model with quasi-likelihood method
fit1 <- improve.kppm(fit0, type="quasi")

# compare
coef(fit0)
coef(fit1)
```

---

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 <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

`fv.object`, `deriv.fv`

Examples

```
plot(increment.fv(Kest(cells), 0.05))
```

`influence.ppm` *Influence Measure for Spatial Point Process Model*

Description

Computes the influence measure for a fitted spatial point process model.

Usage

```
## S3 method for class 'ppm'
influence(model, ...,
        drop = FALSE, iScore=NULL, iHessian=NULL, iArgs=NULL)
```

Arguments

`model`	Fitted point process model (object of class `"ppm"`).
`...`	Ignored.
`drop`	Logical. Whether to include (`drop=FALSE`) or exclude (`drop=TRUE`) contributions from quadrature points that were not used to fit the model.
`iScore,iHessian`	Components of the score vector and Hessian matrix for the irregular parameters, if required. See Details.
`iArgs`	List of extra arguments for the functions `iScore`, `iHessian` if required.

Details

Given a fitted spatial point process model `model`, this function computes the influence measure described in Baddeley, Chang and Song (2013) and Baddeley, Rubak and Turner (2019).

The function `influence` is generic, and `influence.ppm` is the method for objects of class `"ppm"` representing point process models.

The influence of a point process model is a value attached to each data point (i.e. each point of the point pattern to which the `model` was fitted). The influence value $s(x_i)$ at a data point x_i represents the change in the maximised log (pseudo)likelihood that occurs when the point x_i is deleted. A relatively large value of $s(x_i)$ indicates a data point with a large influence on the fitted model.

If the point process model trend has irregular parameters that were fitted (using `ippm`) then the influence calculation requires the first and second derivatives of the log trend with respect to the irregular parameters. The argument `iScore` should be a list, with one entry for each irregular parameter, of R functions that compute the partial derivatives of the log trend (i.e. log intensity or log conditional intensity) with respect to each irregular parameter. The argument `iHessian` should be a list, with p^2 entries where p is the number of irregular parameters, of R functions that compute the second order partial derivatives of the log trend with respect to each pair of irregular parameters.

The result of `influence.ppm` is an object of class `"influence.ppm"`. It can be printed and plotted. It can be converted to a marked point pattern by `as.ppp` (see `as.ppp.influence.ppm`). There are also methods for `[`, `as.owin`, `domain`, `shift`, `integral` and `Smooth`.

Value

An object of class `"influence.ppm"`.

Author(s)

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

References

Baddeley, A. and Chang, Y.M. and Song, Y. (2013) Leverage and influence diagnostics for spatial point process models. *Scandinavian Journal of Statistics* **40**, 86–104.

Baddeley, A., Rubak, E. and Turner, R. (2019) Leverage and influence diagnostics for Gibbs spatial point processes. *Spatial Statistics* **29**, 15–48.

See Also

`leverage.ppm`, `dfbetas.ppm`, `ppmInfluence`, `plot.influence.ppm`

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X ~x+y)
plot(influence(fit))
```

inforder.family *Infinite Order Interaction Family*

Description

An object describing the family of all Gibbs point processes with infinite interaction order.

Details

Advanced Use Only!

This structure would not normally be touched by the user. It describes the interaction structure of Gibbs point processes which have infinite order of interaction, such as the area-interaction process *AreaInter*.

Value

Object of class "isf", see isf.object.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.

See Also

AreaInter to create the area interaction process structure.

Other families: pairwise.family, pairsat.family, ord.family.

integral.msr *Integral of a Measure*

Description

Computes the integral (total value) of a measure over its domain.

Usage

```
## S3 method for class 'msr'
integral(f, domain=NULL, ...)
```

Arguments

f	A signed measure or vector-valued measure (object of class "msr").
domain	Optional window specifying the domain of integration. Alternatively a tessellation.
...	Ignored.

Details

The integral (total value of the measure over its domain) is calculated.

If domain is a window (class "owin") then the integration will be restricted to this window. If domain is a tessellation (class "tess") then the integral of f in each tile of domain will be computed.

For a multitype measure m, use split.msr to separate the contributions for each type of point, as shown in the Examples.

Value

A numeric value, vector, or matrix.

integral(f) returns a numeric value (for a signed measure) or a vector of values (for a vector-valued measure).

If domain is a tessellation then integral(f, domain) returns a numeric vector with one entry for each tile (if f is a signed measure) or a numeric matrix with one row for each tile (if f is a vector-valued measure).

Author(s)

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

See Also

msr, integral

Examples

```
fit <- ppm(cells ~ x)
rr <- residuals(fit)
integral(rr)

# vector-valued measure
rs <- residuals(fit, type="score")
integral(rs)

# multitype
fitA <- ppm(amacrine ~ x)
rrA <- residuals(fitA)
sapply(split(rrA), integral)

# multitype and vector-valued
```

```
rsA <- residuals(fitA, type="score")
sapply(split(rsA), integral)

## integral over a subregion
integral(rr, domain=square(0.5))
## integrals over the tiles of a tessellation
integral(rr, domain=quadrats(cells, 2))
```

`intensity.dppm` *Intensity of Determinantal Point Process Model*

Description

Extracts the intensity of a determinantal point process model.

Usage

```
  ## S3 method for class 'detpointprocfamily'
intensity(X, ...)

  ## S3 method for class 'dppm'
intensity(X, ...)
```

Arguments

`X`	A determinantal point process model (object of class `"detpointprocfamily"` or `"dppm"`).
`...`	Ignored.

Value

A numeric value (if the model is stationary), a pixel image (if the model is non-stationary) or `NA` if the intensity is unknown for the model.

Author(s)

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

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

`intensity.ppm` *Intensity of Fitted Point Process Model*

Description

Computes the intensity of a fitted point process model.

Usage

```
 ## S3 method for class 'ppm'
intensity(X, ..., approx=c("Poisson", "DPP"))
```

Arguments

`X`	A fitted point process model (object of class `"ppm"`).
`...`	Arguments passed to `predict.ppm` in some cases. See Details.
`approx`	Character string (partially matched) specifying the type of approximation to the intensity for a non-Poisson model.

Details

This is a method for the generic function `intensity` for fitted point process models (class `"ppm"`).

The intensity of a point process model is the expected number of random points per unit area.

If `X` is a Poisson point process model, the intensity of the process is computed exactly. The result is a numerical value if `X` is a stationary Poisson point process, and a pixel image if `X` is non-stationary. (In the latter case, the resolution of the pixel image is controlled by the arguments `...` which are passed to `predict.ppm`.)

If `X` is a Gibbs point process model that is not a Poisson model, the intensity is computed approximately:

- if `approx="Poisson"` (the default), the intensity is computed using the Poisson-saddlepoint approximation (Baddeley and Nair, 2012a, 2012b, 2017; Anderssen et al, 2014). This approximation is currently available for pairwise-interaction models (Baddeley and Nair, 2012a, 2012b) and for the area-interaction model and Geyer saturation model (Baddeley and Nair, 2017).

  If the model is non-stationary. the pseudostationary solution (Baddeley and Nair, 2012b; Anderssen et al, 2014) is used. The result is a pixel image, whose resolution is controlled by the arguments `...` which are passed to `predict.ppm`.
- if `approx="DPP"`, the intensity is calculated using the approximation of (Coeurjolly and Lavancier, 2018) based on a determinantal point process. This approximation is more accurate than the Poisson saddlepoint approximation, for inhibitory interactions. However the DPP approximation is only available for stationary pairwise interaction models.

Value

A numeric value (if the model is stationary) or a pixel image.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Gopalan Nair, and Frédéric Lavancier.

References

Anderssen, R.S., Baddeley, A., DeHoog, F.R. and Nair, G.M. (2014) Solution of an integral equation arising in spatial point process theory. *Journal of Integral Equations and Applications* **26** (4) 437–453.

Baddeley, A. and Nair, G. (2012a) Fast approximation of the intensity of Gibbs point processes. *Electronic Journal of Statistics* **6** 1155–1169.

Baddeley, A. and Nair, G. (2012b) Approximating the moments of a spatial point process. *Stat* **1**, 1, 18–30. DOI: 10.1002/sta4.5

Baddeley, A. and Nair, G. (2017) Poisson-saddlepoint approximation for Gibbs point processes with infinite-order interaction: in memory of Peter Hall. *Journal of Applied Probability* **54**, 4, 1008–1026.

Coeurjolly, J.-F. and Lavancier, F. (2018) Approximation intensity for pairwise interaction Gibbs point processes using determinantal point processes. *Electronic Journal of Statistics* **12** 3181–3203.

See Also

intensity, intensity.ppp

Examples

```
fitP <- ppm(swedishpines ~ 1)
intensity(fitP)
fitS <- ppm(swedishpines ~ 1, Strauss(9))
intensity(fitS)
intensity(fitS, approx="D")
fitSx <- ppm(swedishpines ~ x, Strauss(9))
lamSx <- intensity(fitSx)
fitG <- ppm(swedishpines ~ 1, Geyer(9, 1))
lamG <- intensity(fitG)
fitA <- ppm(swedishpines ~ 1, AreaInter(7))
lamA <- intensity(fitA)
```

---

intensity.slrm *Intensity of Fitted Spatial Logistic Regression Model*

Description

Computes the intensity of a fitted spatial logistic regression model, treated as a point process model.

Usage

```
## S3 method for class 'slrm'
intensity(X, ...)
```

Arguments

X	A fitted spatial logistic regression model (object of class "slrm").
...	Arguments passed to predict.slrm in some cases. See Details.

Details

This is a method for the generic function intensity for spatial logistic regression models (class "slrm").

The fitted spatial logistic regression model X is interpreted as a point process model. The intensity of a point process model is defined as the expected number of random points per unit area. The fitted probabilities of presence according to X are converted to intensity values.

The result is a numerical value if X is stationary, and a pixel image if X is non-stationary. In the latter case, the resolution of the pixel image is controlled by the arguments ... which are passed to predict.slrm.

Value

A numeric value (if the model is stationary) or a pixel image.

Author(s)

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

References

Baddeley, A., Berman, M., Fisher, N.I., Hardegen, A., Milne, R.K., Schuhmacher, D., Shah, R. and Turner, R. (2010) Spatial logistic regression and change-of-support for spatial Poisson point processes. *Electronic Journal of Statistics* **4**, 1151–1201. DOI: 10.1214/10-EJS581

See Also

intensity, intensity.ppm

Examples

```
fitS <- slrm(swedishpines ~ 1)
intensity(fitS)
fitX <- slrm(swedishpines ~ x)
intensity(fitX)
```

`interactionorder` *Determine the Order of Interpoint Interaction in a Model*

Description

Given a point process model, report the order of interpoint interaction.

Usage

```
interactionorder(object)

## S3 method for class 'ppm'
interactionorder(object)

## S3 method for class 'interact'
interactionorder(object)

## S3 method for class 'isf'
interactionorder(object)

## S3 method for class 'fii'
interactionorder(object)
```

Arguments

`object`	A point process model (class `"ppm"`) or similar information.

Details

This function determines the order of interpoint interaction in a Gibbs point process model (or a related object).

The interaction order is defined as the largest number `k` such that the probability density of the model contains terms involving `k` points at a time. For example, in a pairwise interaction process such as the Strauss process, the probability density contains interaction terms between each pair of points, but does not contain any terms that involve three points at a time, so the interaction order is 2.

Poisson point processes have interaction order 1. Pairwise-interaction processes have interaction order 2. Point processes with the triplet interaction `Triplets` have interaction order 3. The Geyer saturation model `Geyer` and the area-interaction model `AreaInter` have infinite order of interaction.

Value

A positive integer, or `Inf`.

Author(s)

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

Examples

```
interactionorder(ppm(cells ~ 1))
interactionorder(Strauss(0.1))
interactionorder(Triplets(0.1))
interactionorder(Geyer(0.1, 2))
interactionorder(Hybrid(Strauss(0.1), Triplets(0.2)))
```

---

ippm *Fit Point Process Model Involving Irregular Trend Parameters*

Description

Experimental extension to ppm which finds optimal values of the irregular trend parameters in a point process model.

Usage

```
ippm(Q, ...,
          iScore=NULL,
          start=list(),
          covfunargs=start,
          nlm.args=list(stepmax=1/2),
          silent=FALSE,
          warn.unused=TRUE)
```

Arguments

Q,...	Arguments passed to ppm to fit the point process model.
iScore	Optional. A named list of R functions that compute the partial derivatives of the logarithm of the trend, with respect to each irregular parameter. See Details.
start	Named list containing initial values of the irregular parameters over which to optimise.
covfunargs	Argument passed to ppm. A named list containing values for *all* irregular parameters required by the covariates in the model. Must include all the parameters named in start.
nlm.args	Optional list of arguments passed to nlm to control the optimization algorithm.
silent	Logical. Whether to print warnings if the optimization algorithm fails to converge.
warn.unused	Logical. Whether to print a warning if some of the parameters in start are not used in the model.

Details

This function is an experimental extension to the point process model fitting command ppm. The extension allows the trend of the model to include irregular parameters, which will be maximised by a Newton-type iterative method, using nlm.

For the sake of explanation, consider a Poisson point process with intensity function $\lambda(u)$ at location u. Assume that

$$\lambda(u) = \exp(\alpha + \beta Z(u))\, f(u, \gamma)$$

where α, β, γ are parameters to be estimated, $Z(u)$ is a spatial covariate function, and f is some known function. Then the parameters α, β are called *regular* because they appear in a loglinear form; the parameter γ is called *irregular*.

To fit this model using `ippm`, we specify the intensity using the `trend` formula in the same way as usual for ppm. The trend formula is a representation of the log intensity. In the above example the log intensity is

$$\log \lambda(u) = \alpha + \beta Z(u) + \log f(u, \gamma)$$

So the model above would be encoded with the trend formula `~Z + offset(log(f))`. Note that the irregular part of the model is an *offset* term, which means that it is included in the log trend as it is, without being multiplied by another regular parameter.

The optimisation runs faster if we specify the derivative of $\log f(u, \gamma)$ with respect to γ. We call this the *irregular score*. To specify this, the user must write an R function that computes the irregular score for any value of γ at any location `(x,y)`.

Thus, to code such a problem,

1. The argument `trend` should define the log intensity, with the irregular part as an offset;
2. The argument `start` should be a list containing initial values of each of the irregular parameters;
3. The argument `iScore`, if provided, must be a list (with one entry for each entry of `start`) of functions with arguments `x,y,...`, that evaluate the partial derivatives of $\log f(u, \gamma)$ with respect to each irregular parameter.

The coded example below illustrates the model with two irregular parameters γ, δ and irregular term

$$f((x, y), (\gamma, \delta)) = 1 + \exp(\gamma - \delta x^3)$$

Arguments `...` passed to ppm may also include `interaction`. In this case the model is not a Poisson point process but a more general Gibbs point process; the trend formula `trend` determines the first-order trend of the model (the first order component of the conditional intensity), not the intensity.

Value

A fitted point process model (object of class `"ppm"`) which also belongs to the special class `"ippm"`.

Author(s)

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

See Also

ppm, profilepl

Examples

```
nd <- 32


gamma0 <- 3
delta0 <- 5
POW <- 3
# Terms in intensity
Z <- function(x,y) { -2*y }
f <- function(x,y,gamma,delta) { 1 + exp(gamma - delta * x^POW) }
# True intensity
lamb <- function(x,y,gamma,delta) { 200 * exp(Z(x,y)) * f(x,y,gamma,delta) }
# Simulate realisation
lmax <- max(lamb(0,0,gamma0,delta0), lamb(1,1,gamma0,delta0))
set.seed(42)
X <- rpoispp(lamb, lmax=lmax, win=owin(), gamma=gamma0, delta=delta0)
# Partial derivatives of log f
DlogfDgamma <- function(x,y, gamma, delta) {
  topbit <- exp(gamma - delta * x^POW)
  topbit/(1 + topbit)
}
DlogfDdelta <- function(x,y, gamma, delta) {
  topbit <- exp(gamma - delta * x^POW)
  - (x^POW) * topbit/(1 + topbit)
}
# irregular score
Dlogf <- list(gamma=DlogfDgamma, delta=DlogfDdelta)
# fit model
ippm(X ~Z + offset(log(f)),
     covariates=list(Z=Z, f=f),
     iScore=Dlogf,
     start=list(gamma=1, delta=1),
     nlm.args=list(stepmax=1),
     nd=nd)
```

---

is.dppm *Recognise Fitted Determinantal Point Process Models*

Description

Check that an object inherits the class dppm

Usage

```
is.dppm(x)
```

Arguments

x	Any object.

Value

A single logical value.

Author(s)

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

---

is.hybrid *Test Whether Object is a Hybrid*

Description

Tests where a point process model or point process interaction is a hybrid of several interactions.

Usage

```
is.hybrid(x)

## S3 method for class 'ppm'
is.hybrid(x)

## S3 method for class 'interact'
is.hybrid(x)
```

Arguments

x	A point process model (object of class "ppm") or a point process interaction structure (object of class "interact").

Details

A *hybrid* (Baddeley, Turner, Mateu and Bevan, 2012) is a point process model created by combining two or more point process models, or an interpoint interaction created by combining two or more interpoint interactions.

The function is.hybrid is generic, with methods for point process models (objects of class "ppm") and point process interactions (objects of class "interact"). These functions return TRUE if the object x is a hybrid, and FALSE if it is not a hybrid.

Hybrids of two or more interpoint interactions are created by the function Hybrid. Such a hybrid interaction can then be fitted to point pattern data using ppm.

Value

TRUE if the object is a hybrid, and FALSE otherwise.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A., Turner, R., Mateu, J. and Bevan, A. (2013) Hybrids of Gibbs point process models and their implementation. *Journal of Statistical Software* **55**:11, 1–43. DOI: 10.18637/jss.v055.i11

See Also

Hybrid

Examples

```
S <- Strauss(0.1)
is.hybrid(S)
H <- Hybrid(Strauss(0.1), Geyer(0.2, 3))
is.hybrid(H)

data(redwood)
fit <- ppm(redwood, ~1, H)
is.hybrid(fit)
```

---

is.marked.ppm *Test Whether A Point Process Model is Marked*

Description

Tests whether a fitted point process model involves "marks" attached to the points.

Usage

```
  ## S3 method for class 'ppm'
is.marked(X, ...)
```

Arguments

X	Fitted point process model (object of class "ppm") usually obtained from ppm.
...	Ignored.

Details

"Marks" are observations attached to each point of a point pattern. For example the longleaf dataset contains the locations of trees, each tree being marked by its diameter; the amacrine dataset gives the locations of cells of two types (on/off) and the type of cell may be regarded as a mark attached to the location of the cell.

The argument X is a fitted point process model (an object of class "ppm") typically obtained by fitting a model to point pattern data using ppm.

This function returns TRUE if the *original data* (to which the model X was fitted) were a marked point pattern.

Note that this is not the same as testing whether the model involves terms that depend on the marks (i.e. whether the fitted model ignores the marks in the data). See the Examples for a trick to do this.

If this function returns TRUE, the implications are (for example) that any simulation of this model will require simulation of random marks as well as random point locations.

Value

Logical value, equal to TRUE if X is a model that was fitted to a marked point pattern dataset.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>.

See Also

is.marked, is.marked.ppp

Examples

```
 X <- lansing
 # Multitype point pattern --- trees marked by species



fit1 <- ppm(X, ~ marks, Poisson())
is.marked(fit1)

fit2 <- ppm(X, ~ 1, Poisson())
is.marked(fit2)

## test whether the model formula involves marks
"marks" %in% spatstat.utils::variablesinformula(formula(fit2))

# Unmarked point pattern
fit3 <- ppm(cells, ~ 1, Poisson())
is.marked(fit3)
# FALSE
```

`is.multitype.ppm` *Test Whether A Point Process Model is Multitype*

Description

Tests whether a fitted point process model involves "marks" attached to the points that classify the points into several types.

Usage

```
  ## S3 method for class 'ppm'
is.multitype(X, ...)
```

Arguments

`X`	Fitted point process model (object of class `"ppm"`) usually obtained from `ppm`.
`...`	Ignored.

Details

"Marks" are observations attached to each point of a point pattern. For example the `longleaf` dataset contains the locations of trees, each tree being marked by its diameter; the `amacrine` dataset gives the locations of cells of two types (on/off) and the type of cell may be regarded as a mark attached to the location of the cell.

The argument `X` is a fitted point process model (an object of class `"ppm"`) typically obtained by fitting a model to point pattern data using `ppm`.

This function returns `TRUE` if the *original data* (to which the model `X` was fitted) were a multitype point pattern.

Note that this is not the same as testing whether the model involves terms that depend on the marks (i.e. whether the fitted model ignores the marks in the data). Currently we have not implemented a test for this.

If this function returns `TRUE`, the implications are (for example) that any simulation of this model will require simulation of random marks as well as random point locations.

Value

Logical value, equal to `TRUE` if `X` is a model that was fitted to a multitype point pattern dataset.

Author(s)

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

and Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

is.multitype, is.multitype.ppp

Examples

```
X <- lansing
# Multitype point pattern --- trees marked by species



fit1 <- ppm(X, ~ marks, Poisson())
is.multitype(fit1)
# TRUE

fit2 <- ppm(X, ~ 1, Poisson())
is.multitype(fit2)
# TRUE

# Unmarked point pattern
fit3 <- ppm(cells, ~ 1, Poisson())
is.multitype(fit3)
# FALSE
```

---

is.ppm *Test Whether An Object Is A Fitted Point Process Model*

Description

Checks whether its argument is a fitted point process model (object of class "ppm", "kppm", "lppm" or "slrm").

Usage

```
is.ppm(x)
is.kppm(x)
is.lppm(x)
is.slrm(x)
```

Arguments

x	Any object.

Details

These functions test whether the object x is a fitted point process model object of the specified class.

The result of is.ppm(x) is TRUE if x has "ppm" amongst its classes, and otherwise FALSE. Similarly for the other functions.

Value

A single logical value.

Author(s)

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

`is.stationary.ppm` *Recognise Stationary and Poisson Point Process Models*

Description

Given a point process model (either a model that has been fitted to data, or a model specified by its parameters), determine whether the model is a stationary point process, and whether it is a Poisson point process.

Usage

```
## S3 method for class 'ppm'
is.stationary(x)
## S3 method for class 'kppm'
is.stationary(x)
## S3 method for class 'slrm'
is.stationary(x)
## S3 method for class 'dppm'
is.stationary(x)
## S3 method for class 'detpointprocfamily'
is.stationary(x)

## S3 method for class 'ppm'
is.poisson(x)
## S3 method for class 'kppm'
is.poisson(x)
## S3 method for class 'slrm'
is.poisson(x)
## S3 method for class 'interact'
is.poisson(x)
```

Arguments

`x`	A fitted spatial point process model (object of class `"ppm"`, `"kppm"`, `"lppm"`, `"dppm"` or `"slrm"`) or a specification of a Gibbs point process model (object of class `"rmhmodel"`) or a similar object.

Details

The argument x represents a fitted spatial point process model or a similar object.

is.stationary(x) returns TRUE if x represents a stationary point process, and FALSE if not.

is.poisson(x) returns TRUE if x represents a Poisson point process, and FALSE if not.

The functions is.stationary and is.poisson are generic, with methods for the classes "ppm" (Gibbs point process models), "kppm" (cluster or Cox point process models), "slrm" (spatial logistic regression models) and "rmhmodel" (model specifications for the Metropolis-Hastings algorithm). Additionally is.stationary has a method for classes "detpointprocfamily" and "dppm" (both determinantal point processes) and is.poisson has a method for class "interact" (interaction structures for Gibbs models).

is.poisson.kppm will return FALSE, unless the model x is degenerate: either x has zero intensity so that its realisations are empty with probability 1, or it is a log-Gaussian Cox process where the log intensity has zero variance.

is.poisson.slrm will always return TRUE, by convention.

Value

A logical value.

Author(s)

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

See Also

is.marked to determine whether a model is a marked point process.

summary.ppm for detailed information about a fitted model.

Model-fitting functions ppm, dppm, kppm, slrm.

Examples

```
fit <- ppm(cells ~ x)
is.stationary(fit)
is.poisson(fit)

fut <- kppm(redwood ~ 1, "MatClust")
is.stationary(fut)
is.poisson(fut)

fot <- slrm(cells ~ x)
is.stationary(fot)
is.poisson(fot)
```

`isf.object` *Interaction Structure Family Objects*

Description

Objects of class `"isf"` are used internally by the **spatstat** package to represent the structure of the interpoint interactions in a family of point process models.

Details

Advanced Use Only!

An object of class `"isf"` (Interaction Structure Family) is used internally by the **spatstat** package to represent the common mathematical and algorithmic structure of the interpoint interactions in a family of point process models.

The existing objects of class `"isf"` are:

`pairwise.family`	pairwise interaction
`triplet.family`	triplet interaction
`pairsat.family`	saturated pairwise interaction
`hierpair.family`	hierarchical pairwise interaction
`inforder.family`	infinite order interaction
`hybrid.family`	hybrids of several interactions
`ord.family`	Ord interactions

The information contained in these objects enables the **spatstat** package to select the appropriate algorithm for fitting, predicting and simulating each point process model.

For example, in order to fit a model that involves pairwise interactions, the model-fitting function `ppm` would use information contained in `pairwise.family` to select the appropriate algorithms.

An object of class `"isf"` is essentially a list of functions for various tasks. The internal format is undocumented and may be changed without notice.

Value

An object of class `"isf"`, essentially a list of functions for various tasks.

The internal format is undocumented and may be changed without notice.

Author(s)

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

`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 <r.turner@auckland.ac.nz> 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)
```

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 <r.turner@auckland.ac.nz>

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")
```

`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 <r.turner@auckland.ac.nz>

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

```
data(cells)
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

```
# plot(Jinhom(swedishpines, sigma=bw.diggle, adjust=2))

plot(Jinhom(swedishpines, sigma=10))
```

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 <r.turner@auckland.ac.nz> 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)
```

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)
```

---

kaplan.meier *Kaplan-Meier Estimator using Histogram Data*

Description

Compute the Kaplan-Meier estimator of a survival time distribution function, from histogram data

Usage

```
kaplan.meier(obs, nco, breaks, upperobs=0)
```

Arguments

obs	vector of n integers giving the histogram of all observations (censored or uncensored survival times)
nco	vector of n integers giving the histogram of uncensored observations (those survival times that are less than or equal to the censoring time)
breaks	Vector of $n + 1$ breakpoints which were used to form both histograms.
upperobs	Number of observations beyond the rightmost breakpoint, if any.

Details

This function is needed mainly for internal use in **spatstat**, but may be useful in other applications where you want to form the Kaplan-Meier estimator from a huge dataset.

Suppose T_i are the survival times of individuals $i = 1, \ldots, M$ with unknown distribution function $F(t)$ which we wish to estimate. Suppose these times are right-censored by random censoring times C_i. Thus the observations consist of right-censored survival times $\tilde{T}_i = \min(T_i, C_i)$ and non-censoring indicators $D_i = 1\{T_i \le C_i\}$ for each i.

If the number of observations M is large, it is efficient to use histograms. Form the histogram `obs` of all observed times $\tilde{T}_i$. That is, `obs[k]` counts the number of values $\tilde{T}_i$ in the interval `(breaks[k],breaks[k+1]]` for $k > 1$ and `[breaks[1],breaks[2]]` for $k = 1$. Also form the histogram `nco` of all uncensored times, i.e. those $\tilde{T}_i$ such that $D_i = 1$. These two histograms are the arguments passed to `kaplan.meier`.

The vectors `km` and `lambda` returned by `kaplan.meier` are (histogram approximations to) the Kaplan-Meier estimator of $F(t)$ and its hazard rate $\lambda(t)$. Specifically, `km[k]` is an estimate of `F(breaks[k+1])`, and `lambda[k]` is an estimate of the average of $\lambda(t)$ over the interval `(breaks[k],breaks[k+1])`.

The histogram breaks must include 0. If the histogram breaks do not span the range of the observations, it is important to count how many survival times $\tilde{T}_i$ exceed the rightmost breakpoint, and give this as the value `upperobs`.

Value

A list with two elements:

`km`	Kaplan-Meier estimate of the survival time c.d.f. $F(t)$
`lambda`	corresponding Nelson-Aalen estimate of the hazard rate $\lambda(t)$

These are numeric vectors of length n.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`reduced.sample`, `km.rs`

Kcom *Model Compensator of K Function*

Description

Given a point process model fitted to a point pattern dataset, this function computes the *compensator* of the K function based on the fitted model (as well as the usual nonparametric estimates of K based on the data alone). Comparison between the nonparametric and model-compensated K functions serves as a diagnostic for the model.

Usage

```
Kcom(object, r = NULL, breaks = NULL, ...,
     correction = c("border", "isotropic", "translate"),
     conditional = !is.poisson(object),
     restrict = FALSE,
     model = NULL,
     trend = ~1, interaction = Poisson(), rbord = reach(interaction),
     compute.var = TRUE,
     truecoef = NULL, hi.res = NULL)
```

Arguments

object	Object to be analysed. Either a fitted point process model (object of class "ppm") or a point pattern (object of class "ppp") or quadrature scheme (object of class "quad").
r	Optional. Vector of values of the argument r at which the function $K(r)$ should be computed. This argument is usually not specified. There is a sensible default.
breaks	This argument is for advanced use only.
...	Ignored.
correction	Optional vector of character strings specifying the edge correction(s) to be used. See Kest for options.
conditional	Optional. Logical value indicating whether to compute the estimates for the conditional case. See Details.
restrict	Logical value indicating whether to compute the restriction estimator (restrict=TRUE) or the reweighting estimator (restrict=FALSE, the default). Applies only if conditional=TRUE. See Details.
model	Optional. A fitted point process model (object of class "ppm") to be re-fitted to the data using update.ppm, if object is a point pattern. Overrides the arguments trend,interaction,rbord.
trend,interaction,rbord	Optional. Arguments passed to ppm to fit a point process model to the data, if object is a point pattern. See ppm for details.
compute.var	Logical value indicating whether to compute the Poincare variance bound for the residual K function (calculation is only implemented for the isotropic correction).
truecoef	Optional. Numeric vector. If present, this will be treated as if it were the true coefficient vector of the point process model, in calculating the diagnostic. Incompatible with hi.res.
hi.res	Optional. List of parameters passed to quadscheme. If this argument is present, the model will be re-fitted at high resolution as specified by these parameters. The coefficients of the resulting fitted model will be taken as the true coefficients. Then the diagnostic will be computed for the default quadrature scheme, but using the high resolution coefficients.

Details

This command provides a diagnostic for the goodness-of-fit of a point process model fitted to a point pattern dataset. It computes an estimate of the K function of the dataset, together with a *model compensator* of the K function, which should be approximately equal if the model is a good fit to the data.

The first argument, `object`, is usually a fitted point process model (object of class `"ppm"`), obtained from the model-fitting function `ppm`.

For convenience, `object` can also be a point pattern (object of class `"ppp"`). In that case, a point process model will be fitted to it, by calling `ppm` using the arguments `trend` (for the first order trend), `interaction` (for the interpoint interaction) and `rbord` (for the erosion distance in the border correction for the pseudolikelihood). See `ppm` for details of these arguments.

The algorithm first extracts the original point pattern dataset (to which the model was fitted) and computes the standard nonparametric estimates of the K function. It then also computes the *model compensator* of the K function. The different function estimates are returned as columns in a data frame (of class `"fv"`).

The argument `correction` determines the edge correction(s) to be applied. See `Kest` for explanation of the principle of edge corrections. The following table gives the options for the `correction` argument, and the corresponding column names in the result:

`correction`	**description of correction**	**nonparametric**	**compensator**
`"isotropic"`	Ripley isotropic correction	`iso`	`icom`
`"translate"`	Ohser-Stoyan translation correction	`trans`	`tcom`
`"border"`	border correction	`border`	`bcom`

The nonparametric estimates can all be expressed in the form

$$\hat{K}(r) = \sum_i \sum_{j<i} e(x_i, x_j, r, x) I\{d(x_i, x_j) \le r\}$$

where x_i is the i-th data point, $d(x_i, x_j)$ is the distance between x_i and x_j, and $e(x_i, x_j, r, x)$ is a term that serves to correct edge effects and to re-normalise the sum. The corresponding model compensator is

$$\mathbf{C}\,\tilde{K}(r) = \int_W \lambda(u, x) \sum_j e(u, x_j, r, x \cup u) I\{d(u, x_j) \le r\}$$

where the integral is over all locations u in the observation window, $\lambda(u, x)$ denotes the conditional intensity of the model at the location u, and $x \cup u$ denotes the data point pattern x augmented by adding the extra point u.

If the fitted model is a Poisson point process, then the formulae above are exactly what is computed. If the fitted model is not Poisson, the formulae above are modified slightly to handle edge effects.

The modification is determined by the arguments `conditional` and `restrict`. The value of `conditional` defaults to `FALSE` for Poisson models and `TRUE` for non-Poisson models. If `conditional=FALSE` then the formulae above are not modified. If `conditional=TRUE`, then the algorithm calculates the *restriction estimator* if `restrict=TRUE`, and calculates the *reweighting estimator* if `restrict=FALSE`. See Appendix D of Baddeley, Rubak and Møller (2011). Thus, by default, the reweighting estimator is computed for non-Poisson models.

The nonparametric estimates of $K(r)$ are approximately unbiased estimates of the K-function, assuming the point process is stationary. The model compensators are unbiased estimates *of the mean values of the corresponding nonparametric estimates*, assuming the model is true. Thus, if the model is a good fit, the mean value of the difference between the nonparametric estimates and model compensators is approximately zero.

Value

A function value table (object of class `"fv"`), essentially a data frame of function values. There is a plot method for this class. See `fv.object`.

Author(s)

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

References

Baddeley, A., Rubak, E. and Møller, J. (2011) Score, pseudo-score and residual diagnostics for spatial point process models. *Statistical Science* **26**, 613–646.

See Also

Related functions: `Kres`, `Kest`.

Alternative functions: `Gcom`, `psstG`, `psstA`, `psst`.

Point process models: `ppm`.

Examples

```
    fit0 <- ppm(cells, ~1) # uniform Poisson


    if(interactive()) {
      plot(Kcom(fit0))
# compare the isotropic-correction estimates
      plot(Kcom(fit0), cbind(iso, icom) ~ r)
# uniform Poisson is clearly not correct
    }

    fit1 <- ppm(cells, ~1, Strauss(0.08))

    K1 <- Kcom(fit1)
    K1
    if(interactive()) {
      plot(K1)
      plot(K1, cbind(iso, icom) ~ r)
      plot(K1, cbind(trans, tcom) ~ r)
# how to plot the difference between nonparametric estimates and compensators
      plot(K1, iso - icom ~ r)
# fit looks approximately OK; try adjusting interaction distance
    }
```

```
fit2 <- ppm(cells, ~1, Strauss(0.12))

K2 <- Kcom(fit2)
if(interactive()) {
  plot(K2)
  plot(K2, cbind(iso, icom) ~ r)
  plot(K2, iso - icom ~ r)
}
```

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", "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 <r.turner@auckland.ac.nz>

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)



## K10 <- Kcross(amacrine, "on", "off")

# 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 <r.turner@auckland.ac.nz> 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
fit <- ppm(woods ~marks * polynom(x,y,2))
# alternative (a): use fitted model as 'lambda' argument
K <- Kcross.inhom(woods, "whiteoak", "maple",
                  lambdaI=fit, lambdaJ=fit, update=FALSE)
K <- Kcross.inhom(woods, "whiteoak", "maple",
                  lambdaX=fit, update=FALSE)
# alternative (b): 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))
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", "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 <r.turner@auckland.ac.nz>

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 cross 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 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 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 `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 $K_{i\bullet}(r)$ 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 <r.turner@auckland.ac.nz> 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
fit <- ppm(woods ~ marks * polynom(x,y,2))
K <- Kdot.inhom(woods, "maple", lambdaX=fit, update=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)
```

---

kernel.factor *Scale factor for density kernel*

Description

Returns a scale factor for the kernels used in density estimation for numerical data.

Usage

```
kernel.factor(kernel = "gaussian")
```

Arguments

kernel	String name of the kernel. Options are "gaussian", "rectangular", "triangular", "epanechnikov", "biweight", "cosine" and "optcosine". (Partial matching is used).

Details

Kernel estimation of a probability density in one dimension is performed by density.default using a kernel function selected from the list above.

This function computes a scale constant for the kernel. For the Gaussian kernel, this constant is equal to 1. Otherwise, the constant c is such that the kernel with standard deviation 1 is supported on the interval $[-c, c]$.

For more information about these kernels, see `density.default`.

Value

A single number.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Martin Hazelton

See Also

`density.default`, `dkernel`, `kernel.moment`, `kernel.squint`

Examples

```
kernel.factor("rect")
# bandwidth for Epanechnikov kernel with half-width h=1
h <- 1
bw <- h/kernel.factor("epa")
```

`kernel.moment` *Moment of Smoothing Kernel*

Description

Computes the complete or incomplete mth moment of a smoothing kernel.

Usage

```
kernel.moment(m, r, kernel = "gaussian")
```

Arguments

`m`	Exponent (order of moment). An integer.
`r`	Upper limit of integration for the incomplete moment. A numeric value or numeric vector. Set `r=Inf` to obtain the complete moment.
`kernel`	String name of the kernel. Options are `"gaussian"`, `"rectangular"`, `"triangular"`, `"epanechnikov"`, `"biweight"`, `"cosine"` and `"optcosine"`. (Partial matching is used).

Details

Kernel estimation of a probability density in one dimension is performed by density.default using a kernel function selected from the list above. For more information about these kernels, see density.default.

The function kernel.moment computes the partial integral

$$\int_{-\infty}^{r} t^m k(t) dt$$

where $k(t)$ is the selected kernel, r is the upper limit of integration, and m is the exponent or order. Here $k(t)$ is the **standard form** of the kernel, which has support $[-1, 1]$ and standard deviation $sigma = 1/c$ where c = kernel.factor(kernel).

Value

A single number, or a numeric vector of the same length as r.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Martin Hazelton.

See Also

density.default, dkernel, kernel.factor,

Examples

```
kernel.moment(1, 0.1, "epa")
curve(kernel.moment(2, x, "epa"), from=-1, to=1)
```

---

kernel.squint *Integral of Squared Kernel*

Description

Computes the integral of the squared kernel, for the kernels used in density estimation for numerical data.

Usage

```
kernel.squint(kernel = "gaussian", bw=1)
```

Arguments

kernel	String name of the kernel. Options are "gaussian", "rectangular", "triangular", "epanechnikov", "biweight", "cosine" and "optcosine". (Partial matching is used).
bw	Bandwidth (standard deviation) of the kernel.

Details

Kernel estimation of a probability density in one dimension is performed by density.default using a kernel function selected from the list above.

This function computes the integral of the squared kernel,

$$R = \int_{-\infty}^{\infty} k(x)^2 \, \mathrm{d}x$$

where $k(x)$ is the kernel with bandwidth bw.

Value

A single number.

Author(s)

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

See Also

density.default, dkernel, kernel.moment, kernel.factor

Examples

```
kernel.squint("gaussian", 3)

# integral of squared Epanechnikov kernel with half-width h=1
h <- 1
bw <- h/kernel.factor("epa")
kernel.squint("epa", bw)
```

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", "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).

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 <r.turner@auckland.ac.nz>

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 <r.turner@auckland.ac.nz> 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 <r.turner@auckland.ac.nz>

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)


# (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)
}
```

`km.rs` *Kaplan-Meier and Reduced Sample Estimator using Histograms*

Description

Compute the Kaplan-Meier and Reduced Sample estimators of a survival time distribution function, using histogram techniques

Usage

```
km.rs(o, cc, d, breaks)
```

Arguments

`o`	vector of observed survival times
`cc`	vector of censoring times
`d`	vector of non-censoring indicators
`breaks`	Vector of breakpoints to be used to form histograms.

Details

This function is needed mainly for internal use in **spatstat**, but may be useful in other applications where you want to form the Kaplan-Meier estimator from a huge dataset.

Suppose T_i are the survival times of individuals $i = 1, \ldots, M$ with unknown distribution function $F(t)$ which we wish to estimate. Suppose these times are right-censored by random censoring times C_i. Thus the observations consist of right-censored survival times $\tilde{T}_i = \min(T_i, C_i)$ and non-censoring indicators $D_i = 1\{T_i \le C_i\}$ for each i.

The arguments to this function are vectors `o`, `cc`, `d` of observed values of $\tilde{T}_i$, C_i and D_i respectively. The function computes histograms and forms the reduced-sample and Kaplan-Meier estimates of $F(t)$ by invoking the functions `kaplan.meier` and `reduced.sample`. This is efficient if the lengths of `o`, `cc`, `d` (i.e. the number of observations) is large.

The vectors km and hazard returned by kaplan.meier are (histogram approximations to) the Kaplan-Meier estimator of $F(t)$ and its hazard rate $\lambda(t)$. Specifically, km[k] is an estimate of F(breaks[k+1]), and lambda[k] is an estimate of the average of $\lambda(t)$ over the interval (breaks[k],breaks[k+1]). This approximation is exact only if the survival times are discrete and the histogram breaks are fine enough to ensure that each interval (breaks[k],breaks[k+1]) contains only one possible value of the survival time.

The vector rs is the reduced-sample estimator, rs[k] being the reduced sample estimate of F(breaks[k+1]). This value is exact, i.e. the use of histograms does not introduce any approximation error in the reduced-sample estimator.

Value

A list with five elements

rs	Reduced-sample estimate of the survival time c.d.f. $F(t)$
km	Kaplan-Meier estimate of the survival time c.d.f. $F(t)$
hazard	corresponding Nelson-Aalen estimate of the hazard rate $\lambda(t)$
r	values of t for which $F(t)$ is estimated
breaks	the breakpoints vector

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

reduced.sample, kaplan.meier

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

$$C_f(r) = E\left[\sum_{x \in X} f(m(u), m(x)) 10 < ||u - x|| \leq r \;\middle|\; u \in X\right]$$

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 <r.turner@auckland.ac.nz>

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 <r.turner@auckland.ac.nz>

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])
```

Kmodel *K Function or Pair Correlation Function of a Point Process Model*

Description

Returns the theoretical K function or the pair correlation function of a point process model.

Usage

```
Kmodel(model, ...)

pcfmodel(model, ...)
```

Arguments

model	A fitted point process model of some kind.
...	Ignored.

Details

For certain types of point process models, it is possible to write down a mathematical expression for the K function or the pair correlation function of the model.

The functions Kmodel and pcfmodel give the theoretical K-function and the theoretical pair correlation function for a point process model that has been fitted to data.

The functions Kmodel and pcfmodel are generic, with methods for the classes "kppm" (cluster processes and Cox processes) and "ppm" (Gibbs processes).

The return value is a function in the R language, which takes one argument r. Evaluation of this function, on a numeric vector r, yields values of the desired K function or pair correlation function at these distance values.

Value

A function in the R language, which takes one argument r.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

Kest or pcf to estimate the K function or pair correlation function nonparametrically from data.

Kmodel.kppm for the method for cluster processes and Cox processes.

Kmodel.ppm for the method for Gibbs processes.

`Kmodel.dppm` *K-function or Pair Correlation Function of a Determinantal Point Process Model*

Description

Returns the theoretical K-function or theoretical pair correlation function of a determinantal point process model as a function of one argument r.

Usage

```
   ## S3 method for class 'dppm'
Kmodel(model, ...)

   ## S3 method for class 'dppm'
pcfmodel(model, ...)

   ## S3 method for class 'detpointprocfamily'
Kmodel(model, ...)

   ## S3 method for class 'detpointprocfamily'
pcfmodel(model, ...)
```

Arguments

`model`	Model of class "`detpointprocfamily`" or "`dppm`".
`...`	Ignored (not quite true – there is some undocumented internal use)

Value

A function in the R language, with one numeric argument `r`, that can be used to evaluate the theoretical K-function or pair correlation function of the model at distances `r`.

Author(s)

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

Examples

```
model <- dppMatern(lambda=100, alpha=.01, nu=1, d=2)
KMatern <- Kmodel(model)
pcfMatern <- pcfmodel(model)
plot(KMatern, xlim = c(0,0.05))
plot(pcfMatern, xlim = c(0,0.05))
```

`Kmodel.kppm` *K Function or Pair Correlation Function of Cluster Model or Cox model*

Description

Returns the theoretical K function or the pair correlation function of a cluster point process model or Cox point process model.

Usage

```
   ## S3 method for class 'kppm'
Kmodel(model, ...)

   ## S3 method for class 'kppm'
pcfmodel(model, ...)
```

Arguments

`model`	A fitted cluster point process model (object of class `"kppm"`) typically obtained from the model-fitting algorithm `kppm`.
`...`	Ignored.

Details

For certain types of point process models, it is possible to write down a mathematical expression for the K function or the pair correlation function of the model. In particular this is possible for a fitted cluster point process model (object of class `"kppm"` obtained from `kppm`).

The functions `Kmodel` and `pcfmodel` are generic. The functions documented here are the methods for the class `"kppm"`.

The return value is a `function` in the R language, which takes one argument `r`. Evaluation of this function, on a numeric vector `r`, yields values of the desired K function or pair correlation function at these distance values.

Value

A `function` in the R language, which takes one argument `r`.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

Kest or pcf to estimate the K function or pair correlation function nonparametrically from data.

kppm to fit cluster models.

Kmodel for the generic functions.

Kmodel.ppm for the method for Gibbs processes.

Examples

```
data(redwood)
fit <- kppm(redwood, ~x, "MatClust")
K <- Kmodel(fit)
K(c(0.1, 0.2))
curve(K(x), from=0, to=0.25)
```

Kmodel.ppm *K Function or Pair Correlation Function of Gibbs Point Process model*

Description

Returns the theoretical K function or the pair correlation function of a fitted Gibbs point process model.

Usage

```
   ## S3 method for class 'ppm'
Kmodel(model, ...)

   ## S3 method for class 'ppm'
pcfmodel(model, ...)
```

Arguments

model	A fitted Poisson or Gibbs point process model (object of class "ppm") typically obtained from the model-fitting algorithm ppm.
...	Ignored.

Details

This function computes an *approximation* to the K function or the pair correlation function of a Gibbs point process.

The functions Kmodel and pcfmodel are generic. The functions documented here are the methods for the class "ppm".

The approximation is only available for stationary pairwise-interaction models. It uses the second order Poisson-saddlepoint approximation (Baddeley and Nair, 2012b) which is a combination of the Poisson-Boltzmann-Emden and Percus-Yevick approximations.

The return value is a function in the R language, which takes one argument r. Evaluation of this function, on a numeric vector r, yields values of the desired K function or pair correlation function at these distance values.

Value

A function in the R language, which takes one argument r.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Gopalan Nair.

References

Baddeley, A. and Nair, G. (2012a) Fast approximation of the intensity of Gibbs point processes. *Electronic Journal of Statistics* **6** 1155–1169.

Baddeley, A. and Nair, G. (2012b) Approximating the moments of a spatial point process. *Stat* **1**, 1, 18–30. DOI: 10.1002/sta4.5

See Also

Kest or pcf to estimate the K function or pair correlation function nonparametrically from data.

ppm to fit Gibbs models.

Kmodel for the generic functions.

Kmodel.kppm for the method for cluster/Cox processes.

Examples

```
fit <- ppm(swedishpines, ~1, Strauss(8))
p <- pcfmodel(fit)
K <- Kmodel(fit)
p(6)
K(8)
curve(K(x), from=0, to=15)
```

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"`, `"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 <r.turner@auckland.ac.nz> 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 sub-processes, 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 <r.turner@auckland.ac.nz>

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)
```

---

kppm *Fit Cluster or Cox Point Process Model*

Description

Fit a homogeneous or inhomogeneous cluster process or Cox point process model to a point pattern.

Usage

```
  kppm(X, ...)

  ## S3 method for class 'formula'
kppm(X,
                clusters = c("Thomas","MatClust","Cauchy","VarGamma","LGCP"),
                ...,
                data=NULL)

  ## S3 method for class 'ppp'
kppm(X,
       trend = ~1,
       clusters = c("Thomas","MatClust","Cauchy","VarGamma","LGCP"),
       data = NULL,
       ...,
       covariates=data,
       subset,
       method = c("mincon", "clik2", "palm", "adapcl"),
       improve.type = c("none", "clik1", "wclik1", "quasi"),
       improve.args = list(),
       weightfun=NULL,
       control=list(),
       stabilize=TRUE,
       algorithm,
       statistic="K",
       statargs=list(),
       rmax = NULL,
       epsilon=0.01,
       covfunargs=NULL,
       use.gam=FALSE,
       nd=NULL, eps=NULL)

## S3 method for class 'quad'
kppm(X,
       trend = ~1,
       clusters = c("Thomas","MatClust","Cauchy","VarGamma","LGCP"),
```

```
        data = NULL,
        ...,
        covariates=data,
        subset,
        method = c("mincon", "clik2", "palm", "adapcl"),
        improve.type = c("none", "clik1", "wclik1", "quasi"),
        improve.args = list(),
        weightfun=NULL,
        control=list(),
        stabilize=TRUE,
        algorithm,
        statistic="K",
        statargs=list(),
        rmax = NULL,
        epsilon=0.01,
        covfunargs=NULL,
        use.gam=FALSE,
        nd=NULL, eps=NULL)
```

Arguments

X	A point pattern dataset (object of class "ppp" or "quad") to which the model should be fitted, or a formula in the R language defining the model. See Details.
trend	An R formula, with no left hand side, specifying the form of the log intensity.
clusters	Character string determining the cluster model. Partially matched. Options are "Thomas", "MatClust", "Cauchy", "VarGamma" and "LGCP".
data,covariates	The values of spatial covariates (other than the Cartesian coordinates) required by the model. A named list of pixel images, functions, windows, tessellations or numeric constants.
...	Additional arguments. See Details.
subset	Optional. A subset of the spatial domain, to which the model-fitting should be restricted. A window (object of class "owin") or a logical-valued pixel image (object of class "im"), or an expression (possibly involving the names of entries in data) which can be evaluated to yield a window or pixel image.
method	The fitting method. Either "mincon" for minimum contrast, "clik2" for second order composite likelihood, "adapcl" for adaptive second order composite likelihood, or "palm" for Palm likelihood. Partially matched.
improve.type	Method for updating the initial estimate of the trend. Initially the trend is estimated as if the process is an inhomogeneous Poisson process. The default, improve.type = "none", is to use this initial estimate. Otherwise, the trend estimate is updated by improve.kppm, using information about the pair correlation function. Options are "clik1" (first order composite likelihood, essentially equivalent to "none"), "wclik1" (weighted first order composite likelihood) and "quasi" (quasi likelihood).
improve.args	Additional arguments passed to improve.kppm when improve.type != "none". See Details.

weightfun	Optional weighting function w in the composite likelihoods or Palm likelihood. A function in the R language. See Details.
control	List of control parameters passed to the optimization function optim.
stabilize	Logical value specifying whether to numerically stabilize the optimization algorithm, by specifying suitable default values of control$fnscale and control$parscale.
algorithm	Character string determining the mathematical algorithm to be used to solve the fitting problem. If method="mincon", "clik2" or "palm" this argument is passed to the generic optimization function optim (renamed as the argument method to optim) with default "Nelder-Mead". If method="adapcl") the argument is passed to the equation solver nleqslv (renamed as the argument method to nleqslv) with default "Bryden".
statistic	Name of the summary statistic to be used for minimum contrast estimation: either "K" or "pcf".
statargs	Optional list of arguments to be used when calculating the statistic. See Details.
rmax	Maximum value of interpoint distance to use in the composite likelihood.
epsilon	Tuning parameter for the adaptive composite likelihood method.
covfunargs,use.gam,nd,eps	Arguments passed to ppm when fitting the intensity.

Details

This function fits a clustered point process model to the point pattern dataset X.

The model may be either a *Neyman-Scott cluster process* or another *Cox process*. The type of model is determined by the argument clusters. Currently the options are clusters="Thomas" for the Thomas process, clusters="MatClust" for the Matérn cluster process, clusters="Cauchy" for the Neyman-Scott cluster process with Cauchy kernel, clusters="VarGamma" for the Neyman-Scott cluster process with Variance Gamma kernel (requires an additional argument nu to be passed through the dots; see rVarGamma for details), and clusters="LGCP" for the log-Gaussian Cox process (may require additional arguments passed through ...; see rLGCP for details on argument names). The first four models are Neyman-Scott cluster processes.

The algorithm first estimates the intensity function of the point process using ppm. The argument X may be a point pattern (object of class "ppp") or a quadrature scheme (object of class "quad"). The intensity is specified by the trend argument. If the trend formula is ~1 (the default) then the model is *homogeneous*. The algorithm begins by estimating the intensity as the number of points divided by the area of the window. Otherwise, the model is *inhomogeneous*. The algorithm begins by fitting a Poisson process with log intensity of the form specified by the formula trend. (See ppm for further explanation).

The argument X may also be a formula in the R language. The right hand side of the formula gives the trend as described above. The left hand side of the formula gives the point pattern dataset to which the model should be fitted.

If improve.type="none" this is the final estimate of the intensity. Otherwise, the intensity estimate is updated, as explained in improve.kppm. Additional arguments to improve.kppm are passed as a named list in improve.args.

The cluster parameters of the model are then fitted either by minimum contrast estimation, or by a composite likelihood method (maximum composite likelihood, maximum Palm likelihood, or by solving the adaptive composite likelihood estimating equation).

Minimum contrast: If `method = "mincon"` (the default) clustering parameters of the model will be fitted by minimum contrast estimation, that is, by matching the theoretical K-function of the model to the empirical K-function of the data, as explained in `mincontrast`.

For a homogeneous model (`trend = ~1`) the empirical K-function of the data is computed using `Kest`, and the parameters of the cluster model are estimated by the method of minimum contrast.

For an inhomogeneous model, the inhomogeneous K function is estimated by `Kinhom` using the fitted intensity. Then the parameters of the cluster model are estimated by the method of minimum contrast using the inhomogeneous K function. This two-step estimation procedure is due to Waagepetersen (2007).

If `statistic="pcf"` then instead of using the K-function, the algorithm will use the pair correlation function `pcf` for homogeneous models and the inhomogeneous pair correlation function `pcfinhom` for inhomogeneous models. In this case, the smoothing parameters of the pair correlation can be controlled using the argument `statargs`, as shown in the Examples.

Additional arguments `...` will be passed to `clusterfit` to control the minimum contrast fitting algorithm.

The optimisation is performed by the generic optimisation algorithm `optim`.

Second order composite likelihood: If `method = "clik2"` the clustering parameters of the model will be fitted by maximising the second-order composite likelihood (Guan, 2006). The log composite likelihood is

$$\sum_{i,j} w(d_{ij}) \log \rho(d_{ij};\theta) - \left(\sum_{i,j} w(d_{ij}) \right) \log \int_D \int_D w(\|u-v\|)\rho(\|u-v\|;\theta)\, du\, dv$$

where the sums are taken over all pairs of data points x_i, x_j separated by a distance $d_{ij} = \|x_i - x_j\|$ less than `rmax`, and the double integral is taken over all pairs of locations u, v in the spatial window of the data. Here $\rho(d;\theta)$ is the pair correlation function of the model with cluster parameters θ.

The function w in the composite likelihood is a weighting function and may be chosen arbitrarily. It is specified by the argument `weightfun`. If this is missing or `NULL` then the default is a threshold weight function, $w(d) = 1(d \le R)$, where R is `rmax/2`.

The optimisation is performed by the generic optimisation algorithm `optim`.

Palm likelihood: If `method = "palm"` the clustering parameters of the model will be fitted by maximising the Palm loglikelihood (Tanaka et al, 2008)

$$\sum_{i,j} w(x_i, x_j) \log \lambda_P(x_j \mid x_i;\theta) - \int_D w(x_i,u)\lambda_P(u \mid x_i;\theta)\mathrm{d}u$$

with the same notation as above. Here $\lambda_P(u|v;\theta$ is the Palm intensity of the model at location u given there is a point at v.

The optimisation is performed by the generic optimisation algorithm `optim`.

Adaptive Composite likelihood: If `method = "cladap"` the clustering parameters of the model will be fitted by solving the adaptive second order composite likelihood estimating equation (Lavancier et al, 2021). The estimating function is

$$\sum_{u,v} w(\epsilon \frac{|g(0;\theta)-1|}{g(\|u-v\|;\theta)-1}) \frac{\nabla_\theta g(\|u-v\|;\theta)}{g(\|u-v\|;\theta)} - \int_D \int_D w(\epsilon \frac{M(u,v;\theta)}{\nabla_\theta g(\|u-v\|;\theta)}) \rho(u)\rho(v)\, du\, dv$$

where the sum is taken over all distinct pairs of points. Here $g(d;\theta)$ is the pair correlation function with parameters θ. The partial derivative with respect to θ is $g'(d;\theta)$, and $\rho(u)$ denotes the fitted intensity function of the model.

The tuning parameter ϵ is independent of the data. It can be specified by the argument `epsilon` and has default value 0.01.

The function w in the estimating function is a weighting function of bounded support $[-1,1]$. It is specified by the argument `weightfun`. If this is missing or `NULL` then the default is $w(d) = 1(\|d\| \le 1)\exp(1/(r^2-1))$. The estimating equation is solved using the nonlinear equation solver `nleqslv` from the package **nleqslv**. The package **nleqslv** must be installed in order to use this option.

Fitting the LGCP model requires the **RandomFields** package, except in the default case where the exponential covariance is assumed.

Value

An object of class `"kppm"` representing the fitted model. There are methods for printing, plotting, predicting, simulating and updating objects of this class.

Optimization algorithm

The following details allow greater control over the fitting procedure.

For the first three fitting methods (`method="mincon"`, `"clik2"` and `"palm"`), the optimisation is performed by the generic optimisation algorithm `optim`. The behaviour of this algorithm can be controlled by the following arguments to `kppm`:

- `startpar` determines the initial estimates of the cluster parameters.
- `algorithm` determines the particular optimization method. This argument is passed to `optim` as the argument `method`. Options are listed in the help for `optim`. The default is the Nelder-Mead simplex method.
- `control` is a named list of control parameters, documented in the help for `optim`. Useful control arguments include `trace`, `maxit` and `abstol`.
- `lower` and `upper` specify bounds for the cluster parameters, when `algorithm="L-BFGS-B"` or `algorithm="Brent"`, as described in the help for `optim`.

For `method="adapcl"`, the estimating equation is solved using the nonlinear equation solver `nleqslv` from the package **nleqslv**. The package **nleqslv** must be installed in order to use this option. The behaviour of this algorithm can be controlled by the following arguments to `kppm`:

- `startpar` determines the initial estimates of the cluster parameters.
- `algorithm` determines the method for solving the equation. This argument is passed to `nleqslv` as the argument `method`. Options are listed in the help for `nleqslv`.

- `globStrat` determines the global strategy to be applied. This argument is is passed to `nleqslv` as the argument `global`. Options are listed in the help for `nleqslv`.
- `control` is a named list of control parameters, documented in the help for `nleqslv`.

Log-Gaussian Cox Models

To fit a log-Gaussian Cox model with non-exponential covariance, specify `clusters="LGCP"` and use additional arguments to specify the covariance structure. These additional arguments can be given individually in the call to `kppm`, or they can be collected together in a list called `covmodel`.

For example a Matérn model with parameter $\nu = 0.5$ could be specified either by `kppm(X, clusters="LGCP", model="matern", nu=0.5)` or by `kppm(X, clusters="LGCP", covmodel=list(model="matern", nu=0.5))`.

The argument `model` specifies the type of covariance model: the default is `model="exp"` for an exponential covariance. Alternatives include `"matern"`, `"cauchy"` and `"spheric"`. Model names correspond to functions beginning with `RM` in the **RandomFields** package: for example `model="matern"` corresponds to the function `RMmatern` in the **RandomFields** package.

Additional arguments are passed to the relevant function in the **RandomFields** package: for example if `model="matern"` then the additional argument `nu` is required, and is passed to the function `RMmatern` in the **RandomFields** package.

Note that it is not possible to use *anisotropic* covariance models because the `kppm` technique assumes the pair correlation function is isotropic.

Error and warning messages

See `ppm.ppp` for a list of common error messages and warnings originating from the first stage of model-fitting.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>, with contributions from Abdollah Jalilian and Rasmus Waagepetersen. Adaptive composite likelihood method contributed by Chiara Fend and modified by Adrian Baddeley.

References

Guan, Y. (2006) A composite likelihood approach in fitting spatial point process models. *Journal of the American Statistical Association* **101**, 1502–1512.

Guan, Y., Jalilian, A. and Waagepetersen, R. (2015) Quasi-likelihood for spatial point processes. *Journal of the Royal Statistical Society, Series B* **77**, 677-697.

Jalilian, A., Guan, Y. and Waagepetersen, R. (2012) Decomposition of variance for spatial Cox processes. *Scandinavian Journal of Statistics* **40**, 119–137.

Lavancier, F., Poinas, A., and Waagepetersen, R. (2021) Adaptive estimating function inference for nonstationary determinantal point processes. *Scandinavian Journal of Statistics*, **48** (1), 87–107.

Tanaka, U. and Ogata, Y. and Stoyan, D. (2008) Parameter estimation and model selection for Neyman-Scott point processes. *Biometrical Journal* **50**, 43–57.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

Methods for kppm objects: plot.kppm, fitted.kppm, predict.kppm, simulate.kppm, update.kppm, vcov.kppm, methods.kppm, as.ppm.kppm, as.fv.kppm, Kmodel.kppm, pcfmodel.kppm.

See also improve.kppm for improving the fit of a kppm object.

Minimum contrast fitting algorithm: higher level interface clusterfit; low-level algorithm mincontrast.

Alternative fitting algorithms: thomas.estK, matclust.estK, lgcp.estK, cauchy.estK, vargamma.estK, thomas.estpcf, matclust.estpcf, lgcp.estpcf, cauchy.estpcf, vargamma.estpcf.

Summary statistics: Kest, Kinhom, pcf, pcfinhom.

For fitting Poisson or Gibbs point process models, see ppm.

Examples

```
online <- interactive()
if(!online) op <- spatstat.options(npixel=32, ndummy.min=16)

# method for point patterns
kppm(redwood, ~1, "Thomas")
# method for formulas
kppm(redwood ~ 1, "Thomas")

# different models for clustering
if(online) kppm(redwood ~ x, "MatClust")
kppm(redwood ~ x, "MatClust", statistic="pcf", statargs=list(stoyan=0.2))
kppm(redwood ~ x, cluster="Cauchy", statistic="K")
kppm(redwood, cluster="VarGamma", nu = 0.5, statistic="pcf")

# log-Gaussian Cox process (LGCP) models
kppm(redwood ~ 1, "LGCP", statistic="pcf")
if(require("RandomFields")) {
  # Random Fields package is needed for non-default choice of covariance model
  kppm(redwood ~ x, "LGCP", statistic="pcf",
                            model="matern", nu=0.3,
                            control=list(maxit=10))
}

# Different fitting techniques
kppm(redwood ~ 1, "Thomas", method="c")
kppm(redwood ~ 1, "Thomas", method="p")
# quasi-likelihood improvement
kppm(redwood ~ x, "Thomas", improve.type = "quasi")

if(!online) spatstat.options(op)
```

Kres *Residual K Function*

Description

Given a point process model fitted to a point pattern dataset, this function computes the residual K function, which serves as a diagnostic for goodness-of-fit of the model.

Usage

```
Kres(object, ...)
```

Arguments

object	Object to be analysed. Either a fitted point process model (object of class "ppm"), a point pattern (object of class "ppp"), a quadrature scheme (object of class "quad"), or the value returned by a previous call to Kcom.
...	Arguments passed to Kcom.

Details

This command provides a diagnostic for the goodness-of-fit of a point process model fitted to a point pattern dataset. It computes a residual version of the K function of the dataset, which should be approximately zero if the model is a good fit to the data.

In normal use, object is a fitted point process model or a point pattern. Then Kres first calls Kcom to compute both the nonparametric estimate of the K function and its model compensator. Then Kres computes the difference between them, which is the residual K-function.

Alternatively, object may be a function value table (object of class "fv") that was returned by a previous call to Kcom. Then Kres computes the residual from this object.

Value

A function value table (object of class "fv"), essentially a data frame of function values. There is a plot method for this class. See fv.object.

Author(s)

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

References

Baddeley, A., Rubak, E. and Møller, J. (2011) Score, pseudo-score and residual diagnostics for spatial point process models. *Statistical Science* **26**, 613–646.

See Also

Related functions: `Kcom`, `Kest`.

Alternative functions: `Gres`, `psstG`, `psstA`, `psst`.

Point process models: `ppm`.

Examples

```
    data(cells)
    fit0 <- ppm(cells, ~1) # uniform Poisson

    K0 <- Kres(fit0)
    K0
    plot(K0)
# isotropic-correction estimate
    plot(K0, ires ~ r)
# uniform Poisson is clearly not correct

    fit1 <- ppm(cells, ~1, Strauss(0.08))

    K1 <- Kres(fit1)

    if(interactive()) {
      plot(K1, ires ~ r)
   # fit looks approximately OK; try adjusting interaction distance
      plot(Kres(cells, interaction=Strauss(0.12)))
    }

# How to make envelopes
    # E <- envelope(fit1, Kres, model=fit1, nsim=19)
    # plot(E)

# For computational efficiency
    Kc <- Kcom(fit1)
    K1 <- Kres(Kc)
```

`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 <r.turner@auckland.ac.nz>

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

```
data(bronzefilter)
X <- unmark(bronzefilter)
K <- Kscaled(X)
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 <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

Kest

Examples

```
K <- Ksector(swedishpines, 0, 90)
plot(K)
```

LambertW *Lambert's W Function*

Description

Computes Lambert's W-function.

Usage

```
LambertW(x)
```

Arguments

x	Vector of nonnegative numbers.

Details

Lambert's W-function is the inverse function of $f(y) = ye^y$. That is, W is the function such that

$$W(x)e^{W(x)} = x$$

This command LambertW computes $W(x)$ for each entry in the argument x. If the library **gsl** has been installed, then the function lambert_W0 in that library is invoked. Otherwise, values of the W-function are computed by root-finding, using the function uniroot.

Computation using **gsl** is about 100 times faster.

If any entries of x are infinite or NA, the corresponding results are NA.

Value

Numeric vector.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

References

Corless, R, Gonnet, G, Hare, D, Jeffrey, D and Knuth, D (1996), On the Lambert W function. *Computational Mathematics*, **5**, 325–359.

Roy, R and Olver, F (2010), Lambert W function. In Olver, F, Lozier, D and Boisvert, R (eds.), *NIST Handbook of Mathematical Functions*, Cambridge University Press.

Examples

```
LambertW(exp(1))
```

`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 `<r.turner@auckland.ac.nz>`

See Also

`Kcross`, `Ldot`, `Lest`

Examples

```
data(amacrine)
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 <r.turner@auckland.ac.nz>

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
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 <r.turner@auckland.ac.nz>

See Also

Kdot, Lcross, Lest

Examples

```
data(amacrine)
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 <r.turner@auckland.ac.nz> 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.ppp(ma, sigma=0.15, at="points")
lambdadot <- density.ppp(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)
```

LennardJones *The Lennard-Jones Potential*

Description

Creates the Lennard-Jones pairwise interaction structure which can then be fitted to point pattern data.

Usage

```
LennardJones(sigma0=NA)
```

Arguments

sigma0	Optional. Initial estimate of the parameter σ. A positive number.

Details

In a pairwise interaction point process with the Lennard-Jones pair potential (Lennard-Jones, 1924) each pair of points in the point pattern, a distance d apart, contributes a factor

$$v(d) = \exp\left\{-4\epsilon\left[\left(\frac{\sigma}{d}\right)^{12} - \left(\frac{\sigma}{d}\right)^{6}\right]\right\}$$

to the probability density, where σ and ϵ are positive parameters to be estimated.

See **Examples** for a plot of this expression.

This potential causes very strong inhibition between points at short range, and attraction between points at medium range. The parameter σ is called the *characteristic diameter* and controls the scale of interaction. The parameter ϵ is called the *well depth* and determines the strength of attraction. The potential switches from inhibition to attraction at $d = \sigma$. The maximum value of the pair potential is $\exp(\epsilon)$ occuring at distance $d = 2^{1/6}\sigma$. Interaction is usually considered to be negligible for distances $d > 2.5\sigma \max\{1, \epsilon^{1/6}\}$.

This potential is used to model interactions between uncharged molecules in statistical physics.

The function ppm(), which fits point process models to point pattern data, requires an argument of class "interact" describing the interpoint interaction structure of the model to be fitted. The appropriate description of the Lennard-Jones pairwise interaction is yielded by the function LennardJones(). See the examples below.

Value

An object of class "interact" describing the Lennard-Jones interpoint interaction structure.

Rescaling

To avoid numerical instability, the interpoint distances d are rescaled when fitting the model.

Distances are rescaled by dividing by sigma0. In the formula for $v(d)$ above, the interpoint distance d will be replaced by d/sigma0.

The rescaling happens automatically by default. If the argument sigma0 is missing or NA (the default), then sigma0 is taken to be the minimum nearest-neighbour distance in the data point pattern (in the call to ppm).

If the argument sigma0 is given, it should be a positive number, and it should be a rough estimate of the parameter σ.

The "canonical regular parameters" estimated by ppm are $\theta_1 = 4\epsilon(\sigma/\sigma_0)^{12}$ and $\theta_2 = 4\epsilon(\sigma/\sigma_0)^6$.

Warnings and Errors

Fitting the Lennard-Jones model is extremely unstable, because of the strong dependence between the functions d^{-12} and d^{-6}. The fitting algorithm often fails to converge. Try increasing the number of iterations of the GLM fitting algorithm, by setting gcontrol=list(maxit=1e3) in the call to ppm.

Errors are likely to occur if this model is fitted to a point pattern dataset which does not exhibit both short-range inhibition and medium-range attraction between points. The values of the parameters σ and ϵ may be NA (because the fitted canonical parameters have opposite sign, which usually occurs when the pattern is completely random).

An absence of warnings does not mean that the fitted model is sensible. A negative value of ϵ may be obtained (usually when the pattern is strongly clustered); this does not correspond to a valid point process model, but the software does not issue a warning.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

References

Lennard-Jones, J.E. (1924) On the determination of molecular fields. *Proc Royal Soc London A* **106**, 463–477.

See Also

ppm, pairwise.family, ppm.object

Examples

```
badfit <- ppm(cells ~1, LennardJones(), rbord=0.1)
badfit

fit <- ppm(unmark(longleaf) ~1, LennardJones(), rbord=1)
fit
plot(fitin(fit))
```

```
# Note the Longleaf Pines coordinates are rounded to the nearest decimetre
# (multiple of 0.1 metres) so the apparent inhibition may be an artefact
```

`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 <r.turner@auckland.ac.nz>

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

```
data(cells)
L <- Lest(cells)
plot(L, main="L function for cells")
```

---

leverage.ppm *Leverage Measure for Spatial Point Process Model*

Description

Computes the leverage measure for a fitted spatial point process model.

Usage

```
leverage(model, ...)

## S3 method for class 'ppm'
leverage(model, ...,
         drop = FALSE, iScore=NULL, iHessian=NULL, iArgs=NULL)
```

Arguments

`model`	Fitted point process model (object of class `"ppm"`).
`...`	Ignored, except for the arguments `dimyx` and `eps` which are passed to `as.mask` to control the spatial resolution of the result.
`drop`	Logical. Whether to include (`drop=FALSE`) or exclude (`drop=TRUE`) contributions from quadrature points that were not used to fit the model.

`iScore,iHessian`
: Components of the score vector and Hessian matrix for the irregular parameters, if required. See Details.

`iArgs`
: List of extra arguments for the functions `iScore`, `iHessian` if required.

Details

The function `leverage` is generic, and `leverage.ppm` is the method for objects of class `"ppm"`.

Given a fitted spatial point process model `model`, the function `leverage.ppm` computes the leverage of the model, described in Baddeley, Chang and Song (2013) and Baddeley, Rubak and Turner (2019).

The leverage of a spatial point process model is a function of spatial location, and is typically displayed as a colour pixel image. The leverage value $h(u)$ at a spatial location u represents the change in the fitted trend of the fitted point process model that would have occurred if a data point were to have occurred at the location u. A relatively large value of $h()$ indicates a part of the space where the data have a *potentially* strong effect on the fitted model (specifically, a strong effect on the intensity or conditional intensity of the fitted model) due to the values of the covariates.

If the point process model trend has irregular parameters that were fitted (using `ippm`) then the leverage calculation requires the first and second derivatives of the log trend with respect to the irregular parameters. The argument `iScore` should be a list, with one entry for each irregular parameter, of R functions that compute the partial derivatives of the log trend (i.e. log intensity or log conditional intensity) with respect to each irregular parameter. The argument `iHessian` should be a list, with p^2 entries where p is the number of irregular parameters, of R functions that compute the second order partial derivatives of the log trend with respect to each pair of irregular parameters.

The result of `leverage.ppm` is an object of class `"leverage.ppm"`. It can be printed or plotted. It can be converted to a pixel image by `as.im` (see `as.im.leverage.ppm`). There are also methods for `contour`, `persp`, `[`, `as.function`, `as.owin`, `domain`, `Smooth`, `integral`, and `mean`.

Value

An object of class `"leverage.ppm"`.

Author(s)

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

References

Baddeley, A., Chang, Y.M. and Song, Y. (2013) Leverage and influence diagnostics for spatial point process models. *Scandinavian Journal of Statistics* **40**, 86–104.

Baddeley, A., Rubak, E. and Turner, R. (2019) Leverage and influence diagnostics for Gibbs spatial point processes. *Spatial Statistics* **29**, 15–48.

See Also

`influence.ppm`, `dfbetas.ppm`, `ppmInfluence`, `plot.leverage.ppm` `as.function.leverage.ppm`

Examples

```
if(offline <- !interactive()) op <- spatstat.options(npixel=32, ndummy.min=16)

   X <- rpoispp(function(x,y) { exp(3+3*x) })
   fit <- ppm(X ~x+y)
   le <- leverage(fit)
   if(!offline) plot(le)
   mean(le)

if(offline) spatstat.options(op)
```

`leverage.slrm` *Leverage and Influence Diagnostics for Spatial Logistic Regression*

Description

For a fitted spatial logistic regression model, these functions compute diagnostics of leverage and influence.

Usage

```
## S3 method for class 'slrm'
leverage(model, ...)
## S3 method for class 'slrm'
influence(model, ...)
## S3 method for class 'slrm'
dfbetas(model, ...)
## S3 method for class 'slrm'
dffit(object, ...)
```

Arguments

`model,object`	A fitted spatial logistic regression model (object of class `"slrm"`).
`...`	Arguments passed to methods.

Details

These functions are methods for the generics `leverage`, `influence`, `dfbetas` and `dffit` for the class `"slrm"`.

These functions adapt the standard diagnostics for logistic regression (see `influence.measures`) to a fitted spatial logistic regression model (object of class `"slrm"`). This adaptation was described by Baddeley, Chang and Song (2013).

`leverage.slrm` computes the leverage value (diagonal of the hat matrix) for the covariate data in each pixel. The result is a pixel image.

`influence.slrm` computes the likelihood influence for the data (covariates and presence/absence of points) in each pixel. The result is a pixel image.

dfbetas.slrm computes the parameter influence for the data (covariates and presence/absence of points) in each pixel. The result is a list of pixel images, one image for each of the model coefficients in coef(model). The list can be plotted immediately.

dffit.slrm computes the total influence for the data (covariates and presence/absence of points) in each pixel. The result is a pixel image.

Value

A pixel image, or a list of pixel images.

Author(s)

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

References

Baddeley, A., Chang, Y.M. and Song, Y. (2013) Leverage and influence diagnostics for spatial point process models. *Scandinavian Journal of Statistics* **40**, 86–104.

See Also

influence.measures.

leverage.ppm, influence.ppm, dfbetas.ppm, dffit.ppm

Examples

```
H <- unmark(humberside)
fit <- slrm(H ~ x+y, dimyx=32)
plot(leverage(fit))
plot(influence(fit))
plot(dfbetas(fit))
plot(dffit(fit))
```

---

lgcp.estK *Fit a Log-Gaussian Cox Point Process by Minimum Contrast*

Description

Fits a log-Gaussian Cox point process model to a point pattern dataset by the Method of Minimum Contrast.

Usage

```
lgcp.estK(X, startpar=c(var=1,scale=1),
             covmodel=list(model="exponential"),
             lambda=NULL,
             q = 1/4, p = 2, rmin = NULL, rmax = NULL, ...)
```

Arguments

X	Data to which the model will be fitted. Either a point pattern or a summary statistic. See Details.
startpar	Vector of starting values for the parameters of the log-Gaussian Cox process model.
covmodel	Specification of the covariance model for the log-Gaussian field. See Details.
lambda	Optional. An estimate of the intensity of the point process.
q,p	Optional. Exponents for the contrast criterion.
rmin, rmax	Optional. The interval of r values for the contrast criterion.
...	Optional arguments passed to optim to control the optimisation algorithm. See Details.

Details

This algorithm fits a log-Gaussian Cox point process (LGCP) model to a point pattern dataset by the Method of Minimum Contrast, using the K function of the point pattern.

The shape of the covariance of the LGCP must be specified: the default is the exponential covariance function, but other covariance models can be selected.

The argument X can be either

a point pattern: An object of class "ppp" representing a point pattern dataset. The K function of the point pattern will be computed using Kest, and the method of minimum contrast will be applied to this.

a summary statistic: An object of class "fv" containing the values of a summary statistic, computed for a point pattern dataset. The summary statistic should be the K function, and this object should have been obtained by a call to Kest or one of its relatives.

The algorithm fits a log-Gaussian Cox point process (LGCP) model to X, by finding the parameters of the LGCP model which give the closest match between the theoretical K function of the LGCP model and the observed K function. For a more detailed explanation of the Method of Minimum Contrast, see mincontrast.

The model fitted is a stationary, isotropic log-Gaussian Cox process (Møller and Waagepetersen, 2003, pp. 72-76). To define this process we start with a stationary Gaussian random field Z in the two-dimensional plane, with constant mean μ and covariance function $C(r)$. Given Z, we generate a Poisson point process Y with intensity function $\lambda(u) = \exp(Z(u))$ at location u. Then Y is a log-Gaussian Cox process.

The K-function of the LGCP is

$$K(r) = \int_0^r 2\pi s \exp(C(s))\,\mathrm{d}s.$$

The intensity of the LGCP is

$$\lambda = \exp(\mu + \frac{C(0)}{2}).$$

The covariance function $C(r)$ is parametrised in the form

$$C(r) = \sigma^2 c(r/\alpha)$$

where σ^2 and α are parameters controlling the strength and the scale of autocorrelation, respectively, and $c(r)$ is a known covariance function determining the shape of the covariance. The strength and scale parameters σ^2 and α will be estimated by the algorithm as the values `var` and `scale` respectively. The template covariance function $c(r)$ must be specified as explained below.

In this algorithm, the Method of Minimum Contrast is first used to find optimal values of the parameters σ^2 and α. Then the remaining parameter μ is inferred from the estimated intensity λ.

The template covariance function $c(r)$ is specified using the argument `covmodel`. This should be of the form `list(model="modelname", ...)` where `modelname` is a string identifying the template model as explained below, and `...` are optional arguments of the form `tag=value` giving the values of parameters controlling the *shape* of the template model. The default is the exponential covariance $c(r) = e^{-r}$ so that the scaled covariance is

$$C(r) = \sigma^2 e^{-r/\alpha}.$$

To determine the template model, the string `"modelname"` will be prefixed by `"RM"` and the code will search for a function of this name in the **RandomFields** package. For a list of available models see `RMmodel` in the **RandomFields** package. For example the Matérn covariance with exponent $\nu = 0.3$ is specified by `covmodel=list(model="matern", nu=0.3)` corresponding to the function `RMmatern` in the **RandomFields** package.

If the argument `lambda` is provided, then this is used as the value of λ. Otherwise, if `X` is a point pattern, then λ will be estimated from `X`. If `X` is a summary statistic and `lambda` is missing, then the intensity λ cannot be estimated, and the parameter μ will be returned as `NA`.

The remaining arguments `rmin,rmax,q,p` control the method of minimum contrast; see `mincontrast`.

The optimisation algorithm can be controlled through the additional arguments `"..."` which are passed to the optimisation function `optim`. For example, to constrain the parameter values to a certain range, use the argument `method="L-BFGS-B"` to select an optimisation algorithm that respects box constraints, and use the arguments `lower` and `upper` to specify (vectors of) minimum and maximum values for each parameter.

Value

An object of class `"minconfit"`. There are methods for printing and plotting this object. It contains the following main components:

`par`	Vector of fitted parameter values.
`fit`	Function value table (object of class `"fv"`) containing the observed values of the summary statistic (`observed`) and the theoretical values of the summary statistic computed from the fitted model parameters.

Note

This function is considerably slower than `lgcp.estpcf` because of the computation time required for the integral in the K-function.

Computation can be accelerated, at the cost of less accurate results, by setting `spatstat.options(fastK.lgcp=TRUE)`.

Author(s)

Rasmus Waagepetersen <rw@math.auc.dk>. Adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au

Further modifications by Rasmus Waagepetersen and Shen Guochun, and by Ege Rubak <rubak@math.aau.dk>.

References

Møller, J, Syversveen, A. and Waagepetersen, R. (1998) Log Gaussian Cox Processes. *Scandinavian Journal of Statistics* **25**, 451–482.

Møller, J. and Waagepetersen, R. (2003). Statistical Inference and Simulation for Spatial Point Processes. Chapman and Hall/CRC, Boca Raton.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

`lgcp.estpcf` for alternative method of fitting LGCP.

`matclust.estK`, `thomas.estK` for other models.

`mincontrast` for the generic minimum contrast fitting algorithm, including important parameters that affect the accuracy of the fit.

`RMmodel` in the **RandomFields** package, for covariance function models.

`Kest` for the K function.

Examples

```
if(interactive()) {
  u <- lgcp.estK(redwood)
  print(u)
  plot(u)
} else {
  # faster - better starting point
  u <- lgcp.estK(redwood, c(var=1.05, scale=0.1))
}


if(FALSE) {
  ## takes several minutes!
  lgcp.estK(redwood, covmodel=list(model="matern", nu=0.3))
}
```

---

lgcp.estpcf *Fit a Log-Gaussian Cox Point Process by Minimum Contrast*

Description

Fits a log-Gaussian Cox point process model to a point pattern dataset by the Method of Minimum Contrast using the pair correlation function.

Usage

```
lgcp.estpcf(X,
            startpar=c(var=1,scale=1),
            covmodel=list(model="exponential"),
            lambda=NULL,
            q = 1/4, p = 2, rmin = NULL, rmax = NULL, ..., pcfargs=list())
```

Arguments

X	Data to which the model will be fitted. Either a point pattern or a summary statistic. See Details.
startpar	Vector of starting values for the parameters of the log-Gaussian Cox process model.
covmodel	Specification of the covariance model for the log-Gaussian field. See Details.
lambda	Optional. An estimate of the intensity of the point process.
q,p	Optional. Exponents for the contrast criterion.
rmin, rmax	Optional. The interval of r values for the contrast criterion.
...	Optional arguments passed to optim to control the optimisation algorithm. See Details.
pcfargs	Optional list containing arguments passed to pcf.ppp to control the smoothing in the estimation of the pair correlation function.

Details

This algorithm fits a log-Gaussian Cox point process (LGCP) model to a point pattern dataset by the Method of Minimum Contrast, using the estimated pair correlation function of the point pattern.

The shape of the covariance of the LGCP must be specified: the default is the exponential covariance function, but other covariance models can be selected.

The argument X can be either

a point pattern: An object of class "ppp" representing a point pattern dataset. The pair correlation function of the point pattern will be computed using pcf, and the method of minimum contrast will be applied to this.

a summary statistic: An object of class "fv" containing the values of a summary statistic, computed for a point pattern dataset. The summary statistic should be the pair correlation function, and this object should have been obtained by a call to pcf or one of its relatives.

The algorithm fits a log-Gaussian Cox point process (LGCP) model to X, by finding the parameters of the LGCP model which give the closest match between the theoretical pair correlation function of the LGCP model and the observed pair correlation function. For a more detailed explanation of the Method of Minimum Contrast, see mincontrast.

The model fitted is a stationary, isotropic log-Gaussian Cox process (Møller and Waagepetersen, 2003, pp. 72-76). To define this process we start with a stationary Gaussian random field Z in the two-dimensional plane, with constant mean μ and covariance function $C(r)$. Given Z, we generate a Poisson point process Y with intensity function $\lambda(u) = \exp(Z(u))$ at location u. Then Y is a log-Gaussian Cox process.

The theoretical pair correlation function of the LGCP is

$$g(r) = \exp(C(s))$$

The intensity of the LGCP is

$$\lambda = \exp(\mu + \frac{C(0)}{2}).$$

The covariance function $C(r)$ takes the form

$$C(r) = \sigma^2 c(r/\alpha)$$

where σ^2 and α are parameters controlling the strength and the scale of autocorrelation, respectively, and $c(r)$ is a known covariance function determining the shape of the covariance. The strength and scale parameters σ^2 and α will be estimated by the algorithm. The template covariance function $c(r)$ must be specified as explained below.

In this algorithm, the Method of Minimum Contrast is first used to find optimal values of the parameters σ^2 and α. Then the remaining parameter μ is inferred from the estimated intensity λ.

The template covariance function $c(r)$ is specified using the argument `covmodel`. This should be of the form `list(model="modelname", ...)` where `modelname` is a string identifying the template model as explained below, and `...` are optional arguments of the form `tag=value` giving the values of parameters controlling the *shape* of the template model. The default is the exponential covariance $c(r) = e^{-r}$ so that the scaled covariance is

$$C(r) = \sigma^2 e^{-r/\alpha}.$$

To determine the template model, the string `"modelname"` will be prefixed by `"RM"` and the code will search for a function of this name in the **RandomFields** package. For a list of available models see `RMmodel` in the **RandomFields** package. For example the Matérn covariance with exponent $\nu = 0.3$ is specified by `covmodel=list(model="matern", nu=0.3)` corresponding to the function `RMmatern` in the **RandomFields** package.

If the argument `lambda` is provided, then this is used as the value of λ. Otherwise, if `X` is a point pattern, then λ will be estimated from `X`. If `X` is a summary statistic and `lambda` is missing, then the intensity λ cannot be estimated, and the parameter μ will be returned as `NA`.

The remaining arguments `rmin,rmax,q,p` control the method of minimum contrast; see `mincontrast`.

The optimisation algorithm can be controlled through the additional arguments `"..."` which are passed to the optimisation function `optim`. For example, to constrain the parameter values to a certain range, use the argument `method="L-BFGS-B"` to select an optimisation algorithm that respects box constraints, and use the arguments `lower` and `upper` to specify (vectors of) minimum and maximum values for each parameter.

Value

An object of class `"minconfit"`. There are methods for printing and plotting this object. It contains the following main components:

`par`	Vector of fitted parameter values.
`fit`	Function value table (object of class `"fv"`) containing the observed values of the summary statistic (`observed`) and the theoretical values of the summary statistic computed from the fitted model parameters.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> with modifications by Shen Guochun and Rasmus Waagepetersen <rw@math.auc.dk> and Ege Rubak <rubak@math.aau.dk>.

References

Møller, J., Syversveen, A. and Waagepetersen, R. (1998) Log Gaussian Cox Processes. *Scandinavian Journal of Statistics* **25**, 451–482.

Møller, J. and Waagepetersen, R. (2003). Statistical Inference and Simulation for Spatial Point Processes. Chapman and Hall/CRC, Boca Raton.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

`lgcp.estK` for alternative method of fitting LGCP.

`matclust.estpcf`, `thomas.estpcf` for other models.

`mincontrast` for the generic minimum contrast fitting algorithm, including important parameters that affect the accuracy of the fit.

`RMmodel` in the **RandomFields** package, for covariance function models.

`pcf` for the pair correlation function.

Examples

```
data(redwood)
u <- lgcp.estpcf(redwood, c(var=1, scale=0.1))
u
plot(u)
if(require(RandomFields)) {
  lgcp.estpcf(redwood, covmodel=list(model="matern", nu=0.3))
}
```

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 <r.turner@auckland.ac.nz>

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

```
data(japanesepines)
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 <r.turner@auckland.ac.nz>

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

```
data(ponderosa)
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 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

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 <r.turner@auckland.ac.nz> 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 <r.turner@auckland.ac.nz>

See Also

Kinhom, Linhom, localK, localL.

Examples

```
data(ponderosa)
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 <r.turner@auckland.ac.nz> 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)
```

`logLik.dppm` *Log Likelihood and AIC for Fitted Determinantal Point Process Model*

Description

Extracts the log Palm likelihood, deviance, and AIC of a fitted determinantal point process model.

Usage

```
## S3 method for class 'dppm'
logLik(object, ...)
## S3 method for class 'dppm'
AIC(object, ..., k=2)
## S3 method for class 'dppm'
extractAIC(fit, scale=0, k=2, ...)
## S3 method for class 'dppm'
nobs(object, ...)
```

Arguments

`object,fit`	Fitted point process model. An object of class `"dppm"`.
`...`	Ignored.
`scale`	Ignored.
`k`	Numeric value specifying the weight of the equivalent degrees of freedom in the AIC. See Details.

Details

These functions are methods for the generic commands `logLik`, `extractAIC` and `nobs` for the class `"dppm"`.

An object of class `"dppm"` represents a fitted Cox or cluster point process model. It is obtained from the model-fitting function `dppm`.

These methods apply only when the model was fitted by maximising the Palm likelihood (Tanaka et al, 2008) by calling `dppm` with the argument `method="palm"`.

The method logLik.dppm computes the maximised value of the log Palm likelihood for the fitted model object.

The methods AIC.dppm and extractAIC.dppm compute the Akaike Information Criterion AIC for the fitted model based on the Palm likelihood (Tanaka et al, 2008)

$$AIC = -2\log(PL) + k \times \text{edf}$$

where PL is the maximised Palm likelihood of the fitted model, and edf is the effective degrees of freedom of the model.

The method nobs.dppm returns the number of points in the original data point pattern to which the model was fitted.

The R function step uses these methods, but it does not work for determinantal models yet due to a missing implementation of update.dppm.

Value

logLik returns a numerical value, belonging to the class "logLik", with an attribute "df" giving the degrees of freedom.

AIC returns a numerical value.

extractAIC returns a numeric vector of length 2 containing the degrees of freedom and the AIC value.

nobs returns an integer value.

Author(s)

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

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

Tanaka, U. and Ogata, Y. and Stoyan, D. (2008) Parameter estimation and model selection for Neyman-Scott point processes. *Biometrical Journal* **50**, 43–57.

See Also

dppm, logLik.ppm

Examples

```
fit <- dppm(swedishpines ~ x, dppGauss(), method="palm")
nobs(fit)
logLik(fit)
extractAIC(fit)
AIC(fit)
```

logLik.kppm *Log Likelihood and AIC for Fitted Cox or Cluster Point Process Model*

Description

Extracts the log composite likelihood, deviance, and AIC of a fitted Cox or cluster point process model.

Usage

```
## S3 method for class 'kppm'
logLik(object, ...)
## S3 method for class 'kppm'
AIC(object, ..., k=2)
## S3 method for class 'kppm'
extractAIC(fit, scale=0, k=2, ...)
## S3 method for class 'kppm'
nobs(object, ...)
```

Arguments

object,fit	Fitted point process model. An object of class "kppm".
...	Ignored.
scale	Ignored.
k	Numeric value specifying the weight of the equivalent degrees of freedom in the AIC. See Details.

Details

These functions are methods for the generic commands logLik, extractAIC and nobs for the class "kppm".

An object of class "kppm" represents a fitted Cox or cluster point process model. It is obtained from the model-fitting function kppm.

These methods apply only when the model was fitted by maximising a composite likelihood: either the Palm likelihood (Tanaka et al, 2008) or the second order composite likelihood (Guan, 2006), by calling kppm with the argument method="palm" or method="clik2" respectively.

The method logLik.kppm computes the maximised value of the log composite likelihood for the fitted model object.

The methods AIC.kppm and extractAIC.kppm compute the Akaike Information Criterion AIC for the fitted model based on the composite likelihood

$$AIC = -2\log(CL) + k \times \mathrm{edf}$$

where CL is the maximised composite likelihood of the fitted model, and edf is the effective degrees of freedom of the model.

The method nobs.kppm returns the number of points in the original data point pattern to which the model was fitted.

The R function step uses these methods.

Value

logLik returns a numerical value, belonging to the class "logLik", with an attribute "df" giving the degrees of freedom.

AIC returns a numerical value.

extractAIC returns a numeric vector of length 2 containing the degrees of freedom and the AIC value.

nobs returns an integer value.

Model comparison

The values of log-likelihood and AIC returned by these functions are based on the *composite likelihood* of the cluster process or Cox process model. They are available only when the model was fitted using method="palm" or method="clik2".

For model comparison and model selection, it is valid to compare the logLik values, or to compare the AIC values, but only when all the models are of class "kppm" and were fitted using the same method.

For method="palm" some theoretical justification was provided by Tanaka et al (2008).

Author(s)

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

References

Guan, Y. (2006) A composite likelihood approach in fitting spatial point process models. *Journal of the American Statistical Association* **101**, 1502–1512.

Tanaka, U. and Ogata, Y. and Stoyan, D. (2008) Parameter estimation and model selection for Neyman-Scott point processes. *Biometrical Journal* **50**, 43–57.

See Also

kppm, logLik.ppm

Examples

```
fit <- kppm(redwood ~ x, "Thomas", method="palm")
nobs(fit)
logLik(fit)
extractAIC(fit)
AIC(fit)
step(fit)
```

logLik.mppm *Log Likelihood and AIC for Multiple Point Process Model*

Description

For a point process model that has been fitted to multiple point patterns, these functions extract the log likelihood and AIC, or analogous quantities based on the pseudolikelihood.

Usage

```
 ## S3 method for class 'mppm'
logLik(object, ..., warn=TRUE)

 ## S3 method for class 'mppm'
AIC(object, ..., k=2, takeuchi=TRUE)

 ## S3 method for class 'mppm'
extractAIC(fit, scale = 0, k = 2, ..., takeuchi = TRUE)

 ## S3 method for class 'mppm'
nobs(object, ...)

 ## S3 method for class 'mppm'
getCall(x, ...)

 ## S3 method for class 'mppm'
terms(x, ...)
```

Arguments

object,fit,x	Fitted point process model (fitted to multiple point patterns). An object of class "mppm".
...	Ignored.
warn	If TRUE, a warning is given when the pseudolikelihood is returned instead of the likelihood.
scale	Ignored.
k	Numeric value specifying the weight of the equivalent degrees of freedom in the AIC. See Details.
takeuchi	Logical value specifying whether to use the Takeuchi penalty (takeuchi=TRUE) or the number of fitted parameters (takeuchi=FALSE) in calculating AIC.

Details

These functions are methods for the generic commands logLik, AIC, extractAIC, terms and getCall for the class "mppm".

An object of class "mppm" represents a fitted Poisson or Gibbs point process model fitted to several point patterns. It is obtained from the model-fitting function mppm.

The method logLik.mppm extracts the maximised value of the log likelihood for the fitted model (as approximated by quadrature using the Berman-Turner approximation). If object is not a Poisson process, the maximised log *pseudolikelihood* is returned, with a warning.

The Akaike Information Criterion AIC for a fitted model is defined as

$$AIC = -2\log(L) + k \times \text{penalty}$$

where L is the maximised likelihood of the fitted model, and penalty is a penalty for model complexity, usually equal to the effective degrees of freedom of the model. The method extractAIC.mppm returns the *analogous* quantity $AIC*$ in which L is replaced by $L*$, the quadrature approximation to the likelihood (if fit is a Poisson model) or the pseudolikelihood (if fit is a Gibbs model).

The penalty term is calculated as follows. If takeuchi=FALSE then penalty is the number of fitted parameters. If takeuchi=TRUE then penalty $=$ trace(JH^{-1}) where J and H are the estimated variance and hessian, respectively, of the composite score. These two choices are equivalent for a Poisson process.

The method nobs.mppm returns the total number of points in the original data point patterns to which the model was fitted.

The method getCall.mppm extracts the original call to mppm which caused the model to be fitted.

The method terms.mppm extracts the covariate terms in the model formula as a terms object. Note that these terms do not include the interaction component of the model.

The R function step uses these methods.

Value

See the help files for the corresponding generic functions.

Author(s)

Adrian Baddeley, Ida-Maria Sintorn and Leanne Bischoff. Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.ed
Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

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

See Also

mppm

Examples

```
fit <- mppm(Bugs ~ x, hyperframe(Bugs=waterstriders))
logLik(fit)
AIC(fit)
nobs(fit)
getCall(fit)
```

logLik.ppm *Log Likelihood and AIC for Point Process Model*

Description

Extracts the log likelihood, deviance, and AIC of a fitted Poisson point process model, or analogous quantities based on the pseudolikelihood or logistic likelihood for a fitted Gibbs point process model.

Usage

```
## S3 method for class 'ppm'
logLik(object, ..., new.coef=NULL, warn=TRUE, absolute=FALSE)

## S3 method for class 'ppm'
deviance(object, ...)

## S3 method for class 'ppm'
AIC(object, ..., k=2, takeuchi=TRUE)

## S3 method for class 'ppm'
extractAIC(fit, scale=0, k=2, ..., takeuchi=TRUE)

## S3 method for class 'ppm'
nobs(object, ...)
```

Arguments

object,fit	Fitted point process model. An object of class "ppm".
...	Ignored.
warn	If TRUE, a warning is given when the pseudolikelihood or logistic likelihood is returned instead of the likelihood.
absolute	Logical value indicating whether to include constant terms in the loglikelihood.
scale	Ignored.
k	Numeric value specifying the weight of the equivalent degrees of freedom in the AIC. See Details.
new.coef	New values for the canonical parameters of the model. A numeric vector of the same length as coef(object).
takeuchi	Logical value specifying whether to use the Takeuchi penalty (takeuchi=TRUE) or the number of fitted parameters (takeuchi=FALSE) in calculating AIC.

Details

These functions are methods for the generic commands logLik, deviance, extractAIC and nobs for the class "ppm".

An object of class "ppm" represents a fitted Poisson or Gibbs point process model. It is obtained from the model-fitting function ppm.

The method logLik.ppm computes the maximised value of the log likelihood for the fitted model object (as approximated by quadrature using the Berman-Turner approximation) is extracted. If object is not a Poisson process, the maximised log *pseudolikelihood* is returned, with a warning (if warn=TRUE).

The Akaike Information Criterion AIC for a fitted model is defined as

$$AIC = -2\log(L) + k \times \text{penalty}$$

where L is the maximised likelihood of the fitted model, and penalty is a penalty for model complexity, usually equal to the effective degrees of freedom of the model. The method extractAIC.ppm returns the *analogous* quantity $AIC*$ in which L is replaced by $L*$, the quadrature approximation to the likelihood (if fit is a Poisson model) or the pseudolikelihood or logistic likelihood (if fit is a Gibbs model).

The penalty term is calculated as follows. If takeuchi=FALSE then penalty is the number of fitted parameters. If takeuchi=TRUE then penalty $=$ trace(JH^{-1}) where J and H are the estimated variance and hessian, respectively, of the composite score. These two choices are equivalent for a Poisson process.

The method nobs.ppm returns the number of points in the original data point pattern to which the model was fitted.

The R function step uses these methods.

Value

logLik returns a numerical value, belonging to the class "logLik", with an attribute "df" giving the degrees of freedom.

AIC returns a numerical value.

extractAIC returns a numeric vector of length 2 containing the degrees of freedom and the AIC value.

nobs returns an integer value.

Model comparison

The values of logLik and AIC returned by these functions are based on the *pseudolikelihood* of the Gibbs point process model. If the model is a Poisson process, then the pseudolikelihood is the same as the likelihood, but for other Gibbs models, the pseudolikelihood is different from the likelihood (and the likelihood of a Gibbs model is hard to compute).

For model comparison and model selection, it is valid to compare the logLik values, or to compare the AIC values, but only when all the models are of class "ppm".

Author(s)

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

References

Varin, C. and Vidoni, P. (2005) A note on composite likelihood inference and model selection. *Biometrika* **92**, 519–528.

See Also

ppm, as.owin, anova.ppm, coef.ppm, fitted.ppm, formula.ppm, model.frame.ppm, model.matrix.ppm, plot.ppm, predict.ppm, residuals.ppm, simulate.ppm, summary.ppm, terms.ppm, update.ppm, vcov.ppm.

Examples

```
data(cells)
fit <- ppm(cells, ~x)
nobs(fit)
logLik(fit)
deviance(fit)
extractAIC(fit)
AIC(fit)
step(fit)
```

---

logLik.slrm *Loglikelihood of Spatial Logistic Regression*

Description

Computes the (maximised) loglikelihood of a fitted Spatial Logistic Regression model.

Usage

```
## S3 method for class 'slrm'
logLik(object, ..., adjust = TRUE)
```

Arguments

object	a fitted spatial logistic regression model. An object of class "slrm".
...	Ignored.
adjust	Logical value indicating whether to adjust the loglikelihood of the model to make it comparable with a point process likelihood. See Details.

Details

This is a method for logLik for fitted spatial logistic regression models (objects of class "slrm", usually obtained from the function slrm). It computes the log-likelihood of a fitted spatial logistic regression model.

If adjust=FALSE, the loglikelihood is computed using the standard formula for the loglikelihood of a logistic regression model for a finite set of (pixel) observations.

If adjust=TRUE then the loglikelihood is adjusted so that it is approximately comparable with the likelihood of a point process in continuous space, by subtracting the value $n \log(a)$ where n is the number of points in the original point pattern dataset, and a is the area of one pixel.

Value

A numerical value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> <adrian@maths.uwa.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

slrm

Examples

```
X <- rpoispp(42)
fit <- slrm(X ~ x+y)
logLik(fit)
logLik(fit, adjust=FALSE)
```

---

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 <r.turner@auckland.ac.nz> 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)
```

---

lurking *Lurking Variable Plot*

Description

Plot spatial point process residuals against a covariate

Usage

```
lurking(object, ...)

## S3 method for class 'ppm'
lurking(object, covariate,
                    type="eem",
                    cumulative=TRUE,
                    ...,
                    plot.it = TRUE,
                    plot.sd = is.poisson(object),
                    clipwindow=default.clipwindow(object),
                    rv = NULL,
                    envelope=FALSE, nsim=39, nrank=1,
                    typename,
                    covname,
                    oldstyle=FALSE,
                    check=TRUE,
```

```
                    verbose=TRUE,
                    nx=128,
                    splineargs=list(spar=0.5),
                    internal=NULL)

## S3 method for class 'ppp'
lurking(object, covariate,
                    type="eem",
                    cumulative=TRUE,
                    ...,
                    plot.it = TRUE,
                    plot.sd = is.poisson(object),
                    clipwindow=default.clipwindow(object),
                    rv = NULL,
                    envelope=FALSE, nsim=39, nrank=1,
                    typename,
                    covname,
                    oldstyle=FALSE,
                    check=TRUE,
                    verbose=TRUE,
                    nx=128,
                    splineargs=list(spar=0.5),
                    internal=NULL)
```

Arguments

object	The fitted point process model (an object of class "ppm") for which diagnostics should be produced. This object is usually obtained from ppm. Alternatively, object may be a point pattern (object of class "ppp").
covariate	The covariate against which residuals should be plotted. Either a numeric vector, a pixel image, or an expression. See *Details* below.
type	String indicating the type of residuals or weights to be computed. Choices include "eem", "raw", "inverse" and "pearson". See diagnose.ppm for all possible choices.
cumulative	Logical flag indicating whether to plot a cumulative sum of marks (cumulative=TRUE) or the derivative of this sum, a marginal density of the smoothed residual field (cumulative=FALSE).
...	Arguments passed to plot.default and lines to control the plot behaviour.
plot.it	Logical value indicating whether plots should be shown. If plot.it=FALSE, only the computed coordinates for the plots are returned. See *Value*.
plot.sd	Logical value indicating whether error bounds should be added to plot. The default is TRUE for Poisson models and FALSE for non-Poisson models. See Details.
clipwindow	If not NULL this argument indicates that residuals shall only be computed inside a subregion of the window containing the original point pattern data. Then clipwindow should be a window object of class "owin".

rv	Usually absent. If this argument is present, the point process residuals will not be calculated from the fitted model object, but will instead be taken directly from rv.
envelope	Logical value indicating whether to compute simulation envelopes for the plot. Alternatively envelope may be a list of point patterns to use for computing the simulation envelopes, or an object of class "envelope" containing simulated point patterns.
nsim	Number of simulated point patterns to be generated to produce the simulation envelope, if envelope=TRUE.
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.
typename	Usually absent. If this argument is present, it should be a string, and will be used (in the axis labels of plots) to describe the type of residuals.
covname	A string name for the covariate, to be used in axis labels of plots.
oldstyle	Logical flag indicating whether error bounds should be plotted using the approximation given in the original paper (oldstyle=TRUE), or using the correct asymptotic formula (oldstyle=FALSE).
check	Logical flag indicating whether the integrity of the data structure in object should be checked.
verbose	Logical value indicating whether to print progress reports during Monte Carlo simulation.
nx	Integer. Number of covariate values to be used in the plot.
splineargs	A list of arguments passed to smooth.spline for the estimation of the derivatives in the case cumulative=FALSE.
internal	Internal use only.

Details

This function generates a 'lurking variable' plot for a fitted point process model. Residuals from the model represented by object are plotted against the covariate specified by covariate. This plot can be used to reveal departures from the fitted model, in particular, to reveal that the point pattern depends on the covariate.

The function lurking is generic, with methods for ppm and ppp documented here, and possibly other methods.

The argument object would usually be a fitted point process model (object of class "ppm") produced by the model-fitting algorithm ppm). If object is a point pattern (object of class "ppp") then the model is taken to be the uniform Poisson process (Complete Spatial Randomness) fitted to this point pattern.

First the residuals from the fitted model (Baddeley et al, 2004) are computed at each quadrature point, or alternatively the 'exponential energy marks' (Stoyan and Grabarnik, 1991) are computed at each data point. The argument type selects the type of residual or weight. See diagnose.ppm for options and explanation.

A lurking variable plot for point processes (Baddeley et al, 2004) displays either the cumulative sum of residuals/weights (if cumulative = TRUE) or a kernel-weighted average of the residuals/weights

(if cumulative = FALSE) plotted against the covariate. The empirical plot (solid lines) is shown together with its expected value assuming the model is true (dashed lines) and optionally also the pointwise two-standard-deviation limits (grey shading).

To be more precise, let $Z(u)$ denote the value of the covariate at a spatial location u.

- If cumulative=TRUE then we plot $H(z)$ against z, where $H(z)$ is the sum of the residuals over all quadrature points where the covariate takes a value less than or equal to z, or the sum of the exponential energy weights over all data points where the covariate takes a value less than or equal to z.
- If cumulative=FALSE then we plot $h(z)$ against z, where $h(z)$ is the derivative of $H(z)$, computed approximately by spline smoothing.

For the point process residuals $E(H(z)) = 0$, while for the exponential energy weights $E(H(z)) =$ area of the subset of the window satisfying $Z(u) <= z$.

If the empirical and theoretical curves deviate substantially from one another, the interpretation is that the fitted model does not correctly account for dependence on the covariate. The correct form (of the spatial trend part of the model) may be suggested by the shape of the plot.

If plot.sd = TRUE, then superimposed on the lurking variable plot are the pointwise two-standard-deviation error limits for $H(x)$ calculated for the inhomogeneous Poisson process. The default is plot.sd = TRUE for Poisson models and plot.sd = FALSE for non-Poisson models.

By default, the two-standard-deviation limits are calculated from the exact formula for the asymptotic variance of the residuals under the asymptotic normal approximation, equation (37) of Baddeley et al (2006). However, for compatibility with the original paper of Baddeley et al (2005), if oldstyle=TRUE, the two-standard-deviation limits are calculated using the innovation variance, an over-estimate of the true variance of the residuals.

The argument covariate is either a numeric vector, a pixel image, or an R language expression. If it is a numeric vector, it is assumed to contain the values of the covariate for each of the quadrature points in the fitted model. The quadrature points can be extracted by quad.ppm(object).

If covariate is a pixel image, it is assumed to contain the values of the covariate at each location in the window. The values of this image at the quadrature points will be extracted.

Alternatively, if covariate is an expression, it will be evaluated in the same environment as the model formula used in fitting the model object. It must yield a vector of the same length as the number of quadrature points. The expression may contain the terms x and y representing the cartesian coordinates, and may also contain other variables that were available when the model was fitted. Certain variable names are reserved words; see ppm.

Note that lurking variable plots for the x and y coordinates are also generated by diagnose.ppm, amongst other types of diagnostic plots. This function is more general in that it enables the user to plot the residuals against any chosen covariate that may have been present.

For advanced use, even the values of the residuals/weights can be altered. If the argument rv is present, the residuals will not be calculated from the fitted model object but will instead be taken directly from the object rv. If type = "eem" then rv should be similar to the return value of eem, namely, a numeric vector with length equal to the number of data points in the original point pattern. Otherwise, rv should be similar to the return value of residuals.ppm, that is, rv should be an object of class "msr" (see msr) representing a signed measure.

Value

The (invisible) return value is an object belonging to the class "lurk", for which there are methods for plot and print.

This object is a list containing two dataframes empirical and theoretical. The first dataframe empirical contains columns covariate and value giving the coordinates of the lurking variable plot. The second dataframe theoretical contains columns covariate, mean and sd giving the coordinates of the plot of the theoretical mean and standard deviation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>.

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.

Baddeley, A., Møller, J. and Pakes, A.G. (2006) Properties of residuals for spatial point processes. *Annals of the Institute of Statistical Mathematics* **60**, 627–649.

Stoyan, D. and Grabarnik, P. (1991) Second-order characteristics for stochastic structures connected with Gibbs point processes. *Mathematische Nachrichten*, 151:95–100.

See Also

residuals.ppm, diagnose.ppm, residuals.ppm, qqplot.ppm, eem, ppm

Examples

```
(a <- lurking(nztrees, expression(x), type="raw"))
fit <- ppm(nztrees ~x, Poisson(), nd=128)
(b <- lurking(fit, expression(x), type="raw"))
lurking(fit, expression(x), type="raw", cumulative=FALSE)
```

---

lurking.mppm *Lurking Variable Plot for Multiple Point Patterns*

Description

Generate a lurking variable plot of spatial point process residuals against a covariate, for a model fitted to several point patterns.

Usage

```
## S3 method for class 'mppm'
lurking(object, covariate, type="eem",
                        ...,
                        separate = FALSE,
                        plot.it = TRUE,
                        covname, oldstyle = FALSE, nx = 512, main="")
```

Arguments

object	The fitted model. An object of class "mppm" representing a point process model fitted to several point patterns.
covariate	The covariate to be used on the horizontal axis. Either an expression which can be evaluated in the original data, or a list of pixel images, one image for each point pattern in the original data.
type	String indicating the type of residuals or weights to be computed. Choices include "eem", "raw", "inverse" and "pearson". See diagnose.ppm for all possible choices.
...	Additional arguments passed to lurking.ppm, including arguments controlling the plot.
separate	Logical value indicating whether to compute a separate lurking variable plot for each of the original point patterns. If FALSE (the default), a single lurking-variable plot is produced by combining residuals from all patterns.
plot.it	Logical value indicating whether plots should be shown. If plot.it=FALSE, only the computed coordinates for the plots are returned. See *Value*.
covname	A string name for the covariate, to be used in axis labels of plots.
oldstyle	Logical flag indicating whether error bounds should be plotted using the approximation given in the original paper (oldstyle=TRUE), or using the correct asymptotic formula (oldstyle=FALSE).
nx	Integer. Number of covariate values to be used in the plot.
main	Character string giving a main title for the plot.

Details

This function generates a 'lurking variable' plot for a point process model fitted to several point patterns. Residuals from the model represented by object are plotted against the covariate specified by covariate. This plot can be used to reveal departures from the fitted model.

The function lurking is generic. This is the method for the class mppm. The argument object must be a fitted point process model object of class "mppm") produced by the model-fitting algorithm mppm.

Value

If separate=FALSE (the default), the return value is an object belonging to the class "lurk", for which there are methods for plot and print. See lurking for details of the format.

If separate=TRUE, the result is a list of such objects, and also belongs to the class anylist so that it can be printed and plotted.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, with thanks to Nicholas Read.

See Also

lurking.ppm

Examples

```
fit <- mppm(Points ~ Image + Group, demohyper)
lurking(fit, expression(Image), type="P")
lurking(fit, expression(Image), type="P", separate=TRUE)
```

---

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 <r.turner@auckland.ac.nz>

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'
data(amacrine)
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.

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 <r.turner@auckland.ac.nz> 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 <r.turner@auckland.ac.nz>

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 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 marked point pattern. An object of class `"ppp"`.
`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 <r.turner@auckland.ac.nz>

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 <r.turner@auckland.ac.nz>

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
data(longleaf)
# Subset of this large pattern
swcorner <- owin(c(0,100),c(0,100))
sub <- longleaf[ , swcorner]
# mark correlation function
mv <- markvario(sub)
plot(mv)
```

---

matclust.estK — *Fit the Matern Cluster Point Process by Minimum Contrast*

Description

Fits the Matérn Cluster point process to a point pattern dataset by the Method of Minimum Contrast.

Usage

```
matclust.estK(X, startpar=c(kappa=1,scale=1), lambda=NULL,
            q = 1/4, p = 2, rmin = NULL, rmax = NULL, ...)
```

Arguments

X	Data to which the Matérn Cluster model will be fitted. Either a point pattern or a summary statistic. See Details.
startpar	Vector of starting values for the parameters of the Matérn Cluster process.
lambda	Optional. An estimate of the intensity of the point process.
q,p	Optional. Exponents for the contrast criterion.
rmin, rmax	Optional. The interval of r values for the contrast criterion.
...	Optional arguments passed to optim to control the optimisation algorithm. See Details.

Details

This algorithm fits the Matérn Cluster point process model to a point pattern dataset by the Method of Minimum Contrast, using the K function.

The argument X can be either

a point pattern: An object of class "ppp" representing a point pattern dataset. The K function of the point pattern will be computed using Kest, and the method of minimum contrast will be applied to this.

a summary statistic: An object of class "fv" containing the values of a summary statistic, computed for a point pattern dataset. The summary statistic should be the K function, and this object should have been obtained by a call to Kest or one of its relatives.

The algorithm fits the Matérn Cluster point process to X, by finding the parameters of the Matérn Cluster model which give the closest match between the theoretical K function of the Matérn Cluster process and the observed K function. For a more detailed explanation of the Method of Minimum Contrast, see mincontrast.

The Matérn Cluster point process is described in Møller and Waagepetersen (2003, p. 62). It is a cluster process formed by taking a pattern of parent points, generated according to a Poisson process with intensity κ, and around each parent point, generating a random number of offspring points, such that the number of offspring of each parent is a Poisson random variable with mean μ, and the locations of the offspring points of one parent are independent and uniformly distributed inside a circle of radius R centred on the parent point, where R is equal to the parameter scale. The named vector of stating values can use either R or scale as the name of the second component, but the latter is recommended for consistency with other cluster models.

The theoretical K-function of the Matérn Cluster process is

$$K(r) = \pi r^2 + \frac{1}{\kappa} h(\frac{r}{2R})$$

where the radius R is the parameter scale and

$$h(z) = 2 + \frac{1}{\pi}[(8z^2 - 4)\text{arccos}(z) - 2\text{arcsin}(z) + 4z\sqrt{(1 - z^2)^3} - 6z\sqrt{1 - z^2}]$$

for $z <= 1$, and $h(z) = 1$ for $z > 1$. The theoretical intensity of the Matérn Cluster process is $\lambda = \kappa\mu$.

In this algorithm, the Method of Minimum Contrast is first used to find optimal values of the parameters κ and R. Then the remaining parameter μ is inferred from the estimated intensity λ.

If the argument `lambda` is provided, then this is used as the value of λ. Otherwise, if `X` is a point pattern, then λ will be estimated from `X`. If `X` is a summary statistic and `lambda` is missing, then the intensity λ cannot be estimated, and the parameter μ will be returned as `NA`.

The remaining arguments `rmin,rmax,q,p` control the method of minimum contrast; see `mincontrast`.

The Matérn Cluster process can be simulated, using `rMatClust`.

Homogeneous or inhomogeneous Matérn Cluster models can also be fitted using the function `kppm`.

The optimisation algorithm can be controlled through the additional arguments `"..."` which are passed to the optimisation function `optim`. For example, to constrain the parameter values to a certain range, use the argument `method="L-BFGS-B"` to select an optimisation algorithm that respects box constraints, and use the arguments `lower` and `upper` to specify (vectors of) minimum and maximum values for each parameter.

Value

An object of class `"minconfit"`. There are methods for printing and plotting this object. It contains the following main components:

`par`	Vector of fitted parameter values.
`fit`	Function value table (object of class `"fv"`) containing the observed values of the summary statistic (`observed`) and the theoretical values of the summary statistic computed from the fitted model parameters.

Author(s)

Rasmus Waagepetersen <rw@math.auc.dk> Adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

References

Møller, J. and Waagepetersen, R. (2003). Statistical Inference and Simulation for Spatial Point Processes. Chapman and Hall/CRC, Boca Raton.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

`kppm`, `lgcp.estK`, `thomas.estK`, `mincontrast`, `Kest`, `rMatClust` to simulate the fitted model.

Examples

```
data(redwood)
u <- matclust.estK(redwood, c(kappa=10, scale=0.1))
u
plot(u)
```

matclust.estpcf *Fit the Matérn Cluster Point Process by Minimum Contrast Using Pair Correlation*

Description

Fits the Matérn Cluster point process to a point pattern dataset by the Method of Minimum Contrast using the pair correlation function.

Usage

```
matclust.estpcf(X, startpar=c(kappa=1,scale=1), lambda=NULL,
            q = 1/4, p = 2, rmin = NULL, rmax = NULL, ...,
            pcfargs=list())
```

Arguments

X	Data to which the Matérn Cluster model will be fitted. Either a point pattern or a summary statistic. See Details.
startpar	Vector of starting values for the parameters of the Matérn Cluster process.
lambda	Optional. An estimate of the intensity of the point process.
q,p	Optional. Exponents for the contrast criterion.
rmin, rmax	Optional. The interval of r values for the contrast criterion.
...	Optional arguments passed to optim to control the optimisation algorithm. See Details.
pcfargs	Optional list containing arguments passed to pcf.ppp to control the smoothing in the estimation of the pair correlation function.

Details

This algorithm fits the Matérn Cluster point process model to a point pattern dataset by the Method of Minimum Contrast, using the pair correlation function.

The argument X can be either

a point pattern: An object of class "ppp" representing a point pattern dataset. The pair correlation function of the point pattern will be computed using pcf, and the method of minimum contrast will be applied to this.

a summary statistic: An object of class "fv" containing the values of a summary statistic, computed for a point pattern dataset. The summary statistic should be the pair correlation function, and this object should have been obtained by a call to pcf or one of its relatives.

The algorithm fits the Matérn Cluster point process to X, by finding the parameters of the Matérn Cluster model which give the closest match between the theoretical pair correlation function of the Matérn Cluster process and the observed pair correlation function. For a more detailed explanation of the Method of Minimum Contrast, see mincontrast.

The Matérn Cluster point process is described in Møller and Waagepetersen (2003, p. 62). It is a cluster process formed by taking a pattern of parent points, generated according to a Poisson process with intensity κ, and around each parent point, generating a random number of offspring points, such that the number of offspring of each parent is a Poisson random variable with mean μ, and the locations of the offspring points of one parent are independent and uniformly distributed inside a circle of radius R centred on the parent point, where R is equal to the parameter `scale`. The named vector of stating values can use either `R` or `scale` as the name of the second component, but the latter is recommended for consistency with other cluster models.

The theoretical pair correlation function of the Matérn Cluster process is

$$g(r) = 1 + \frac{1}{4\pi R\kappa r} h(\frac{r}{2R})$$

where the radius R is the parameter `scale` and

$$h(z) = \frac{16}{\pi}[z\arccos(z) - z^2\sqrt{1-z^2}]$$

for $z <= 1$, and $h(z) = 0$ for $z > 1$. The theoretical intensity of the Matérn Cluster process is $\lambda = \kappa\mu$.

In this algorithm, the Method of Minimum Contrast is first used to find optimal values of the parameters κ and R. Then the remaining parameter μ is inferred from the estimated intensity λ.

If the argument `lambda` is provided, then this is used as the value of λ. Otherwise, if `X` is a point pattern, then λ will be estimated from `X`. If `X` is a summary statistic and `lambda` is missing, then the intensity λ cannot be estimated, and the parameter μ will be returned as `NA`.

The remaining arguments `rmin,rmax,q,p` control the method of minimum contrast; see `mincontrast`.

The Matérn Cluster process can be simulated, using `rMatClust`.

Homogeneous or inhomogeneous Matérn Cluster models can also be fitted using the function `kppm`.

The optimisation algorithm can be controlled through the additional arguments `"..."` which are passed to the optimisation function `optim`. For example, to constrain the parameter values to a certain range, use the argument `method="L-BFGS-B"` to select an optimisation algorithm that respects box constraints, and use the arguments `lower` and `upper` to specify (vectors of) minimum and maximum values for each parameter.

Value

An object of class `"minconfit"`. There are methods for printing and plotting this object. It contains the following main components:

`par`	Vector of fitted parameter values.
`fit`	Function value table (object of class `"fv"`) containing the observed values of the summary statistic (`observed`) and the theoretical values of the summary statistic computed from the fitted model parameters.

Author(s)

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

References

Møller, J. and Waagepetersen, R. (2003). Statistical Inference and Simulation for Spatial Point Processes. Chapman and Hall/CRC, Boca Raton.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

kppm, matclust.estK, thomas.estpcf, thomas.estK, lgcp.estK, mincontrast, pcf, rMatClust to simulate the fitted model.

Examples

```
data(redwood)
u <- matclust.estpcf(redwood, c(kappa=10, R=0.1))
u
plot(u, legendpos="topright")
```

---

measureContinuous *Discrete and Continuous Components of a Measure*

Description

Given a measure A (object of class "msr") these functions find the discrete and continuous parts of A.

Usage

```
measureDiscrete(x)
measureContinuous(x)
```

Arguments

x	A measure (object of class "msr").

Details

The functions measureDiscrete and measureContinuous return the discrete and continuous components, respectively, of a measure.

If x is a measure, then measureDiscrete(x) is a measure consisting only of the discrete (atomic) component of x, and measureContinuous(x) is a measure consisting only of the continuous (diffuse) component of x.

Value

Another measure (object of class "msr") on the same spatial domain.

Author(s)

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

References

Halmos, P.R. (1950) *Measure Theory*. Van Nostrand.

See Also

msr, with.msr, split.msr, measurePositive

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X, ~x+y)
rp <- residuals(fit, type="pearson")

rp
measureDiscrete(rp)
measureContinuous(rp)
```

---

measureVariation *Positive and Negative Parts, and Variation, of a Measure*

Description

Given a measure A (object of class "msr") these functions find the positive part, negative part and variation of A.

Usage

```
measurePositive(x)
measureNegative(x)
measureVariation(x)
totalVariation(x)
```

Arguments

x	A measure (object of class "msr").

Details

The functions measurePositive and measureNegative return the positive and negative parts of the measure, and measureVariation returns the variation (sum of positive and negative parts). The function totalVariation returns the total variation norm.

If μ is a signed measure, it can be represented as

$$\mu = \mu_+ - \mu_-$$

where μ_+ and μ_- are *nonnegative* measures called the positive and negative parts of μ. In a nutshell, the positive part of μ consists of all positive contributions or increments, and the negative part consists of all negative contributions multiplied by -1.

The variation $|\mu|$ is defined by

$$\mu = \mu_+ + \mu_-$$

and is also a nonnegative measure.

The total variation norm is the integral of the variation.

Value

The result of measurePositive, measureNegative and measureVariation is another measure (object of class "msr") on the same spatial domain. The result of totalVariation is a non-negative number.

Author(s)

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

References

Halmos, P.R. (1950) *Measure Theory*. Van Nostrand.

See Also

msr, with.msr, split.msr, measureDiscrete

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X, ~x+y)
rp <- residuals(fit, type="pearson")

measurePositive(rp)
measureNegative(rp)
measureVariation(rp)

# total variation norm
totalVariation(rp)
```

---

methods.dppm *Methods for Determinantal Point Process Models*

Description

These are methods for the class "dppm".

Usage

```
## S3 method for class 'dppm'
coef(object, ...)
## S3 method for class 'dppm'
formula(x, ...)
## S3 method for class 'dppm'
print(x, ...)
## S3 method for class 'dppm'
terms(x, ...)
## S3 method for class 'dppm'
labels(object, ...)
```

Arguments

x,object	An object of class "dppm", representing a fitted determinantal point process model.
...	Arguments passed to other methods.

Details

These functions are methods for the generic commands coef, formula, print, terms and labels for the class "dppm".

An object of class "dppm" represents a fitted determinantal point process model. It is obtained from dppm.

The method coef.dppm returns the vector of *regression coefficients* of the fitted model. It does not return the interaction parameters.

Value

See the help files for the corresponding generic functions.

Author(s)

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

See Also

dppm, plot.dppm, predict.dppm, simulate.dppm, as.ppm.dppm.

Examples

```
fit <- dppm(swedishpines ~ x + y, dppGauss, method="c")
coef(fit)
formula(fit)
tf <- terms(fit)
labels(fit)
```

`methods.fii` *Methods for Fitted Interactions*

Description

These are methods specifically for the class `"fii"` of fitted interpoint interactions.

Usage

```
## S3 method for class 'fii'
print(x, ...)

## S3 method for class 'fii'
coef(object, ...)

## S3 replacement method for class 'fii'
coef(object, ...) <- value

## S3 method for class 'fii'
plot(x, ...)

## S3 method for class 'fii'
summary(object,...)

## S3 method for class 'summary.fii'
print(x, ...)

## S3 method for class 'summary.fii'
coef(object, ...)
```

Arguments

`x,object`	An object of class `"fii"` representing a fitted interpoint interaction.
`...`	Arguments passed to other methods.
`value`	Numeric vector containing new values for the fitted interaction coefficients.

Details

These are methods for the class `"fii"`. An object of class `"fii"` represents a fitted interpoint interaction. It is usually obtained by using the command `fitin` to extract the fitted interaction part of a fitted point process model. See `fitin` for further explanation of this class.

The commands listed here are methods for the generic functions `print`, `summary`, `plot`, `coef` and `coef<-` for objects of the class `"fii"`.

Following the usual convention, `summary.fii` returns an object of class `summary.fii`, for which there is a print method. The effect is that, when the user types `summary(x)`, the summary is printed, but when the user types `y <- summary(x)`, the summary information is saved.

The method coef.fii extracts the canonical coefficients of the fitted interaction, and returns them as a numeric vector. The method coef.summary.fii transforms these values into quantities that are more easily interpretable, in a format that depends on the particular model.

There are also methods for the generic commands reach and as.interact, described elsewhere.

Value

The print and plot methods return NULL.

The summary method returns an object of class summary.fii.

coef.fii returns a numeric vector. coef.summary.fii returns data whose structure depends on the model.

Author(s)

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

See Also

fitin, reach.fii, as.interact.fii

Examples

```
mod <- ppm(cells, ~1, Strauss(0.1))
f <- fitin(mod)
f
summary(f)
plot(f)
coef(f)
coef(summary(f))
```

---

methods.influence.ppm *Methods for Influence Objects*

Description

Methods for the class "influence.ppm".

Usage

```
  ## S3 method for class 'influence.ppm'
as.ppp(X, ...)

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

 ## S3 method for class 'influence.ppm'
domain(X, ...)
```

```
 ## S3 method for class 'influence.ppm'
Smooth(X, ...)

 ## S3 method for class 'influence.ppm'
Window(X, ...)

 ## S3 method for class 'influence.ppm'
integral(f, domain, ...)
```

Arguments

X,W,f	An object of class "influence.ppm".
domain	Optional. Domain of integration: a window (class "owin") or a tessellation (class "tess").
...	Additional arguments. See Details.
fatal	Logical value indicating what to do if the data cannot be converted to a window. If fatal=TRUE (the default) an error occurs. If fatal=FALSE a value of NULL is returned.

Details

These functions are methods for the class "influence.ppm". An object of this class represents the influence measure of a fitted point process model (see influence.ppm).

For as.ppp, domain, integral and Window, additional arguments (...) are ignored. For as.owin and Smooth, additional arguments are passed to the method for class "ppp".

Value

A window (object of class "owin") for as.owin, domain and Window. A point pattern (object of class "ppp") for as.ppp. A numeric value or numeric vector for integral. A pixel image, or list of pixel images, for Smooth.

Author(s)

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

See Also

influence.ppm, plot.influence.ppm, [.influence.ppm

Examples

```
fit <- ppm(cells ~ x)
a <- influence(fit)
Window(a)
```

`methods.kppm` *Methods for Cluster Point Process Models*

Description

These are methods for the class "kppm".

Usage

```
## S3 method for class 'kppm'
coef(object, ...)
## S3 method for class 'kppm'
formula(x, ...)
## S3 method for class 'kppm'
print(x, ...)
## S3 method for class 'kppm'
terms(x, ...)
## S3 method for class 'kppm'
labels(object, ...)
```

Arguments

x,object	An object of class "kppm", representing a fitted cluster point process model.
...	Arguments passed to other methods.

Details

These functions are methods for the generic commands coef, formula, print, terms and labels for the class "kppm".

An object of class "kppm" represents a fitted cluster point process model. It is obtained from kppm.

The method coef.kppm returns the vector of *regression coefficients* of the fitted model. It does not return the clustering parameters.

Value

See the help files for the corresponding generic functions.

Author(s)

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

See Also

kppm, plot.kppm, predict.kppm, simulate.kppm, update.kppm, vcov.kppm, as.ppm.kppm.

Examples

```
data(redwood)
fit <- kppm(redwood ~ x, "MatClust")
coef(fit)
formula(fit)
tf <- terms(fit)
labels(fit)
```

`methods.leverage.ppm` *Methods for Leverage Objects*

Description

Methods for the class `"leverage.ppm"`.

Usage

```
  ## S3 method for class 'leverage.ppm'
as.im(X, ..., what=c("smooth", "nearest"))

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

 ## S3 method for class 'leverage.ppm'
domain(X, ...)

 ## S3 method for class 'leverage.ppm'
integral(f, domain, ...)

 ## S3 method for class 'leverage.ppm'
mean(x, ...)

 ## S3 method for class 'leverage.ppm'
Smooth(X, ...)

 ## S3 method for class 'leverage.ppm'
Window(X, ...)
```

Arguments

`X,x,W,f`	An object of class `"leverage.ppm"`.
`domain`	Optional. Domain of integration: a window (class `"owin"`) or a tessellation (class `"tess"`).
`...`	Additional arguments. See Details.

fatal	Logical value indicating what to do if the data cannot be converted to a window. If fatal=TRUE (the default) an error occurs. If fatal=FALSE a value of NULL is returned.
what	Character string (partially matched) specifying which image data should be extracted. See plot.leverage.ppm for explanation.

Details

These functions are methods for the class "leverage.ppm". An object of this class represents the leverage measure of a fitted point process model (see leverage.ppm).

For as.im, domain and Window, additional arguments (...) are ignored. For as.owin, integral, mean and Smooth, additional arguments are passed to the method for class "im".

Value

A window (object of class "owin") for as.owin, domain and Window. A numeric value or numeric vector for integral. A pixel image, or list of pixel images, for as.im and Smooth.

Author(s)

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

See Also

leverage.ppm, plot.leverage.ppm, [.leverage.ppm, as.function.leverage.ppm.

Examples

```
fit <- ppm(cells ~ x)
a <- leverage(fit)
integral(a)
```

---

methods.objsurf *Methods for Objective Function Surfaces*

Description

Methods for printing and plotting an objective function surface.

Usage

```
## S3 method for class 'objsurf'
print(x, ...)
## S3 method for class 'objsurf'
plot(x, ...)
## S3 method for class 'objsurf'
image(x, ...)
```

```
## S3 method for class 'objsurf'
contour(x, ...)
## S3 method for class 'objsurf'
persp(x, ...)
## S3 method for class 'objsurf'
summary(object, ...)
## S3 method for class 'summary.objsurf'
print(x, ...)
```

Arguments

x,object	Object of class "objsurf" representing an objective function surface.
...	Additional arguments passed to plot methods.

Details

These are methods for the generic functions print, plot, image, contour, persp and summary for the class "objsurf".

Value

For print.objsurf, print.summary.objsurf, plot.objsurf and image.objsurf the value is NULL.

For contour.objsurf and persp.objsurf the value is described in the help for contour.default and persp.default respectively.

For summary.objsurf the result is a list, of class summary.objsurf, containing summary information. This list is printed in sensible format by print.summary.objsurf.

Author(s)

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

See Also

objsurf

Examples

```
fit <- kppm(redwood ~ 1, "Thomas")
os <- objsurf(fit)
 os
 summary(os)
 plot(os)
 contour(os, add=TRUE)
 persp(os)
```

`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)
#
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.slrm` *Methods for Spatial Logistic Regression Models*

Description

These are methods for the class `"slrm"`.

Usage

```
## S3 method for class 'slrm'
formula(x, ...)
## S3 method for class 'slrm'
print(x, ...)
## S3 method for class 'slrm'
summary(object, ...)
## S3 method for class 'slrm'
terms(x, ...)
## S3 method for class 'slrm'
labels(object, ...)
## S3 method for class 'slrm'
deviance(object, ...)
## S3 method for class 'slrm'
update(object, ..., evaluate = TRUE, env = parent.frame())
```

Arguments

`x,object`	An object of class `"slrm"`, representing a fitted spatial logistic regression model.
`...`	Arguments passed to other methods.
`evaluate`	Logical value. If `TRUE`, evaluate the updated call to `slrm`, so that the model is refitted; if `FALSE`, simply return the updated call.
`env`	Optional environment in which the model should be updated.

Details

These functions are methods for the generic commands `formula`, `update`, `print`, `summary`, `terms`, `labels` and `deviance` for the class `"slrm"`.

An object of class `"slrm"` represents a fitted spatial logistic regression model. It is obtained from `slrm`.

Value

See the help files for the corresponding generic functions.

Author(s)

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

See Also

slrm, plot.slrm, predict.slrm, simulate.slrm, vcov.slrm, coef.slrm.

Examples

```
fit <- slrm(redwood ~ x)
coef(fit)
formula(fit)
tf <- terms(fit)
labels(fit)
deviance(fit)
```

---

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))
```

---

methods.zclustermodel *Methods for Cluster Models*

Description

Methods for the experimental class of cluster models.

Usage

```
 ## S3 method for class 'zclustermodel'
pcfmodel(model, ...)

 ## S3 method for class 'zclustermodel'
Kmodel(model, ...)

 ## S3 method for class 'zclustermodel'
intensity(X, ...)

 ## S3 method for class 'zclustermodel'
predict(object, ...,
                  locations, type = "intensity", ngrid = NULL)

 ## S3 method for class 'zclustermodel'
print(x, ...)

 ## S3 method for class 'zclustermodel'
clusterradius(model,...,thresh=NULL, precision=FALSE)
```

```
 ## S3 method for class 'zclustermodel'
reach(x, ..., epsilon)
```

Arguments

model,object,x,X	Object of class "zclustermodel".
...	Arguments passed to other methods.
locations	Locations where prediction should be performed. A window or a point pattern.
type	Currently must equal "intensity".
ngrid	Pixel grid dimensions for prediction, if locations is a rectangle or polygon.
thresh,epsilon	Tolerance thresholds
precision	Logical value stipulating whether the precision should also be returned.

Details

Experimental.

Value

Same as for other methods.

Author(s)

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

See Also

zclustermodel

Examples

```
m <- zclustermodel("Thomas", kappa=10, mu=5, scale=0.1)
m2 <- zclustermodel("VarGamma", kappa=10, mu=10, scale=0.1, nu=0.7)
m
m2
g <- pcfmodel(m)
g(0.2)
g2 <- pcfmodel(m2)
g2(1)
Z <- predict(m, locations=square(2))
Z2 <- predict(m2, locations=square(1))
varcount(m, square(1))
varcount(m2, square(1))
```

`methods.zgibbsmodel` *Methods for Gibbs Models*

Description

Methods for the experimental class of Gibbs models

Usage

```
## S3 method for class 'zgibbsmodel'
as.interact(object)
## S3 method for class 'zgibbsmodel'
as.isf(object)
## S3 method for class 'zgibbsmodel'
interactionorder(object)
## S3 method for class 'zgibbsmodel'
is.poisson(x)
## S3 method for class 'zgibbsmodel'
is.stationary(x)
## S3 method for class 'zgibbsmodel'
print(x, ...)
## S3 method for class 'zgibbsmodel'
intensity(X, ..., approx=c("Poisson", "DPP"))
```

Arguments

`object,x,X`	Object of class `"zgibbsmodel"`.
`...`	Additional arguments.
`approx`	Character string (partially matched) specifying the type of approximation.

Details

Experimental.

Value

Same as for other methods.

Author(s)

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

See Also

`zgibbsmodel`

Examples

```
m <- zgibbsmodel(10, Strauss(0.1), -0.5)
m
is.poisson(m)
is.stationary(m)
interactionorder(m)
as.interact(m)
as.isf(m)
intensity(m)
intensity(m, approx="D")
```

`mincontrast` *Method of Minimum Contrast*

Description

A general low-level algorithm for fitting theoretical point process models to point pattern data by the Method of Minimum Contrast.

Usage

```
mincontrast(observed, theoretical, startpar, ...,
          ctrl=list(q = 1/4, p = 2, rmin=NULL, rmax=NULL),
          fvlab=list(label=NULL, desc="minimum contrast fit"),
          explain=list(dataname=NULL, modelname=NULL, fname=NULL),
          action.bad.values=c("warn", "stop", "silent"),
          control=list(), stabilize=TRUE,
          pspace=NULL)
```

Arguments

`observed`	Summary statistic, computed for the data. An object of class `"fv"`.
`theoretical`	An R language function that calculates the theoretical expected value of the summary statistic, given the model parameters. See Details.
`startpar`	Vector of initial values of the parameters of the point process model (passed to `theoretical`).
`...`	Additional arguments passed to the function `theoretical` and to the optimisation algorithm `optim`.
`ctrl`	Optional. List of arguments controlling the optimisation. See Details.
`fvlab`	Optional. List containing some labels for the return value. See Details.
`explain`	Optional. List containing strings that give a human-readable description of the model, the data and the summary statistic.
`action.bad.values`	String (partially matched) specifying what to do if values of the summary statistic are `NA`, `NaN` or infinite. See Details.

control	Optional. Argument passed to optim. A list of parameters which control the behaviour of the optimization algorithm.
stabilize	Logical value specifying whether to numerically stabilize the optimization algorithm, by specifying suitable default values of control$fnscale and control$parscale.
pspace	For internal use by the package only.

Details

This function is a general algorithm for fitting point process models by the Method of Minimum Contrast. If you want to fit the Thomas process, see thomas.estK. If you want to fit a log-Gaussian Cox process, see lgcp.estK. If you want to fit the Matérn cluster process, see matclust.estK.

The Method of Minimum Contrast (Pfanzagl, 1969; Diggle and Gratton, 1984) is a general technique for fitting a point process model to point pattern data. First a summary function (typically the K function) is computed from the data point pattern. Second, the theoretical expected value of this summary statistic under the point process model is derived (if possible, as an algebraic expression involving the parameters of the model) or estimated from simulations of the model. Then the model is fitted by finding the optimal parameter values for the model to give the closest match between the theoretical and empirical curves.

The argument observed should be an object of class "fv" (see fv.object) containing the values of a summary statistic computed from the data point pattern. Usually this is the function $K(r)$ computed by Kest or one of its relatives.

The argument theoretical should be a user-supplied function that computes the theoretical expected value of the summary statistic. It must have an argument named par that will be the vector of parameter values for the model (the length and format of this vector are determined by the starting values in startpar). The function theoretical should also expect a second argument (the first argument other than par) containing values of the distance r for which the theoretical value of the summary statistic $K(r)$ should be computed. The value returned by theoretical should be a vector of the same length as the given vector of r values.

The argument ctrl determines the contrast criterion (the objective function that will be minimised). The algorithm minimises the criterion

$$D(\theta) = \int_{r_{\min}}^{r_{\max}} |\hat{F}(r)^q - F_\theta(r)^q|^p \, \mathrm{d}r$$

where θ is the vector of parameters of the model, $\hat{F}(r)$ is the observed value of the summary statistic computed from the data, $F_\theta(r)$ is the theoretical expected value of the summary statistic, and p, q are two exponents. The default is q = 1/4, p=2 so that the contrast criterion is the integrated squared difference between the fourth roots of the two functions (Waagepetersen, 2007).

The argument action.bad.values specifies what to do if some of the values of the summary statistic are NA, NaN or infinite. If action.bad.values="stop", or if all of the values are bad, then a fatal error occurs. Otherwise, the domain of the summary function is shortened to avoid the bad values. The shortened domain is the longest interval on which the function values are finite (provided this interval is at least half the length of the original domain). A warning is issued if action.bad.values="warn" (the default) and no warning is issued if action.bad.values="silent".

The other arguments just make things print nicely. The argument fvlab contains labels for the component fit of the return value. The argument explain contains human-readable strings describing the data, the model and the summary statistic.

The "..." argument of mincontrast can be used to pass extra arguments to the function theoretical and/or to the optimisation function optim. In this case, the function theoretical should also have a "..." argument and should ignore it (so that it ignores arguments intended for optim).

Value

An object of class "minconfit". There are methods for printing and plotting this object. It contains the following components:

par	Vector of fitted parameter values.
fit	Function value table (object of class "fv") containing the observed values of the summary statistic (observed) and the theoretical values of the summary statistic computed from the fitted model parameters.
opt	The object returned from the optimizer optim.
crtl	List of parameters determining the contrast objective.
info	List of explanatory strings.

Author(s)

Rasmus Waagepetersen <rw@math.auc.dk>, adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

References

Diggle, P.J. and Gratton, R.J. (1984) Monte Carlo methods of inference for implicit statistical models. *Journal of the Royal Statistical Society, series B* **46**, 193 – 212.

Møller, J. and Waagepetersen, R. (2003). Statistical Inference and Simulation for Spatial Point Processes. Chapman and Hall/CRC, Boca Raton.

Pfanzagl, J. (1969). On the measurability and consistency of minimum contrast estimates. *Metrika* **14**, 249–276.

Waagepetersen, R. (2007). An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

kppm, lgcp.estK, matclust.estK, thomas.estK,

---

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 <r.turner@auckland.ac.nz>

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

```
data(longleaf)
miplot(longleaf)
opa <- par(mfrow=c(2,3))
data(cells)
data(japanesepines)
data(redwood)
plot(cells)
plot(japanesepines)
```

```
plot(redwood)
miplot(cells)
miplot(japanesepines)
miplot(redwood)
par(opa)
```

`model.depends` *Identify Covariates Involved in each Model Term*

Description

Given a fitted model (of any kind), identify which of the covariates is involved in each term of the model.

Usage

```
model.depends(object)
model.is.additive(object)
model.covariates(object, fitted=TRUE, offset=TRUE)
has.offset.term(object)
has.offset(object)
```

Arguments

`object`	A fitted model of any kind.
`fitted,offset`	Logical values determining which type of covariates to include.

Details

The `object` can be a fitted model of any kind, including models of the classes `lm`, `glm` and `ppm`.

To be precise, `object` must belong to a class for which there are methods for `formula`, `terms` and `model.matrix`.

The command `model.depends` determines the relationship between the original covariates (the data supplied when `object` was fitted) and the canonical covariates (the columns of the design matrix). It returns a logical matrix, with one row for each canonical covariate, and one column for each of the original covariates, with the `i,j` entry equal to `TRUE` if the `i`th canonical covariate depends on the `j`th original covariate.

If the model formula of `object` includes offset terms (see `offset`), then the return value of `model.depends` also has an attribute `"offset"`. This is a logical value or matrix with one row for each offset term and one column for each of the original covariates, with the `i,j` entry equal to `TRUE` if the `i`th offset term depends on the `j`th original covariate.

The command `model.covariates` returns a character vector containing the names of all (original) covariates that were actually used to fit the model. By default, this includes all covariates that appear in the model formula, including offset terms as well as canonical covariate terms. To omit the offset terms, set `offset=FALSE`. To omit the canonical covariate terms, set `fitted=FALSE`.

The command model.is.additive determines whether the model is additive, in the sense that there is no canonical covariate that depends on two or more original covariates. It returns a logical value.

The command has.offset.term is a faster way to determine whether the model *formula* includes an offset term.

The functions model.depends and has.offset.term only detect offset terms which are present in the model formula. They do not detect numerical offsets in the model object, that were inserted using the offset argument in lm, glm etc. To detect the presence of offsets of both kinds, use has.offset.

Value

A logical value or matrix.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppm, model.matrix

Examples

```
x <- 1:10
y <- 3*x + 2
z <- rep(c(-1,1), 5)
fit <- lm(y ~ poly(x,2) + sin(z))
model.depends(fit)
model.covariates(fit)
model.is.additive(fit)

fitoff1 <- lm(y ~ x + offset(z))
fitoff2 <- lm(y ~ x, offset=z)
has.offset.term(fitoff1)
has.offset(fitoff1)
has.offset.term(fitoff2)
has.offset(fitoff2)
```

---

model.frame.ppm *Extract the Variables in a Point Process Model*

Description

Given a fitted point process model, this function returns a data frame containing all the variables needed to fit the model using the Berman-Turner device.

Usage

```
 ## S3 method for class 'ppm'
model.frame(formula, ...)

 ## S3 method for class 'kppm'
model.frame(formula, ...)

 ## S3 method for class 'dppm'
model.frame(formula, ...)

 ## S3 method for class 'slrm'
model.frame(formula, ...)
```

Arguments

formula	A fitted point process model. An object of class "ppm", "kppm", "slrm", or "dppm".
...	Additional arguments passed to model.frame.glm.

Details

The function model.frame is generic. These functions are method for model.frame for fitted point process models (objects of class "ppm", "kppm", "slrm", or "dppm"). The first argument should be a fitted point process model; it has to be named formula for consistency with the generic function.

The result is a data frame containing all the variables used in fitting the model. The data frame has one row for each quadrature point used in fitting the model. The quadrature scheme can be extracted using quad.ppm.

Value

A data.frame containing all the variables used in the fitted model, plus additional variables specified in .... It has an additional attribute "terms" containing information about the model formula. For details see model.frame.glm.

Author(s)

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

References

Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.

See Also

ppm, kppm, dppm, slrm, model.frame, model.matrix.ppm

Examples

```
fit <- ppm(cells ~ x)
mf <- model.frame(fit)
kfit <- kppm(redwood ~ x, "Thomas")
kmf <- model.frame(kfit)
sfit <- slrm(cells ~ x)
smf <- model.frame(sfit)
```

---

model.images *Compute Images of Constructed Covariates*

Description

For a point process model fitted to spatial point pattern data, this function computes pixel images of the covariates in the design matrix.

Usage

```
  model.images(object, ...)

  ## S3 method for class 'ppm'
model.images(object, W = as.owin(object), ...)

  ## S3 method for class 'kppm'
model.images(object, W = as.owin(object), ...)

  ## S3 method for class 'dppm'
model.images(object, W = as.owin(object), ...)


  ## S3 method for class 'slrm'
model.images(object, ...)
```

Arguments

object	The fitted point process model. An object of class "ppm" or "kppm" or "slrm" or "dppm".
W	A window (object of class "owin") in which the images should be computed. Defaults to the window in which the model was fitted.
...	Other arguments (such as na.action) passed to model.matrix.lm.

Details

This command is similar to model.matrix.ppm except that it computes pixel images of the covariates, instead of computing the covariate values at certain points only.

The object must be a fitted spatial point process model object of class "ppm" (produced by the model-fitting function ppm) or class "kppm" (produced by the fitting function kppm) or class "dppm" (produced by the fitting function dppm) or class "slrm" (produced by slrm).

The spatial covariates required by the model-fitting procedure are computed at every pixel location in the window W. For slrm objects, the covariates are computed on the pixels that were used to fit the model.

Note that the spatial covariates computed here are not necessarily the original covariates that were supplied when fitting the model. Rather, they are the canonical covariates, the covariates that appear in the loglinear representation of the (conditional) intensity and in the columns of the design matrix. For example, they might include dummy or indicator variables for different levels of a factor, depending on the contrasts that are in force.

The pixel resolution is determined by W if W is a mask (that is W$type = "mask"). Otherwise, the pixel resolution is determined by spatstat.options.

The format of the result depends on whether the original point pattern data were marked or unmarked.

- If the original dataset was unmarked, the result is a named list of pixel images (objects of class "im") containing the values of the spatial covariates. The names of the list elements are the names of the covariates determined by model.matrix.lm. The result is also of class "solist" so that it can be plotted immediately.
- If the original dataset was a multitype point pattern, the result is a hyperframe with one column for each possible type of points. Each column is a named list of pixel images (objects of class "im") containing the values of the spatial covariates. The row names of the hyperframe are the names of the covariates determined by model.matrix.lm.

Value

A list (of class "solist") or array (of class "hyperframe") containing pixel images (objects of class "im").

Author(s)

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

See Also

model.matrix.ppm, model.matrix, ppm, ppm.object, dppm, kppm, slrm, im, im.object, plot.solist, spatstat.options

Examples

```
fit <- ppm(cells ~ x)
model.images(fit)
B <- owin(c(0.2, 0.4), c(0.3, 0.8))
model.images(fit, B)
fit2 <- ppm(cells ~ cut(x,3))
model.images(fit2)
fit3 <- slrm(japanesepines ~ x)
```

```
model.images(fit3)
fit4 <- ppm(amacrine ~ marks + x)
model.images(fit4)
```

`model.matrix.mppm` *Extract Design Matrix of Point Process Model for Several Point Patterns*

Description

Given a point process model fitted to a list of point patterns, this function extracts the design matrix.

Usage

```
   ## S3 method for class 'mppm'
model.matrix(object, ..., keepNA=TRUE, separate=FALSE)
```

Arguments

`object`	A point process model fitted to several point patterns. An object of class `"mppm"`.
`...`	Other arguments (such as `na.action`) passed to `model.matrix.lm`.
`keepNA`	Logical. Determines whether rows containing `NA` values will be deleted or retained.
`separate`	Logical value indicating whether to split the model matrix into sub-matrices corresponding to each of the original point patterns.

Details

This command is a method for the generic function `model.matrix`. It extracts the design matrix of a point process model fitted to several point patterns.

The argument `object` must be a fitted point process model (object of class `"mppm"`) produced by the fitting algorithm `mppm`). This represents a point process model that has been fitted to a list of several point pattern datasets. See `mppm` for information.

The result is a matrix with one column for every constructed covariate in the model, and one row for every quadrature point.

If `separate=TRUE` this matrix will be split into sub-matrices corresponding to the original point patterns, and the result will be a list containing these matrices.

Value

A matrix (or list of matrices). Columns of the matrix are canonical covariates in the model.

Author(s)

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

See Also

model.matrix, mppm.

Examples

```
fit <- mppm(Points ~ Image + x, demohyper)
head(model.matrix(fit))
# matrix with three columns: '(Intercept)', 'x' and 'Image'
```

---

model.matrix.ppm *Extract Design Matrix from Point Process Model*

Description

Given a point process model that has been fitted to spatial point pattern data, this function extracts the design matrix of the model.

Usage

```
   ## S3 method for class 'ppm'
model.matrix(object,
                              data=model.frame(object, na.action=NULL),
                              ...,
                              Q=NULL, keepNA=TRUE)

   ## S3 method for class 'kppm'
model.matrix(object,
                              data=model.frame(object, na.action=NULL),
                              ...,
                              Q=NULL, keepNA=TRUE)

   ## S3 method for class 'dppm'
model.matrix(object,
                              data=model.frame(object, na.action=NULL),
                              ...,
                              Q=NULL, keepNA=TRUE)


   ## S3 method for class 'ippm'
model.matrix(object,
                              data=model.frame(object, na.action=NULL),
                              ...,
                              Q=NULL, keepNA=TRUE,
      irregular=FALSE)
```

Arguments

object	The fitted point process model. An object of class "ppm" or "kppm" or "dppm" or "ippm".
data	A model frame, containing the data required for the Berman-Turner device.
Q	A point pattern (class "ppp") or quadrature scheme (class "quad") specifying new locations where the covariates should be computed.
keepNA	Logical. Determines whether rows containing NA values will be deleted or retained.
...	Other arguments (such as na.action) passed to model.matrix.lm.
irregular	Logical value indicating whether to include the irregular score components.

Details

These commands are methods for the generic function model.matrix. They extract the design matrix of a spatial point process model (class "ppm" or "kppm" or "dppm").

More precisely, this command extracts the design matrix of the generalised linear model associated with a spatial point process model.

The object must be a fitted point process model (object of class "ppm" or "kppm" or "dppm") fitted to spatial point pattern data. Such objects are produced by the model-fitting functions ppm, kppm, and dppm.

The methods model.matrix.ppm, model.matrix.kppm, and model.matrix.dppm extract the model matrix for the GLM.

The result is a matrix, with one row for every quadrature point in the fitting procedure, and one column for every constructed covariate in the design matrix.

If there are NA values in the covariates, the argument keepNA determines whether to retain or delete the corresponding rows of the model matrix. The default keepNA=TRUE is to retain them. Note that this differs from the default behaviour of many other methods for model.matrix, which typically delete rows containing NA.

The quadrature points themselves can be extracted using quad.ppm.

Value

A matrix. Columns of the matrix are canonical covariates in the model. Rows of the matrix correspond to quadrature points in the fitting procedure (provided keepNA=TRUE).

Author(s)

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

See Also

model.matrix, model.images, ppm, kppm, dppm, ippm, ppm.object, quad.ppm, residuals.ppm

Examples

```
fit <- ppm(cells ~ x)
head(model.matrix(fit))
model.matrix(fit, Q=runifpoint(5))
kfit <- kppm(redwood ~ x, "Thomas")
m <- model.matrix(kfit)
```

`model.matrix.slrm` *Extract Design Matrix from Spatial Logistic Regression Model*

Description

This function extracts the design matrix of a spatial logistic regression model.

Usage

```
   ## S3 method for class 'slrm'
model.matrix(object, ..., keepNA=TRUE)
```

Arguments

`object`	A fitted spatial logistic regression model. An object of class `"slrm"`.
`...`	Other arguments (such as `na.action`) passed to `model.matrix.lm`.
`keepNA`	Logical. Determines whether rows containing `NA` values will be deleted or retained.

Details

This command is a method for the generic function `model.matrix`. It extracts the design matrix of a spatial logistic regression.

The `object` must be a fitted spatial logistic regression (object of class `"slrm"`). Such objects are produced by the model-fitting function `slrm`.

Usually the result is a matrix with one column for every constructed covariate in the model, and one row for every pixel in the grid used to fit the model.

If `object` was fitted using split pixels (by calling `slrm` using the argument `splitby`) then the matrix has one row for every pixel or half-pixel.

Value

A matrix. Columns of the matrix are canonical covariates in the model.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

model.matrix, model.images, slrm.

Examples

```
fit <- slrm(japanesepines ~x)
head(model.matrix(fit))
# matrix with two columns: '(Intercept)' and 'x'
```

---

mppm *Fit Point Process Model to Several Point Patterns*

Description

Fits a Gibbs point process model to several point patterns simultaneously.

Usage

```
mppm(formula, data, interaction=Poisson(), ...,
     iformula=NULL,
     random=NULL,
     weights=NULL,
     use.gam = FALSE,
     reltol.pql=1e-3,
     gcontrol=list())
```

Arguments

formula	A formula describing the systematic part of the model. Variables in the formula are names of columns in data.
data	A hyperframe (object of class "hyperframe", see hyperframe) containing the point pattern responses and the explanatory variables.
interaction	Interpoint interaction(s) appearing in the model. Either an object of class "interact" describing the point process interaction structure, or a hyperframe (with the same number of rows as data) whose entries are objects of class "interact".
...	Arguments passed to ppm controlling the fitting procedure.
iformula	Optional. A formula (with no left hand side) describing the interaction to be applied to each case. Each variable name in the formula should either be the name of a column in the hyperframe interaction, or the name of a column in the hyperframe data that is a vector or factor.
random	Optional. A formula (with no left hand side) describing a random effect. Variable names in the formula may be any of the column names of data and interaction. The formula must be recognisable to lme.
weights	Optional. Numeric vector of case weights for each row of data.
use.gam	Logical flag indicating whether to fit the model using gam or glm.

`reltol.pql` Relative tolerance for successive steps in the penalised quasi-likelihood algorithm, used when the model includes random effects. The algorithm terminates when the root mean square of the relative change in coefficients is less than `reltol.pql`.

`gcontrol` List of arguments to control the fitting algorithm. Arguments are passed to `glm.control` or `gam.control` or `lmeControl` depending on the kind of model being fitted. If the model has random effects, the arguments are passed to `lmeControl`. Otherwise, if `use.gam=TRUE` the arguments are passed to `gam.control`, and if `use.gam=FALSE` (the default) they are passed to `glm.control`.

Details

This function fits a common point process model to a dataset containing several different point patterns.

It extends the capabilities of the function `ppm` to deal with data such as

- replicated observations of spatial point patterns
- two groups of spatial point patterns
- a designed experiment in which the response from each unit is a point pattern.

The syntax of this function is similar to that of standard R model-fitting functions like `lm` and `glm`. The first argument `formula` is an R formula describing the systematic part of the model. The second argument `data` contains the responses and the explanatory variables. Other arguments determine the stochastic structure of the model.

Schematically, the data are regarded as the results of a designed experiment involving n experimental units. Each unit has a 'response', and optionally some 'explanatory variables' (covariates) describing the experimental conditions for that unit. In this context, *the response from each unit is a point pattern*. The value of a particular covariate for each unit can be either a single value (numerical, logical or factor), or a spatial covariate. A 'spatial' covariate is a quantity that depends on spatial location, for example, the soil acidity or altitude at each location. For the purposes of `mppm`, a spatial covariate must be stored as a pixel image (object of class `"im"`) which gives the values of the covariate at a fine grid of locations.

The argument `data` is a hyperframe (a generalisation of a data frame, see `hyperframe`). This is like a data frame except that the entries can be objects of any class. The hyperframe has one row for each experimental unit, and one column for each variable (response or explanatory variable).

The `formula` should be an R formula. The left hand side of `formula` determines the 'response' variable. This should be a single name, which should correspond to a column in `data`.

The right hand side of `formula` determines the spatial trend of the model. It specifies the linear predictor, and effectively represents the **logarithm** of the spatial trend. Variables in the formula must be the names of columns of `data`, or one of the reserved names

x,y Cartesian coordinates of location

marks Mark attached to point

id which is a factor representing the serial number (1 to n) of the point pattern, i.e. the row number in the data hyperframe.

The column of responses in data must consist of point patterns (objects of class "ppp"). The individual point pattern responses can be defined in different spatial windows. If some of the point patterns are marked, then they must all be marked, and must have the same type of marks.

The scope of models that can be fitted to each pattern is the same as the scope of ppm, that is, Gibbs point processes with interaction terms that belong to a specified list, including for example the Poisson process, Strauss process, Geyer's saturation model, and piecewise constant pairwise interaction models. Additionally, it is possible to include random effects as explained in the section on Random Effects below.

The stochastic part of the model is determined by the arguments interaction and (optionally) iformula.

- In the simplest case, interaction is an object of class "interact", determining the interpoint interaction structure of the point process model, for all experimental units.
- Alternatively, interaction may be a hyperframe, whose entries are objects of class "interact". It should have the same number of rows as data.
  - If interaction consists of only one column, then the entry in row i is taken to be the interpoint interaction for the ith experimental unit (corresponding to the ith row of data).
  - If interaction has more than one column, then the argument iformula is also required. Each row of interaction determines several interpoint interaction structures that might be applied to the corresponding row of data. The choice of interaction is determined by iformula; this should be an R formula, without a left hand side. For example if interaction has two columns called A and B then iformula = ~B indicates that the interpoint interactions are taken from the second column.

Variables in iformula typically refer to column names of interaction. They can also be names of columns in data, but only for columns of numeric, logical or factor values. For example iformula = ~B * group (where group is a column of data that contains a factor) causes the model with interpoint interaction B to be fitted with different interaction parameters for each level of group.

Value

An object of class "mppm" representing the fitted model.

There are methods for print, summary, coef, AIC, anova, fitted, fixef, logLik, plot, predict, ranef, residuals, summary, terms and vcov for this class.

The default methods for update and formula also work on this class.

Random Effects

It is also possible to include random effects in the trend term. The argument random is a formula, with no left-hand side, that specifies the structure of the random effects. The formula should be recognisable to lme (see the description of the argument random for lme).

The names in the formula random may be any of the covariates supplied by data. Additionally the formula may involve the name id, which is a factor representing the serial number (1 to n) of the point pattern in the list X.

Author(s)

Adrian Baddeley, Ida-Maria Sintorn and Leanne Bischoff. Implemented in **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A. and Turner, R. Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42** (2000) 283–322.

Baddeley, A., Bischof, L., Sintorn, I.-M., Haggarty, S., Bell, M. and Turner, R. Analysis of a designed experiment where the response is a spatial point pattern. In preparation.

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

Bell, M. and Grunwald, G. (2004) Mixed models for the analysis of replicated spatial point patterns. *Biostatistics* **5**, 633–648.

See Also

ppm, print.mppm, summary.mppm, coef.mppm,

Examples

```
# Waterstriders data
 H <- hyperframe(Y = waterstriders)
 mppm(Y ~ 1,  data=H)
 mppm(Y ~ 1,  data=H, Strauss(7))
 mppm(Y ~ id, data=H)
 mppm(Y ~ x,  data=H)

# Synthetic data from known model
n <- 10
H <- hyperframe(V=1:n,
                U=runif(n, min=-1, max=1),
                M=factor(letters[1 + (1:n) %% 3]))
H$Z <- setcov(square(1))
H$U <- with(H, as.im(U, as.rectangle(Z)))
H$Y <- with(H, rpoispp(eval.im(exp(2+3*Z))))

fit <- mppm(Y ~Z + U + V, data=H)
```

---

msr *Signed or Vector-Valued Measure*

Description

Defines an object representing a signed measure or vector-valued measure on a spatial domain.

Usage

```
msr(qscheme, discrete, density, check=TRUE)
```

Arguments

qscheme	A quadrature scheme (object of class "quad" usually extracted from a fitted point process model).
discrete	Vector or matrix containing the values (masses) of the discrete component of the measure, for each of the data points in qscheme.
density	Vector or matrix containing values of the density of the diffuse component of the measure, for each of the quadrature points in qscheme.
check	Logical. Whether to check validity of the arguments.

Details

This function creates an object that represents a signed or vector valued *measure* on the two-dimensional plane. It is not normally called directly by the user.

A signed measure is a classical mathematical object (Diestel and Uhl, 1977) which can be visualised as a collection of electric charges, positive and/or negative, spread over the plane. Electric charges may be concentrated at specific points (atoms), or spread diffusely over a region.

An object of class "msr" represents a signed (i.e. real-valued) or vector-valued measure in the **spatstat** package.

Spatial residuals for point process models (Baddeley et al, 2005, 2008) take the form of a real-valued or vector-valued measure. The function residuals.ppm returns an object of class "msr" representing the residual measure. Various other diagnostic tools such as dfbetas.ppm and dffit.ppm also return an object of class "msr".

The function msr would not normally be called directly by the user. It is the low-level creator function that makes an object of class "msr" from raw data.

The first argument qscheme is a quadrature scheme (object of class "quad"). It is typically created by quadscheme or extracted from a fitted point process model using quad.ppm. A quadrature scheme contains both data points and dummy points. The data points of qscheme are used as the locations of the atoms of the measure. All quadrature points (i.e. both data points and dummy points) of qscheme are used as sampling points for the density of the continuous component of the measure.

The argument discrete gives the values of the atomic component of the measure for each *data point* in qscheme. It should be either a numeric vector with one entry for each data point, or a numeric matrix with one row for each data point.

The argument density gives the values of the *density* of the diffuse component of the measure, at each *quadrature point* in qscheme. It should be either a numeric vector with one entry for each quadrature point, or a numeric matrix with one row for each quadrature point.

If both discrete and density are vectors (or one-column matrices) then the result is a signed (real-valued) measure. Otherwise, the result is a vector-valued measure, with the dimension of the vector space being determined by the number of columns in the matrices discrete and/or density. (If one of these is a k-column matrix and the other is a 1-column matrix, then the latter is replicated to k columns).

The class "msr" has methods for print, plot and [. There is also a function Smooth.msr for smoothing a measure.

Value

An object of class "msr".

Guide to using measures

Objects of class "msr", representing measures, are returned by the functions residuals.ppm, dfbetas.ppm, dffit.ppm and possibly by other functions.

There are methods for printing and plotting a measure, along with many other operations, which can be listed by typing methods(class="msr").

The print and summary methods report basic information about a measure, such as the total value of the measure, and the spatial domain on which it is defined.

The plot method displays the measure. It is documented separately in plot.msr.

A measure can be smoothed using Smooth.msr, yielding a pixel image which is sometimes easier to interpret than the plot of the measure itself.

The subset operator [can be used to restrict the measure to a subregion of space, or to extract one of the scalar components of a vector-valued measure. It is documented separately in [.msr.

The total value of a measure, or the value on a subregion, can be obtained using integral.msr. The value of a measure m on a subregion B can be obtained by integral(m, domain=B) or integral(m[B]). The values of a measure m on each tile of a tessellation A can be obtained by integral(m, domain=A).

Some mathematical operations on measures are supported, such as multiplying a measure by a single number, or adding two measures.

Measures can be separated into components in different ways using as.layered.msr, unstack.msr and split.msr.

Internal components of the data structure of an "msr" object can be extracted using with.msr.

Author(s)

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

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.

Baddeley, A., Møller, J. and Pakes, A.G. (2008) Properties of residuals for spatial point processes. *Annals of the Institute of Statistical Mathematics* **60**, 627–649.

Diestel, J. and Uhl, J.J. Jr (1977) *Vector measures*. Providence, RI, USA: American Mathematical Society.

Halmos, P.R. (1950) *Measure Theory*. Van Nostrand.

See Also

plot.msr, Smooth.msr, [.msr, with.msr, split.msr, Ops.msr, measureVariation, measureContinuous.

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X, ~x+y)

rp <- residuals(fit, type="pearson")
rp

rs <- residuals(fit, type="score")
rs
colnames(rs)

# An equivalent way to construct the Pearson residual measure by hand
Q <- quad.ppm(fit)
lambda <- fitted(fit)
slam <- sqrt(lambda)
Z <- is.data(Q)
m <- msr(Q, discrete=1/slam[Z], density = -slam)
m
```

`MultiHard` *The Multitype Hard Core Point Process Model*

Description

Creates an instance of the multitype hard core point process model which can then be fitted to point pattern data.

Usage

```
MultiHard(hradii, types=NULL)
```

Arguments

`hradii`	Matrix of hard core radii
`types`	Optional; vector of all possible types (i.e. the possible levels of the `marks` variable in the data)

Details

This is a multitype version of the hard core process. A pair of points of types i and j must not lie closer than h_{ij} units apart.

The argument `types` need not be specified in normal use. It will be determined automatically from the point pattern data set to which the MultiStrauss interaction is applied, when the user calls `ppm`. However, the user should be confident that the ordering of types in the dataset corresponds to the ordering of rows and columns in the matrix `hradii`.

The matrix `hradii` must be symmetric, with entries which are either positive numbers or `NA`. A value of `NA` indicates that no distance constraint should be applied for this combination of types.

Note that only the hardcore radii are specified in `MultiHard`. The canonical parameters $\log(\beta_j)$ are estimated by `ppm()`, not fixed in `MultiHard()`.

Value

An object of class "interact" describing the interpoint interaction structure of the multitype hard core process with hard core radii $hradii[i, j]$.

Warnings

In order that ppm can fit the multitype hard core model correctly to a point pattern X, this pattern must be marked, with markformat equal to vector and the mark vector marks(X) must be a factor. If the argument types is specified it is interpreted as a set of factor levels and this set must equal levels(marks(X)).

Changed Syntax

Before **spatstat** version 1.37-0, the syntax of this function was different: MultiHard(types=NULL, hradii). The new code attempts to handle the old syntax as well.

Author(s)

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

See Also

ppm, pairwise.family, ppm.object, MultiStrauss, MultiStraussHard, Strauss.

See ragsMultiHard and rmh for simulation.

Examples

```
h <- matrix(c(1,2,2,1), nrow=2,ncol=2)

# prints a sensible description of itself
MultiHard(h)

# Fit the stationary multitype hardcore process to `amacrine'
# with hard core operating only between cells of the same type.
h <- 0.02 * matrix(c(1, NA, NA, 1), nrow=2,ncol=2)
ppm(amacrine ~1, MultiHard(h))
```

MultiStrauss *The Multitype Strauss Point Process Model*

Description

Creates an instance of the multitype Strauss point process model which can then be fitted to point pattern data.

Usage

```
MultiStrauss(radii, types=NULL)
```

Arguments

`radii`	Matrix of interaction radii
`types`	Optional; vector of all possible types (i.e. the possible levels of the `marks` variable in the data)

Details

The (stationary) multitype Strauss process with m types, with interaction radii r_{ij} and parameters β_j and γ_{ij} is the pairwise interaction point process in which each point of type j contributes a factor β_j to the probability density of the point pattern, and a pair of points of types i and j closer than r_{ij} units apart contributes a factor γ_{ij} to the density.

The nonstationary multitype Strauss process is similar except that the contribution of each individual point x_i is a function $\beta(x_i)$ of location and type, rather than a constant beta.

The function `ppm()`, which fits point process models to point pattern data, requires an argument of class `"interact"` describing the interpoint interaction structure of the model to be fitted. The appropriate description of the multitype Strauss process pairwise interaction is yielded by the function `MultiStrauss()`. See the examples below.

The argument `types` need not be specified in normal use. It will be determined automatically from the point pattern data set to which the MultiStrauss interaction is applied, when the user calls `ppm`. However, the user should be confident that the ordering of types in the dataset corresponds to the ordering of rows and columns in the matrix `radii`.

The matrix `radii` must be symmetric, with entries which are either positive numbers or `NA`. A value of `NA` indicates that no interaction term should be included for this combination of types.

Note that only the interaction radii are specified in `MultiStrauss`. The canonical parameters $\log(\beta_j)$ and $\log(\gamma_{ij})$ are estimated by `ppm()`, not fixed in `MultiStrauss()`.

Value

An object of class `"interact"` describing the interpoint interaction structure of the multitype Strauss process with interaction radii $radii[i, j]$.

Warnings

In order that `ppm` can fit the multitype Strauss model correctly to a point pattern `X`, this pattern must be marked, with `markformat` equal to `vector` and the mark vector `marks(X)` must be a factor. If the argument `types` is specified it is interpreted as a set of factor levels and this set must equal `levels(marks(X))`.

Changed Syntax

Before **spatstat** version `1.37-0`, the syntax of this function was different: `MultiStrauss(types=NULL, radii)`. The new code attempts to handle the old syntax as well.

Author(s)

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

, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

ppm, pairwise.family, ppm.object, Strauss, MultiHard

Examples

```
r <- matrix(c(1,2,2,1), nrow=2,ncol=2)
MultiStrauss(r)
# prints a sensible description of itself
r <- 0.03 * matrix(c(1,2,2,1), nrow=2,ncol=2)
X <- amacrine

ppm(X ~1, MultiStrauss(r))
# fit the stationary multitype Strauss process to `amacrine'

# ppm(X ~polynom(x,y,3), MultiStrauss(r, c("off","on")))
# fit a nonstationary multitype Strauss process with log-cubic trend
```

MultiStraussHard *The Multitype/Hard Core Strauss Point Process Model*

Description

Creates an instance of the multitype/hard core Strauss point process model which can then be fitted to point pattern data.

Usage

```
MultiStraussHard(iradii, hradii, types=NULL)
```

Arguments

iradii	Matrix of interaction radii
hradii	Matrix of hard core radii
types	Optional; vector of all possible types (i.e. the possible levels of the marks variable in the data)

Details

This is a hybrid of the multitype Strauss process (see MultiStrauss) and the hard core process (case $\gamma = 0$ of the Strauss process). A pair of points of types i and j must not lie closer than h_{ij} units apart; if the pair lies more than h_{ij} and less than r_{ij} units apart, it contributes a factor γ_{ij} to the probability density.

The argument types need not be specified in normal use. It will be determined automatically from the point pattern data set to which the MultiStraussHard interaction is applied, when the user calls ppm. However, the user should be confident that the ordering of types in the dataset corresponds to the ordering of rows and columns in the matrices iradii and hradii.

The matrices iradii and hradii must be symmetric, with entries which are either positive numbers or NA. A value of NA indicates that no interaction term should be included for this combination of types.

Note that only the interaction radii and hardcore radii are specified in MultiStraussHard. The canonical parameters $\log(\beta_j)$ and $\log(\gamma_{ij})$ are estimated by ppm(), not fixed in MultiStraussHard().

Value

An object of class "interact" describing the interpoint interaction structure of the multitype/hard core Strauss process with interaction radii $iradii[i, j]$ and hard core radii $hradii[i, j]$.

Warnings

In order that ppm can fit the multitype/hard core Strauss model correctly to a point pattern X, this pattern must be marked, with markformat equal to vector and the mark vector marks(X) must be a factor. If the argument types is specified it is interpreted as a set of factor levels and this set must equal levels(marks(X)).

Changed Syntax

Before **spatstat** version 1.37-0, the syntax of this function was different: MultiStraussHard(types=NULL, iradii, hradii). The new code attempts to handle the old syntax as well.

Author(s)

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

, Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

ppm, pairwise.family, ppm.object, MultiStrauss, MultiHard, Strauss

Examples

```
r <- matrix(3, nrow=2,ncol=2)
h <- matrix(c(1,2,2,1), nrow=2,ncol=2)
MultiStraussHard(r,h)
# prints a sensible description of itself
```

```
r <- 0.04 * matrix(c(1,2,2,1), nrow=2,ncol=2)
h <- 0.02 * matrix(c(1,NA,NA,1), nrow=2,ncol=2)
X <- amacrine

fit <- ppm(X ~1, MultiStraussHard(r,h))
# fit stationary multitype hardcore Strauss process to `amacrine'
```

---

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 <r.turner@auckland.ac.nz>

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.

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.

The value of k should be greater than 1 in order to avoid infinite peaks in the intensity estimate around each data point. 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 <r.turner@auckland.ac.nz>

References

NEED REFERENCES. TRY CRESSIE

See Also

density.ppp, intensity 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 `<r.turner@auckland.ac.nz>`

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))
```

npfun *Dummy Function Returns Number of Points*

Description

Returns a summary function which is constant with value equal to the number of points in the point pattern.

Usage

```
npfun(X, ..., r)
```

Arguments

X	Point pattern.
...	Ignored.
r	Vector of values of the distance argument r.

Details

This function is normally not called by the user. Instead it is passed as an argument to the function psst.

Value

Object of class "fv" representing a constant function.

Author(s)

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

See Also

psst

Examples

```
fit0 <- ppm(cells, ~1, nd=10)
v <- psst(fit0, npfun)
```

objsurf *Objective Function Surface*

Description

For a model that was fitted by optimisation, compute the values of the objective function in a neighbourhood of the optimal value.

Usage

```
objsurf(x, ...)

## S3 method for class 'dppm'
objsurf(x, ..., ngrid = 32, xlim=NULL, ylim=NULL,
                                ratio = 1.5, verbose = TRUE)

## S3 method for class 'kppm'
objsurf(x, ..., ngrid = 32, xlim=NULL, ylim=NULL,
                                ratio = 1.5, verbose = TRUE)

## S3 method for class 'minconfit'
objsurf(x, ..., ngrid = 32, xlim=NULL, ylim=NULL,
                                ratio = 1.5, verbose = TRUE)
```

Arguments

x	Some kind of model that was fitted by finding the optimal value of an objective function. An object of class "dppm", "kppm" or "minconfit".
...	Extra arguments are usually ignored.
ngrid	Number of grid points to evaluate along each axis. Either a single integer, or a pair of integers. For example ngrid=32 would mean a 32 * 32 grid.
xlim,ylim	Optional. Numeric vectors of length 2, specifying the limits for the two parameters to be considered.
ratio	Number greater than 1 determining the range of parameter values to be considered. If the optimal parameter value is opt then the objective function will be evaluated for values between opt/ratio and opt * ratio.
verbose	Logical value indicating whether to print progress reports.

Details

The object x should be some kind of model that was fitted by maximising or minimising the value of an objective function. The objective function will be evaluated on a grid of values of the model parameters.

Currently the following types of objects are accepted:

- an object of class "dppm" representing a determinantal point process. See dppm.

- an object of class "kppm" representing a cluster point process or Cox point process. See kppm.
- an object of class "minconfit" representing a minimum-contrast fit between a summary function and its theoretical counterpart. See mincontrast.

The result is an object of class "objsurf" which can be printed and plotted: see methods.objsurf.

Value

An object of class "objsurf" which can be printed and plotted. Essentially a list containing entries x, y, z giving the parameter values and objective function values.

There are methods for plot, print, summary, image, contour and persp.

Author(s)

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

See Also

methods.objsurf, kppm, mincontrast

Examples

```
fit <- kppm(redwood ~ 1, "Thomas")
os <- objsurf(fit)

if(interactive()) {
  plot(os)
  contour(os, add=TRUE)
  persp(os)
}
```

Ops.msr *Arithmetic Operations on Measures*

Description

These group generic methods for the class "msr" allow the arithmetic operators +, -, * and / to be applied directly to measures.

Usage

```
## S3 methods for group generics have prototypes:
Ops(e1, e2)
```

Arguments

e1, e2	objects of class "msr".

Details

Arithmetic operators on a measure A are only defined in some cases. The arithmetic operator is effectively applied to the value of A(W) for every spatial domain W. If the result is a measure, then this operation is valid.

If A is a measure (object of class "msr") then the operations -A and +A are defined.

If A and B are measures with the same dimension (i.e. both are scalar-valued, or both are k-dimensional vector-valued) then A + B and A - B are defined.

If A is a measure and z is a numeric value, then A * z and A / z are defined, and z * A is defined.

Value

Another measure (object of class "msr").

Author(s)

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

See Also

with.msr

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X, ~x+y)
rp <- residuals(fit, type="pearson")
rp

-rp
2 * rp
rp /2

rp - rp

rr <- residuals(fit, type="raw")
rp - rr
```

Ord *Generic Ord Interaction model*

Description

Creates an instance of an Ord-type interaction point process model which can then be fitted to point pattern data.

Usage

```
Ord(pot, name)
```

Arguments

pot	An S language function giving the user-supplied interaction potential.
name	Character string.

Details

Ord's point process model (Ord, 1977) is a Gibbs point process of infinite order. Each point x_i in the point pattern x contributes a factor $g(a_i)$ where $a_i = a(x_i, x)$ is the area of the tile associated with x_i in the Dirichlet tessellation of x.

Ord (1977) proposed fitting this model to forestry data when $g(a)$ has a simple "threshold" form. That model is implemented in our function OrdThresh. The present function Ord implements the case of a completely general Ord potential $g(a)$ specified as an S language function pot.

This is experimental.

Value

An object of class "interact" describing the interpoint interaction structure of a point process.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.

Ord, J.K. (1977) Contribution to the discussion of Ripley (1977).

Ord, J.K. (1978) How many trees in a forest? *Mathematical Scientist* **3**, 23–33.

Ripley, B.D. (1977) Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, Series B*, **39**, 172 – 212.

See Also

ppm, ppm.object, OrdThresh

`ord.family` *Ord Interaction Process Family*

Description

An object describing the family of all Ord interaction point processes

Details

Advanced Use Only!

This structure would not normally be touched by the user. It describes the family of point process models introduced by Ord (1977).

If you need to create a specific Ord-type model for use in analysis, use the function `OrdThresh` or `Ord`.

Value

Object of class `"isf"`, see `isf.object`.

Author(s)

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

and Rolf Turner `<r.turner@auckland.ac.nz>`

References

Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.

Ord, J.K. (1977) Contribution to the discussion of Ripley (1977).

Ord, J.K. (1978) How many trees in a forest? *Mathematical Scientist* **3**, 23–33.

Ripley, B.D. (1977) Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, Series B*, **39**, 172 – 212.

See Also

`pairwise.family`, `pairsat.family`, `Ord`, `OrdThresh`

`OrdThresh` *Ord's Interaction model*

Description

Creates an instance of Ord's point process model which can then be fitted to point pattern data.

Usage

```
OrdThresh(r)
```

Arguments

`r`	Positive number giving the threshold value for Ord's model.

Details

Ord's point process model (Ord, 1977) is a Gibbs point process of infinite order. Each point x_i in the point pattern x contributes a factor $g(a_i)$ where $a_i = a(x_i, x)$ is the area of the tile associated with x_i in the Dirichlet tessellation of x. The function g is simply $g(a) = 1$ if $a \geq r$ and $g(a) = \gamma < 1$ if $a < r$, where r is called the threshold value.

This function creates an instance of Ord's model with a given value of r. It can then be fitted to point process data using `ppm`.

Value

An object of class `"interact"` describing the interpoint interaction structure of a point process.

Author(s)

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

and Rolf Turner `<r.turner@auckland.ac.nz>`

References

Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.

Ord, J.K. (1977) Contribution to the discussion of Ripley (1977).

Ord, J.K. (1978) How many trees in a forest? *Mathematical Scientist* **3**, 23–33.

Ripley, B.D. (1977) Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, Series B*, **39**, 172 – 212.

See Also

`ppm`, `ppm.object`

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 <r.turner@auckland.ac.nz> 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))
```

PairPiece *The Piecewise Constant Pairwise Interaction Point Process Model*

Description

Creates an instance of a pairwise interaction point process model with piecewise constant potential function. The model can then be fitted to point pattern data.

Usage

```
PairPiece(r)
```

Arguments

r	vector of jump points for the potential function

Details

A pairwise interaction point process in a bounded region is a stochastic point process with probability density of the form

$$f(x_1, \ldots, x_n) = \alpha \prod_i b(x_i) \prod_{i<j} h(x_i, x_j)$$

where $x_1, \ldots, x_n$ represent the points of the pattern. The first product on the right hand side is over all points of the pattern; the second product is over all unordered pairs of points of the pattern.

Thus each point x_i of the pattern contributes a factor $b(x_i)$ to the probability density, and each pair of points x_i, x_j contributes a factor $h(x_i, x_j)$ to the density.

The pairwise interaction term $h(u, v)$ is called *piecewise constant* if it depends only on the distance between u and v, say $h(u, v) = H(||u - v||)$, and H is a piecewise constant function (a function which is constant except for jumps at a finite number of places). The use of piecewise constant interaction terms was first suggested by Takacs (1986).

The function `ppm()`, which fits point process models to point pattern data, requires an argument of class `"interact"` describing the interpoint interaction structure of the model to be fitted. The appropriate description of the piecewise constant pairwise interaction is yielded by the function `PairPiece()`. See the examples below.

The entries of `r` must be strictly increasing, positive numbers. They are interpreted as the points of discontinuity of H. It is assumed that $H(s) = 1$ for all $s > r_{max}$ where r_{max} is the maximum value in `r`. Thus the model has as many regular parameters (see `ppm`) as there are entries in `r`. The i-th regular parameter θ_i is the logarithm of the value of the interaction function H on the interval $[r_{i-1}, r_i)$.

If `r` is a single number, this model is similar to the Strauss process, see `Strauss`. The difference is that in `PairPiece` the interaction function is continuous on the right, while in `Strauss` it is continuous on the left.

The analogue of this model for multitype point processes has not yet been implemented.

Value

An object of class `"interact"` describing the interpoint interaction structure of a point process. The process is a pairwise interaction process, whose interaction potential is piecewise constant, with jumps at the distances given in the vector r.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

References

Takacs, R. (1986) Estimator for the pair potential of a Gibbsian point process. *Statistics* **17**, 429–433.

See Also

ppm, pairwise.family, ppm.object, Strauss rmh.ppm

Examples

```
PairPiece(c(0.1,0.2))
# prints a sensible description of itself
data(cells)

ppm(cells ~1, PairPiece(r = c(0.05, 0.1, 0.2)))
# fit a stationary piecewise constant pairwise interaction process

# ppm(cells ~polynom(x,y,3), PairPiece(c(0.05, 0.1)))
# nonstationary process with log-cubic polynomial trend
```

---

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 <r.turner@auckland.ac.nz> 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)
```

`pairsat.family` *Saturated Pairwise Interaction Point Process Family*

Description

An object describing the Saturated Pairwise Interaction family of point process models

Details

Advanced Use Only!

This structure would not normally be touched by the user. It describes the "saturated pairwise interaction" family of point process models.

If you need to create a specific interaction model for use in spatial pattern analysis, use the function `Saturated()` or the two existing implementations of models in this family, `Geyer()` and `SatPiece()`.

Geyer (1999) introduced the "saturation process", a modification of the Strauss process in which the total contribution to the potential from each point (from its pairwise interaction with all other points) is trimmed to a maximum value c. This model is implemented in the function `Geyer()`.

The present class `pairsat.family` is the extension of this saturation idea to all pairwise interactions. Note that the resulting models are no longer pairwise interaction processes - they have interactions of infinite order.

`pairsat.family` is an object of class `"isf"` containing a function `pairwise$eval` for evaluating the sufficient statistics of any saturated pairwise interaction point process model in which the original pair potentials take an exponential family form.

Value

Object of class `"isf"`, see `isf.object`.

Author(s)

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

and Rolf Turner `<r.turner@auckland.ac.nz>`

References

Geyer, C.J. (1999) Likelihood Inference for Spatial Point Processes. Chapter 3 in O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. Van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*, Chapman and Hall / CRC, Monographs on Statistics and Applied Probability, number 80. Pages 79–140.

See Also

Geyer to create the Geyer saturation process.

SatPiece to create a saturated process with piecewise constant pair potential.

Saturated to create a more general saturation model.

Other families: inforder.family, ord.family, pairwise.family.

`Pairwise` *Generic Pairwise Interaction model*

Description

Creates an instance of a pairwise interaction point process model which can then be fitted to point pattern data.

Usage

```
Pairwise(pot, name, par, parnames, printfun)
```

Arguments

`pot`	An R language function giving the user-supplied pairwise interaction potential.
`name`	Character string.
`par`	List of numerical values for irregular parameters
`parnames`	Vector of names of irregular parameters
`printfun`	Do not specify this argument: for internal use only.

Details

This code constructs a member of the pairwise interaction family pairwise.family with arbitrary pairwise interaction potential given by the user.

Each pair of points in the point pattern contributes a factor $h(d)$ to the probability density, where d is the distance between the two points. The factor term $h(d)$ is

$$h(d) = \exp(-\theta \mathrm{pot}(d))$$

provided $\mathrm{pot}(d)$ is finite, where θ is the coefficient vector in the model.

The function `pot` must take as its first argument a matrix of interpoint distances, and evaluate the potential for each of these distances. The result must be either a matrix with the same dimensions as its input, or an array with its first two dimensions the same as its input (the latter case corresponds to a vector-valued potential).

If irregular parameters are present, then the second argument to `pot` should be a vector of the same type as `par` giving those parameter values.

The values returned by `pot` may be finite numeric values, or `-Inf` indicating a hard core (that is, the corresponding interpoint distance is forbidden). We define $h(d) = 0$ if $\mathrm{pot}(d) = -\infty$. Thus, a potential value of minus infinity is *always* interpreted as corresponding to $h(d) = 0$, regardless of the sign and magnitude of θ.

Value

An object of class "interact" describing the interpoint interaction structure of a point process.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppm, pairwise.family, ppm.object

Examples

```
#This is the same as StraussHard(r=0.7,h=0.05)
strpot <- function(d,par) {
      r <- par$r
      h <- par$h
      value <- (d <= r)
      value[d < h] <- -Inf
      value
}
mySH <- Pairwise(strpot, "StraussHard process", list(r=0.7,h=0.05),
        c("interaction distance r", "hard core distance h"))
data(cells)
ppm(cells, ~ 1, mySH, correction="isotropic")

# Fiksel (1984) double exponential interaction
# see Stoyan, Kendall, Mecke 1987 p 161

fikspot <- function(d, par) {
   r <- par$r
   h <- par$h
   zeta <- par$zeta
   value <- exp(-zeta * d)
   value[d < h] <- -Inf
   value[d > r] <- 0
   value
}
Fiksel <- Pairwise(fikspot, "Fiksel double exponential process",
                   list(r=3.5, h=1, zeta=1),
                   c("interaction distance r",
                     "hard core distance h",
                     "exponential coefficient zeta"))
data(spruces)
fit <- ppm(unmark(spruces), ~1, Fiksel, rbord=3.5)
fit
plot(fitin(fit), xlim=c(0,4))
coef(fit)
# corresponding values obtained by Fiksel (1984) were -1.9 and -6.0
```

pairwise.family *Pairwise Interaction Process Family*

Description

An object describing the family of all pairwise interaction Gibbs point processes.

Details

Advanced Use Only!

This structure would not normally be touched by the user. It describes the pairwise interaction family of point process models.

If you need to create a specific pairwise interaction model for use in modelling, use the function Pairwise or one of the existing functions listed below.

Anyway, pairwise.family is an object of class "isf" containing a function pairwise.family$eval for evaluating the sufficient statistics of any pairwise interaction point process model taking an exponential family form.

Value

Object of class "isf", see isf.object.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

Other families: pairsat.family, ord.family, inforder.family.

Pairwise interactions: Poisson, Pairwise, PairPiece, Fiksel, Hardcore, LennardJones, MultiHard, MultiStrauss, MultiStraussHard, Strauss, StraussHard, Softcore.

Other interactions: AreaInter, Geyer, Saturated, Ord, OrdThresh.

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 <r.turner@auckland.ac.nz> 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)
```

parameters *Extract Model Parameters in Understandable Form*

Description

Given a fitted model of some kind, this function extracts all the parameters needed to specify the model, and returns them as a list.

Usage

```
parameters(model, ...)

## S3 method for class 'dppm'
parameters(model, ...)

## S3 method for class 'kppm'
parameters(model, ...)

## S3 method for class 'slrm'
parameters(model, ...)

## S3 method for class 'ppm'
parameters(model, ...)

## S3 method for class 'profilepl'
parameters(model, ...)

## S3 method for class 'fii'
parameters(model, ...)

## S3 method for class 'interact'
parameters(model, ...)
```

Arguments

model	A fitted model of some kind.
...	Arguments passed to methods.

Details

The argument model should be a fitted model of some kind. This function extracts all the parameters that would be needed to specify the model, and returns them as a list.

The function parameters is generic, with methods for class "ppm", "kppm", "dppm" and "profilepl" and other classes.

Value

A named list, whose format depends on the fitted model.

Author(s)

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

See Also

coef

Examples

```
parameters(Strauss(0.1))
fit1 <- ppm(cells ~ x, Strauss(0.1))
parameters(fit1)
fit2 <- kppm(redwood ~ x, "Thomas")
parameters(fit2)
```

---

parres *Partial Residuals for Point Process Model*

Description

Computes the smoothed partial residuals, a diagnostic for transformation of a covariate in a Poisson point process model.

Usage

```
parres(model, covariate, ...,
       smooth.effect=FALSE, subregion=NULL,
       bw = "nrd0", adjust=1, from = NULL, to = NULL, n = 512,
       bw.input = c("points", "quad"), bw.restrict=FALSE, covname)
```

Arguments

model	Fitted point process model (object of class "ppm").
covariate	The covariate of interest. Either a character string matching the name of one of the canonical covariates in the model, or one of the names "x" or "y" referring to the Cartesian coordinates, or one of the names of the covariates given when model was fitted, or a pixel image (object of class "im") or function(x,y) supplying the values of a covariate at any location. If the model depends on only one covariate, then this covariate is the default; otherwise a covariate must be specified.
smooth.effect	Logical. Determines the choice of algorithm. See Details.

subregion	Optional. A window (object of class "owin") specifying a subset of the spatial domain of the data. The calculation will be confined to the data in this subregion.
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.
...	Additional arguments passed to density.default.
bw.input	Character string specifying the input data used for automatic bandwidth selection.
bw.restrict	Logical value, specifying whether bandwidth selection is performed using data from the entire spatial domain or from the subregion.
covname	Optional. Character string to use as the name of the covariate.

Details

This command computes the smoothed partial residual diagnostic (Baddeley, Chang, Song and Turner, 2012) for the transformation of a covariate in a Poisson point process model.

The argument model must be a fitted Poisson point process model.

The diagnostic works in two different ways:

Canonical covariate: The argument covariate may be a character string which is the name of one of the *canonical covariates* in the model. The canonical covariates are the functions Z_j that appear in the expression for the Poisson point process intensity

$$\lambda(u) = \exp(\beta_1 Z_1(u) + \ldots + \beta_p Z_p(u))$$

at spatial location u. Type names(coef(model)) to see the names of the canonical covariates in model. If the selected covariate is Z_j, then the diagnostic plot concerns the model term $\beta_j Z_j(u)$. The plot shows a smooth estimate of a function $h(z)$ that should replace this linear term, that is, $\beta_j Z_j(u)$ should be replaced by $h(Z_j(u))$. The linear function is also plotted as a dotted line.

New covariate: If the argument covariate is a pixel image (object of class "im") or a function(x,y), it is assumed to provide the values of a covariate that is not present in the model. Alternatively covariate can be the name of a covariate that was supplied when the model was fitted (i.e. in the call to ppm) but which does not feature in the model formula. In either case we speak of a new covariate $Z(u)$. If the fitted model intensity is $\lambda(u)$ then we consider modifying this to $\lambda(u)\exp(h(Z(u)))$ where $h(z)$ is some function. The diagnostic plot shows an estimate of $h(z)$. **Warning: in this case the diagnostic is not theoretically justified. This option is provided for research purposes.**

Alternatively covariate can be one of the character strings "x" or "y" signifying the Cartesian coordinates. The behaviour here depends on whether the coordinate was one of the canonical covariates in the model.

If there is more than one canonical covariate in the model that depends on the specified covariate, then the covariate effect is computed using all these canonical covariates. For example in a log-quadratic model which includes the terms x and I(x^2), the quadratic effect involving both these terms will be computed.

There are two choices for the algorithm. If `smooth.effect=TRUE`, the fitted covariate effect (according to `model`) is added to the point process residuals, then smoothing is applied to these values. If `smooth.effect=FALSE`, the point process residuals are smoothed first, and then the fitted covariate effect is added to the result.

The smoothing bandwidth is controlled by the arguments `bw`, `adjust`, `bw.input` and `bw.restrict`. If `bw` is a numeric value, then the bandwidth is taken to be `adjust * bw`. If `bw` is a string representing a bandwidth selection rule (recognised by `density.default`) then the bandwidth is selected by this rule.

The data used for automatic bandwidth selection are specified by `bw.input` and `bw.restrict`. If `bw.input="points"` (the default) then bandwidth selection is based on the covariate values at the points of the original point pattern dataset to which the model was fitted. If `bw.input="quad"` then bandwidth selection is based on the covariate values at every quadrature point used to fit the model. If `bw.restrict=TRUE` then the bandwidth selection is performed using only data from inside the `subregion`.

Value

A function value table (object of class `"fv"`) containing the values of the smoothed partial residual, the estimated variance, and the fitted effect of the covariate. Also belongs to the class `"parres"` which has methods for `print` and `plot`.

Slow computation

In a large dataset, computation can be very slow if the default settings are used, because the smoothing bandwidth is selected automatically. To avoid this, specify a numerical value for the bandwidth `bw`. One strategy is to use a coarser subset of the data to select `bw` automatically. The selected bandwidth can be read off the print output for `parres`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz>, Ya-Mei Chang and Yong Song.

References

Baddeley, A., Chang, Y.-M., Song, Y. and Turner, R. (2013) Residual diagnostics for covariate effects in spatial point process models. *Journal of Computational and Graphical Statistics*, **22**, 886–905.

See Also

`addvar`, `rhohat`, `rho2hat`

Examples

```
X <-  rpoispp(function(x,y){exp(3+x+2*x^2)})
model <- ppm(X ~x+y)
tra <- parres(model, "x")
plot(tra)
tra
```

```
plot(parres(model, "x", subregion=square(0.5)))
model2 <- ppm(X ~x+I(x^2)+y)
plot(parres(model2, "x"))
Z <- setcov(owin())
plot(parres(model2, Z))

#' when the model involves only one covariate
modelb <- ppm(bei ~ elev + I(elev^2), data=bei.extra)
plot(parres(modelb))
```

---

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 <r.turner@auckland.ac.nz> 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 <r.turner@auckland.ac.nz>

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 <r.turner@auckland.ac.nz> 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, kernel="epanechnikov", bw=NULL,
                    stoyan=0.15,
                    correction=c("translate", "Ripley"),
                    divisor = c("r", "d"),
                    var.approx = FALSE,
                    domain=NULL,
                    ratio=FALSE, close=NULL)
```

Arguments

X	A point pattern (object of class "ppp").
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. Either a single numeric value giving the standard deviation of the kernel, or a character string specifying a bandwidth selection rule recognised by density.default. If bw is missing or NULL, the default value is computed using Stoyan's rule of thumb: see Details.
...	Other arguments passed to the kernel density estimation function density.default.
stoyan	Coefficient for Stoyan's bandwidth selection rule; see Details.
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". See Details.
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: 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):

- 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.

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 `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.

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.

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 <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk> and Martin Hazelton.

References

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

`Kest`, `pcf`, `density.default`, `bw.stoyan`, `bw.pcf`, `lohboot`.

Examples

```
X <- simdat

p <- pcf(X)
plot(p, main="pair correlation function for X")
# indicates inhibition at distances r < 0.3

pd <- pcf(X, divisor="d")

# compare estimates
plot(p, cbind(iso, theo) ~ r, col=c("blue", "red"),
       ylim.covers=0, main="", lwd=c(2,1), lty=c(1,3), legend=FALSE)
plot(pd, iso ~ r, col="green", lwd=2, add=TRUE)
legend("center", col=c("blue", "green"), lty=1, lwd=2,
       legend=c("divisor=r","divisor=d"))

# calculate approximate variance and show POINTWISE confidence bands
pv <- pcf(X, var.approx=TRUE)
plot(pv, cbind(iso, iso+2*sqrt(v), iso-2*sqrt(v)) ~ r)
```

---

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"))
```

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.

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 <r.turner@auckland.ac.nz>

See Also

Mark connection function markconnect.

Multitype pair correlation pcfdot, pcfmulti.

Pair correlation pcf,pcf.ppp.

Kcross

Examples

```
data(amacrine)
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, stoyan=0.15,
               correction = c("isotropic", "Ripley", "translate"),
               sigma = NULL, 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 smoothing kernel, passed to `density.default`.
`bw`	Bandwidth for smoothing kernel, passed to `density.default`.
`...`	Other arguments passed to the 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 kernel smoothing.

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.

If the arguments lambdaI and lambdaJ are missing or null, they are estimated from X by kernel smoothing using a leave-one-out estimator.

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 <r.turner@auckland.ac.nz>

See Also

pcf.ppp, pcfinhom, pcfcross, pcfdot.inhom

Examples

```
data(amacrine)
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"))
```

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.

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 <r.turner@auckland.ac.nz>

See Also

Mark connection function markconnect.

Multitype pair correlation pcfcross, pcfmulti.

Pair correlation pcf,pcf.ppp.

Kdot

Examples

```
data(amacrine)
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, stoyan=0.15,
                correction = c("isotropic", "Ripley", "translate"),
                sigma = NULL, 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 smoothing kernel, passed to density.default.
bw	Bandwidth for smoothing kernel, passed to density.default.
...	Other arguments passed to the 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 lambdadot is estimated by kernel smoothing.

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.

If the arguments lambdaI and lambdadot are missing or null, they are estimated from X by kernel smoothing using a leave-one-out estimator.

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 <r.turner@auckland.ac.nz>

See Also

pcf.ppp, pcfinhom, pcfdot, pcfcross.inhom

Examples

```
data(amacrine)
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,
         kernel = "epanechnikov", bw = NULL, stoyan = 0.15,
         correction = c("translate", "Ripley"),
         divisor = c("r", "d"),
         renormalise = TRUE, normpower=1,
         update = TRUE, leaveoneout = TRUE,
         reciplambda = NULL,
         sigma = NULL, varcov = NULL, 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.
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. Either a single numeric value, or a character string specifying a bandwidth selection rule recognised by density.default. If bw is missing or NULL, the default value is computed using Stoyan's rule of thumb: see bw.stoyan.
...	Other arguments passed to the kernel density estimation function density.default.
stoyan	Coefficient for Stoyan's bandwidth selection rule; see bw.stoyan.
correction	Character string or character vector specifying the choice of edge correction. See Kest for explanation and options.

divisor	Choice of divisor in the estimation formula: either "r" (the default) or "d". See pcf.ppp.
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.
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.ppp.

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 inhomogeneous pair correlation function $g_{\mathrm{inhom}}(r)$ has been estimated
theo	vector of values equal to 1, the theoretical value of $g_{\mathrm{inhom}}(r)$ for the Poisson process
trans	vector of values of $g_{\mathrm{inhom}}(r)$ estimated by translation correction
iso	vector of values of $g_{\mathrm{inhom}}(r)$ estimated by Ripley isotropic correction

as required.

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 <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

pcf, pcf.ppp, bw.stoyan, bw.pcf, Kinhom

Examples

```
data(residualspaper)
X <- residualspaper$Fig4b
plot(pcfinhom(X, stoyan=0.2, sigma=0.1))
fit <- ppm(X, ~polynom(x,y,2))
plot(pcfinhom(X, lambda=fit, normpower=2))
```

---

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")
```

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.

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 <r.turner@auckland.ac.nz>

See Also

pcfcross, pcfdot, pcf.ppp.

Examples

```
adult <- (marks(longleaf) >= 30)
juvenile <- !adult
p <- pcfmulti(longleaf, adult, juvenile)
```

Penttinen *Penttinen Interaction*

Description

Creates an instance of the Penttinen pairwise interaction point process model, which can then be fitted to point pattern data.

Usage

```
Penttinen(r)
```

Arguments

r	circle radius

Details

Penttinen (1984, Example 2.1, page 18), citing Cormack (1979), described the pairwise interaction point process with interaction factor

$$h(d) = e^{\theta A(d)} = \gamma^{A(d)}$$

between each pair of points separated by a distance \$d\$. Here $A(d)$ is the area of intersection between two discs of radius r separated by a distance d, normalised so that $A(0) = 1$.

The scale of interaction is controlled by the disc radius r: two points interact if they are closer than $2r$ apart. The strength of interaction is controlled by the canonical parameter θ, which must be less than or equal to zero, or equivalently by the parameter $\gamma = e^{\theta}$, which must lie between 0 and 1.

The potential is inhibitory, i.e.\ this model is only appropriate for regular point patterns. For $\gamma = 0$ the model is a hard core process with hard core diameter $2r$. For $\gamma = 1$ the model is a Poisson process.

The irregular parameter r must be given in the call to `Penttinen`, while the regular parameter θ will be estimated.

This model can be considered as a pairwise approximation to the area-interaction model `AreaInter`.

Value

An object of class `"interact"` describing the interpoint interaction structure of a point process.

Author(s)

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

References

Cormack, R.M. (1979) Spatial aspects of competition between individuals. Pages 151–212 in *Spatial and Temporal Analysis in Ecology*, eds. R.M. Cormack and J.K. Ord, International Co-operative Publishing House, Fairland, MD, USA.

Penttinen, A. (1984) *Modelling Interaction in Spatial Point Patterns: Parameter Estimation by the Maximum Likelihood Method.* Jyväskylä Studies in Computer Science, Economics and Statistics **7**, University of Jyväskylä, Finland.

See Also

`ppm`, `ppm.object`, `Pairwise`, `AreaInter`.

Examples

```
fit <- ppm(cells ~ 1, Penttinen(0.07))
fit
reach(fit) # interaction range is circle DIAMETER
```

`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 <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

berman.test

Examples

```
# 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 <r.turner@auckland.ac.nz>

See Also

cdf.test

Examples

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

# 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.dppm` *Plot a fitted determinantal point process*

Description

Plots a fitted determinantal point process model, displaying the fitted intensity and the fitted summary function.

Usage

```
  ## S3 method for class 'dppm'
plot(x, ..., what=c("intensity", "statistic"))
```

Arguments

`x`	Fitted determinantal point process model. An object of class `"dppm"`.
`...`	Arguments passed to `plot.ppm` and `plot.fv` to control the plot.
`what`	Character vector determining what will be plotted.

Details

This is a method for the generic function `plot` for the class `"dppm"` of fitted determinantal point process models.

The argument x should be a determinantal point process model (object of class "dppm") obtained using the function dppm.

The choice of plots (and the order in which they are displayed) is controlled by the argument what. The options (partially matched) are "intensity" and "statistic".

This command is capable of producing two different plots:

what="intensity" specifies the fitted intensity of the model, which is plotted using plot.ppm. By default this plot is not produced for stationary models.

what="statistic" specifies the empirical and fitted summary statistics, which are plotted using plot.fv. This is only meaningful if the model has been fitted using the Method of Minimum Contrast, and it is turned off otherwise.

Value

Null.

Author(s)

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

See Also

dppm, plot.ppm,

Examples

```
fit <- dppm(swedishpines ~ x + y, dppGauss, method="c")
plot(fit)
```

---

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 <r.turner@auckland.ac.nz>

See Also

envelope, plot.fv

Examples

```
data(cells)
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 <r.turner@auckland.ac.nz>

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, 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)
```

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.
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.

Details

This is the plot method for the class "fv".

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 data.

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.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>
and Rolf Turner <r.turner@auckland.ac.nz>

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)
```

`plot.influence.ppm` *Plot Influence Measure*

Description

Plots an influence measure that has been computed by `influence.ppm`.

Usage

```
 ## S3 method for class 'influence.ppm'
plot(x, ..., multiplot=TRUE)
```

Arguments

`x`	Influence measure (object of class `"influence.ppm"`) computed by `influence.ppm`.
`...`	Arguments passed to `plot.ppp` to control the plotting.
`multiplot`	Logical value indicating whether it is permissible to plot more than one panel. This happens if the original point process model is multitype.

Details

This is the plot method for objects of class `"influence.ppm"`. These objects are computed by the command `influence.ppm`.

For a point process model fitted by maximum likelihood or maximum pseudolikelihood (the default), influence values are associated with the data points. The display shows circles centred at the data points with radii proportional to the influence values. If the original data were a multitype point pattern, then if `multiplot=TRUE` (the default), there is one such display for each possible type of point, while if `multiplot=FALSE` there is a single plot combining all data points regardless of type.

For a model fitted by logistic composite likelihood (`method="logi"` in `ppm`) influence values are associated with the data points and also with the dummy points used to fit the model. The display consist of two panels, for the data points and dummy points respectively, showing circles with radii proportional to the influence values. If the original data were a multitype point pattern, then if `multiplot=TRUE` (the default), there is one pair of panels for each possible type of point, while if `multiplot=FALSE` there is a single plot combining all data and dummy points regardless of type.

Use the argument `clipwin` to restrict the plot to a subset of the full data.

Value

None.

Author(s)

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

References

Baddeley, A. and Chang, Y.M. and Song, Y. (2013) Leverage and influence diagnostics for spatial point process models. *Scandinavian Journal of Statistics* **40**, 86–104.

See Also

`influence.ppm`

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X, ~x+y)
plot(influence(fit))
```

`plot.kppm` *Plot a fitted cluster point process*

Description

Plots a fitted cluster point process model, displaying the fitted intensity and the fitted K-function.

Usage

```
  ## S3 method for class 'kppm'
plot(x, ...,
                      what=c("intensity", "statistic", "cluster"),
                      pause=interactive(),
                      xname)
```

Arguments

`x`	Fitted cluster point process model. An object of class `"kppm"`.
`...`	Arguments passed to `plot.ppm` and `plot.fv` to control the plot.
`what`	Character vector determining what will be plotted.
`pause`	Logical value specifying whether to pause between plots.
`xname`	Optional. Character string. The name of the object `x` for use in the title of the plot.

Details

This is a method for the generic function `plot` for the class `"kppm"` of fitted cluster point process models.

The argument `x` should be a cluster point process model (object of class `"kppm"`) obtained using the function `kppm`.

The choice of plots (and the order in which they are displayed) is controlled by the argument `what`. The options (partially matched) are `"intensity"`, `"statistic"` and `"cluster"`.

This command is capable of producing three different plots:

what="intensity" specifies the fitted intensity of the model, which is plotted using plot.ppm. By default this plot is not produced for stationary models.

what="statistic" specifies the empirical and fitted summary statistics, which are plotted using plot.fv. This is only meaningful if the model has been fitted using the Method of Minimum Contrast, and it is turned off otherwise.

what="cluster" specifies a fitted cluster, which is computed by clusterfield and plotted by plot.im. It is only meaningful for Poisson cluster (incl. Neyman-Scott) processes, and it is turned off for log-Gaussian Cox processes (LGCP). If the model is stationary (and non-LGCP) this option is turned on by default and shows a fitted cluster positioned at the centroid of the observation window. For non-stationary (and non-LGCP) models this option is only invoked if explicitly told so, and in that case an additional argument locations (see clusterfield) must be given to specify where to position the parent point(s) .

Alternatively what="all" selects all available options.

Value

Null.

Author(s)

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

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

kppm, plot.ppm,

Examples

```
data(redwood)
fit <- kppm(redwood~1, "Thomas")
plot(fit)
```

---

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.leverage.ppm *Plot Leverage Function*

Description

Generate a pixel image plot, or a contour plot, or a perspective plot, of a leverage function that has been computed by leverage.ppm.

Usage

```
 ## S3 method for class 'leverage.ppm'
plot(x, ...,
                            what=c("smooth", "nearest", "exact"),
                            showcut=TRUE,
                            args.cut=list(drawlabels=FALSE),
                            multiplot=TRUE)

 ## S3 method for class 'leverage.ppm'
contour(x, ...,
                            what=c("smooth", "nearest"),
                            showcut=TRUE,
                            args.cut=list(col=3, lwd=3, drawlabels=FALSE),
                            multiplot=TRUE)

 ## S3 method for class 'leverage.ppm'
persp(x, ...,
              what=c("smooth", "nearest"),
              main, zlab="leverage")
```

Arguments

x	Leverage function (object of class "leverage.ppm") computed by leverage.ppm.
...	Arguments passed to plot.im or contour.im or persp.im controlling the plot.
what	Character string (partially matched) specifying the values to be plotted. See Details.
showcut	Logical. If TRUE, a contour line is plotted at the level equal to the theoretical mean of the leverage.
args.cut	Optional list of arguments passed to contour.default to control the plotting of the contour line for the mean leverage.
multiplot	Logical value indicating whether it is permissible to display several plot panels.
main	Optional main title. A character string or character vector.
zlab	Label for the z axis. A character string.

Details

These functions are the plot, contour and persp methods for objects of class "leverage.ppm". Such objects are computed by the command leverage.ppm.

The plot method displays the leverage function as a colour pixel image using plot.im, and draws a single contour line at the mean leverage value using contour.default. Use the argument clipwin to restrict the plot to a subset of the full data.

The contour method displays the leverage function as a contour plot, and also draws a single contour line at the mean leverage value, using contour.im.

The persp method displays the leverage function as a surface in perspective view, using persp.im.

Since the exact values of leverage are computed only at a finite set of quadrature locations, there are several options for these plots:

`what="smooth"`: (the default) an image plot showing a smooth function, obtained by applying kernel smoothing to the exact leverage values;

`what="nearest"`: an image plot showing a piecewise-constant function, obtained by taking the exact leverage value at the nearest quadrature point;

`what="exact"`: a symbol plot showing the exact values of leverage as circles, centred at the quadrature points, with diameters proportional to leverage.

The pixel images are already contained in the object `x` and were computed by `leverage.ppm`; the resolution of these images is controlled by arguments to `leverage.ppm`.

Value

Same as for `plot.im`, `contour.im` and `persp.im` respectively.

Author(s)

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

References

Baddeley, A., Chang, Y.M. and Song, Y. (2013) Leverage and influence diagnostics for spatial point process models. *Scandinavian Journal of Statistics* **40**, 86–104.

See Also

`leverage.ppm`.

Examples

```
if(offline <- !interactive()) op <- spatstat.options(npixel=32, ndummy.min=16)

   X <- rpoispp(function(x,y) { exp(3+3*x) })
   fit <- ppm(X ~x+y)
   lef <- leverage(fit)
   plot(lef)
   contour(lef)
   persp(lef)

if(offline) spatstat.options(op)
```

plot.mppm *plot a Fitted Multiple Point Process Model*

Description

Given a point process model fitted to multiple point patterns by mppm, compute spatial trend or conditional intensity surface of the model, in a form suitable for plotting, and (optionally) plot this surface.

Usage

```
  ## S3 method for class 'mppm'
plot(x, ...,
                trend=TRUE, cif=FALSE, se=FALSE,
                how=c("image", "contour", "persp"))
```

Arguments

x	A point process model fitted to multiple point patterns, typically obtained from the model-fitting algorithm mppm. An object of class "mppm".
...	Arguments passed to plot.ppm or plot.anylist controlling the plot.
trend	Logical value indicating whether to plot the fitted trend.
cif	Logical value indicating whether to plot the fitted conditional intensity.
se	Logical value indicating whether to plot the standard error of the fitted trend.
how	Single character string indicating the style of plot to be performed.

Details

This is the plot method for the class "mppm" of point process models fitted to multiple point patterns (see mppm).

It invokes subfits to compute the fitted model for each individual point pattern dataset, then calls plot.ppm to plot these individual models. These individual plots are displayed using plot.anylist, which generates either a series of separate plot frames or an array of plot panels on a single page.

Value

NULL.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Ida-Maria Sintorn and Leanne Bischoff. Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

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

See Also

plot.ppm, mppm, plot.listof

Examples

```
# Synthetic data from known model
n <- 9
H <- hyperframe(V=1:n,
                U=runif(n, min=-1, max=1))
H$Z <- setcov(square(1))
H$U <- with(H, as.im(U, as.rectangle(Z)))
H$Y <- with(H, rpoispp(eval.im(exp(2+3*Z))))

fit <- mppm(Y ~Z + U + V, data=H)

plot(fit)
```

---

plot.msr *Plot a Signed or Vector-Valued Measure*

Description

Plot a signed measure or vector-valued measure.

Usage

```
## S3 method for class 'msr'
plot(x, ...,
                      add = FALSE,
                      how = c("image", "contour", "imagecontour"),
                      main = NULL,
                      do.plot = TRUE,
                      multiplot = TRUE,
                      massthresh = 0,
                      equal.markscale = FALSE,
                      equal.ribbon = FALSE)
```

Arguments

x	The signed or vector measure to be plotted. An object of class "msr" (see msr).
...	Extra arguments passed to Smooth.ppp to control the interpolation of the continuous density component of x, or passed to plot.im or plot.ppp to control the appearance of the plot.

add	Logical flag; if TRUE, the graphics are added to the existing plot. If FALSE (the default) a new plot is initialised.
how	String indicating how to display the continuous density component.
main	String. Main title for the plot.
do.plot	Logical value determining whether to actually perform the plotting.
multiplot	Logical value indicating whether it is permissible to display a plot with multiple panels (representing different components of a vector-valued measure, or different types of points in a multitype measure.)
massthresh	Threshold for plotting atoms. A single numeric value or NULL. If massthresh=0 (the default) then only atoms with nonzero mass will be plotted. If massthresh > 0 then only atoms whose absolute mass exceeds massthresh will be plotted. If massthresh=NULL, then all atoms of the measure will be plotted.
equal.markscale	Logical value indicating whether different panels should use the same symbol map (to represent the masses of atoms of the measure).
equal.ribbon	Logical value indicating whether different panels should use the same colour map (to represent the density values in the diffuse component of the measure).

Details

This is the plot method for the class "msr".

The continuous density component of x is interpolated from the existing data by Smooth.ppp, and then displayed as a colour image by plot.im.

The discrete atomic component of x is then superimposed on this image by plotting the atoms as circles (for positive mass) or squares (for negative mass) by plot.ppp. By default, atoms with zero mass are not plotted at all.

To smooth both the discrete and continuous components, use Smooth.msr.

Use the argument clipwin to restrict the plot to a subset of the full data.

To remove atoms with tiny masses, use the argument massthresh.

Value

(Invisible) colour map (object of class "colourmap") for the colour image.

Author(s)

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

See Also

msr, Smooth.ppp, Smooth.msr, plot.im, plot.ppp

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X, ~x+y)
rp <- residuals(fit, type="pearson")
rs <- residuals(fit, type="score")

plot(rp)
plot(rs)
plot(rs, how="contour")
```

`plot.plotppm` *Plot a plotppm Object Created by plot.ppm*

Description

The function plot.ppm produces objects which specify plots of fitted point process models. The function plot.plotppm carries out the actual plotting of these objects.

Usage

```
  ## S3 method for class 'plotppm'
plot(x, data = NULL, trend = TRUE, cif = TRUE,
             se = TRUE, pause = interactive(),
             how = c("persp", "image", "contour"),
             ..., pppargs)
```

Arguments

`x`	An object of class `plotppm` produced by `plot.ppm()` .
`data`	The point pattern (an object of class ppp) to which the point process model was fitted (by `ppm`).
`trend`	Logical scalar; should the trend component of the fitted model be plotted?
`cif`	Logical scalar; should the complete conditional intensity of the fitted model be plotted?
`se`	Logical scalar; should the estimated standard error of the fitted intensity be plotted?
`pause`	Logical scalar indicating whether to pause with a prompt after each plot. Set `pause=FALSE` if plotting to a file.
`how`	Character string or character vector indicating the style or styles of plots to be performed.
`...`	Extra arguments to the plotting functions `persp`, `image` and `contour`.
`pppargs`	List of extra arguments passed to `plot.ppp` when displaying the original point pattern data.

Details

If argument data is supplied then the point pattern will be superimposed on the image and contour plots.

Sometimes a fitted model does not have a trend component, or the trend component may constitute all of the conditional intensity (if the model is Poisson). In such cases the object x will not contain a trend component, or will contain only a trend component. This will also be the case if one of the arguments trend and cif was set equal to FALSE in the call to plot.ppm() which produced x. If this is so then only the item which is present will be plotted. Explicitly setting trend=TRUE, or cif=TRUE, respectively, will then give an error.

Value

None.

Warning

Arguments which are passed to persp, image, and contour via the ... argument get passed to any of the other functions listed in the how argument, and won't be recognized by them. This leads to a lot of annoying but harmless warning messages. Arguments to persp may be supplied via spatstat.options() which alleviates the warning messages in this instance.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

plot.ppm()

Examples

```
if(interactive()) {
m <- ppm(cells ~ 1, Strauss(0.05))
mpic <- plot(m)
# Perspective plot only, with altered parameters:
 plot(mpic,how="persp", theta=-30,phi=40,d=4)
# All plots, with altered parameters for perspective plot:
op <- spatstat.options(par.persp=list(theta=-30,phi=40,d=4))
plot(mpic)
# Revert
spatstat.options(op)
}
```

plot.ppm *plot a Fitted Point Process Model*

Description

Given a fitted point process model obtained by ppm, create spatial trend and conditional intensity surfaces of the model, in a form suitable for plotting, and (optionally) plot these surfaces.

Usage

```
  ## S3 method for class 'ppm'
plot(x, ngrid = c(40,40), superimpose = TRUE,
                  trend = TRUE, cif = TRUE, se = TRUE, pause = interactive(),
                  how=c("persp","image", "contour"), plot.it = TRUE,
                  locations = NULL, covariates=NULL, ...)
```

Arguments

x	A fitted point process model, typically obtained from the model-fitting algorithm ppm. An object of class "ppm".
ngrid	The dimensions for a grid on which to evaluate, for plotting, the spatial trend and conditional intensity. A vector of 1 or 2 integers. If it is of length 1, ngrid is replaced by c(ngrid,ngrid).
superimpose	logical flag; if TRUE (and if plot=TRUE) the original data point pattern will be superimposed on the plots.
trend	logical flag; if TRUE, the spatial trend surface will be produced.
cif	logical flag; if TRUE, the conditional intensity surface will be produced.
se	logical flag; if TRUE, the estimated standard error of the spatial trend surface will be produced.
pause	logical flag indicating whether to pause with a prompt after each plot. Set pause=FALSE if plotting to a file. (This flag is ignored if plot=FALSE).
how	character string or character vector indicating the style or styles of plots to be performed. Ignored if plot=FALSE.
plot.it	logical scalar; should a plot be produced immediately?
locations	If present, this determines the locations of the pixels at which predictions are computed. It must be a binary pixel image (an object of class "owin" with type "mask"). (Incompatible with ngrid).
covariates	Values of external covariates required by the fitted model. Passed to predict.ppm.
...	extra arguments to the plotting functions persp, image and contour.

Details

This is the plot method for the class "ppm" (see ppm.object for details of this class).

It invokes predict.ppm to compute the spatial trend and conditional intensity of the fitted point process model. See predict.ppm for more explanation about spatial trend and conditional intensity.

The default action is to create a rectangular grid of points in (the bounding box of) the observation window of the data point pattern, and evaluate the spatial trend and conditional intensity of the fitted spatial point process model x at these locations. If the argument locations= is supplied, then the spatial trend and conditional intensity are calculated at the grid of points specified by this argument.

The argument locations, if present, should be a binary image mask (an object of class "owin" and type "mask"). This determines a rectangular grid of locations, or a subset of such a grid, at which predictions will be computed. Binary image masks are conveniently created using as.mask.

The argument covariates gives the values of any spatial covariates at the prediction locations. If the trend formula in the fitted model involves spatial covariates (other than the Cartesian coordinates x, y) then covariates is required.

The argument covariates has the same format and interpretation as in predict.ppm. It may be either a data frame (the number of whose rows must match the number of pixels in locations multiplied by the number of possible marks in the point pattern), or a list of images. If argument locations is not supplied, and covariates **is** supplied, then it **must** be a list of images.

If the fitted model was a marked (multitype) point process, then predictions are made for each possible mark value in turn.

If the fitted model had no spatial trend, then the default is to omit calculating this (flat) surface, unless trend=TRUE is set explicitly.

If the fitted model was Poisson, so that there were no spatial interactions, then the conditional intensity and spatial trend are identical, and the default is to omit the conditional intensity, unless cif=TRUE is set explicitly.

If plot.it=TRUE then plot.plotppm() is called upon to plot the class plotppm object which is produced. (That object is also returned, silently.)

Plots are produced successively using persp, image and contour (or only a selection of these three, if how is given). Extra graphical parameters controlling the display may be passed directly via the arguments ... or indirectly reset using spatstat.options.

Value

An object of class plotppm. Such objects may be plotted by plot.plotppm().

This is a list with components named trend and cif, either of which may be missing. They will be missing if the corresponding component does not make sense for the model, or if the corresponding argument was set equal to FALSE.

Both trend and cif are lists of images. If the model is an unmarked point process, then they are lists of length 1, so that trend[[1]] is an image of the spatial trend and cif[[1]] is an image of the conditional intensity.

If the model is a marked point process, then trend[[i]] is an image of the spatial trend for the mark m[i], and cif[[1]] is an image of the conditional intensity for the mark m[i], where m is the vector of levels of the marks.

Warnings

See warnings in predict.ppm.

Author(s)

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

See Also

plot.plotppm, ppm, ppm.object, predict.ppm, print.ppm, persp, image, contour, plot, spatstat.options

Examples

```
m <- ppm(cells ~1, Strauss(0.05))
pm <- plot(m) # The object ``pm'' will be plotted as well as saved
              # for future plotting.
pm
```

---

plot.profilepl *Plot Profile Likelihood*

Description

Plot the profile (pseudo) likelihood against the irregular parameters, for a model that was fitted by maximum profile (pseudo)likelihood.

Usage

```
## S3 method for class 'profilepl'
plot(x, ..., add = FALSE, main = NULL, tag = TRUE,
                          coeff = NULL, xvariable = NULL,
                          col = 1, lty = 1, lwd = 1,
                          col.opt = "green", lty.opt = 3, lwd.opt = 1)
```

Arguments

x	A point process model fitted by maximum profile (pseudo)likelihood. Object of class "profilepl", obtained from profilepl.
...	Additional plot arguments passed to plot.default and lines.
add	Logical. If TRUE, the plot is drawn over the existing plot.
main	Optional. Main title for the plot. A character string or character vector.
tag	Logical value. If TRUE (the default), when the plot contains multiple curves corresponding to different values of a parameter, each curve will be labelled with the values of the irregular parameter.

coeff	Optional. If this is given, it should be a character string matching the name of one of the fitted model coefficients. This coefficient will then be plotted on the vertical axis.
xvariable	Optional. The name of the irregular parameter that should be plotted along the horizontal axis. The default is the first irregular parameter.
col,lty,lwd	Graphical parameters (colour, line type, line width) for the curves on the plot.
col.opt, lty.opt, lwd.opt	Graphical parameters for indicating the optimal parameter value.

Details

This is the plot method for the class "profilepl" of fitted point process models obtained by maximising the profile likelihood or profile pseudolikelihood.

The default behaviour is to plot the profile likelihood or profile pseudolikelihood on the vertical axis, against the value of the irregular parameter on the horizontal axis.

If there are several irregular parameters, then one of them is plotted on the horizontal axis, and the plot consists of many different curves, corresponding to different values of the other parameters. The parameter to be plotted on the horizontal axis is specified by the argument xvariable; the default is to use the parameter that was listed first in the original call to profilepl.

If coeff is given, it should be the name of one of the fitted model coefficients names(coef(as.ppm(x))). The fitted value of that coefficient is plotted on the vertical axis.

Value

Null.

Author(s)

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

References

Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.

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

See Also

profilepl

Examples

```
live <- interactive()
nr <- if(live) 20 else 3

# one irregular parameter
```

```
rr <- data.frame(r=seq(0.05,0.15, length=nr))
ps <- profilepl(rr, Strauss, cells)
plot(ps)                      # profile pseudolikelihood
plot(ps, coeff="Interaction") # fitted interaction coefficient log(gamma)

# two irregular parameters
smax <- if(live) 3 else 2
rs <- expand.grid(r=seq(0.05,0.15, length=nr), sat=1:smax)
pg <- profilepl(rs, Geyer, cells)
plot(pg) # profile pseudolikelihood against r for each value of 'sat'
plot(pg, coeff="Interaction")
plot(pg, xvariable="sat", col=ifelse(r < 0.1, "red", "green"))
```

`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 <r.turner@auckland.ac.nz>

See Also

quadrat.test, plot.tess, text.default, plot.quadratcount

Examples

```
plot(quadrat.test(swedishpines, 3))
```

---

plot.rppm *Plot a Recursively Partitioned Point Process Model*

Description

Given a model which has been fitted to point pattern data by recursive partitioning, plot the partition tree or the fitted intensity.

Usage

```
## S3 method for class 'rppm'
plot(x, ..., what = c("tree", "spatial"), treeplot=NULL)
```

Arguments

x	Fitted point process model of class "rppm" produced by the function rppm.
what	Character string (partially matched) specifying whether to plot the partition tree or the fitted intensity.
...	Arguments passed to plot.rpart and text.rpart (if what="tree") or passed to plot.im (if what="spatial") controlling the appearance of the plot.
treeplot	Optional. A function to be used to plot and label the partition tree, replacing the two functions plot.rpart and text.rpart.

Details

If what="tree" (the default), the partition tree will be plotted using plot.rpart, and labelled using text.rpart.

If the argument treeplot is given, then plotting and labelling will be performed by treeplot instead. A good choice is the function prp in package **rpart.plot**.

If what="spatial", the predicted intensity will be computed using predict.rppm, and this intensity will be plotted as an image using plot.im.

Value

If what="tree", a list containing x and y coordinates of the plotted nodes of the tree. If what="spatial", the return value of plot.im.

Author(s)

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

See Also

rppm

Examples

```
# Murchison gold data
mur <- solapply(murchison, rescale, s=1000, unitname="km")
mur$dfault <- distfun(mur$faults)
#
fit <- rppm(gold ~ dfault + greenstone, data=mur)
#
opa <- par(mfrow=c(1,2))
plot(fit)
plot(fit, what="spatial")
par(opa)
```

---

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 <r.turner@auckland.ac.nz>

See Also

`scan.test`, `scanLRTS`

Examples

```
if(interactive()) {
  a <- scan.test(redwood, seq(0.04, 0.1, by=0.01),
                 method="poisson", nsim=19)
} else {
  a <- scan.test(redwood, c(0.05, 0.1), method="poisson", nsim=2)
}
plot(a)
as.im(a)
plot(a, what="radius")
```

`plot.slrm` *Plot a Fitted Spatial Logistic Regression*

Description

Plots a fitted Spatial Logistic Regression model.

Usage

```
  ## S3 method for class 'slrm'
plot(x, ..., type = "intensity")
```

Arguments

x	a fitted spatial logistic regression model. An object of class "slrm".
...	Extra arguments passed to plot.im to control the appearance of the plot.
type	Character string (partially) matching one of "probabilities", "intensity" or "link".

Details

This is a method for plot for fitted spatial logistic regression models (objects of class "slrm", usually obtained from the function slrm).

This function plots the result of predict.slrm.

Value

None.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> <adrian@maths.uwa.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

slrm, predict.slrm, plot.im

Examples

```
data(copper)
X <- copper$SouthPoints
Y <- copper$SouthLines
Z <- distmap(Y)
fit <- slrm(X ~ Z)
plot(fit)
plot(fit, type="link")
```

---

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 <r.turner@auckland.ac.nz> 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)
```

Poisson *Poisson Point Process Model*

Description

Creates an instance of the Poisson point process model which can then be fitted to point pattern data.

Usage

```
Poisson()
```

Details

The function ppm, which fits point process models to point pattern data, requires an argument `interaction` of class `"interact"` describing the interpoint interaction structure of the model to be fitted. The appropriate description of the Poisson process is provided by the value of the function `Poisson`.

This works for all types of Poisson processes including multitype and nonstationary Poisson processes.

Value

An object of class `"interact"` describing the interpoint interaction structure of the Poisson point process (namely, there are no interactions).

Author(s)

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

See Also

ppm, Strauss

Examples

```
ppm(nztrees ~1, Poisson())
# fit the stationary Poisson process to 'nztrees'
# no edge correction needed

lon <- longleaf

longadult <- unmark(subset(lon, marks >= 30))
ppm(longadult ~ x, Poisson())
# fit the nonstationary Poisson process
# with intensity lambda(x,y) = exp( a + bx)

# trees marked by species
lans <- lansing

ppm(lans ~ marks, Poisson())
# fit stationary marked Poisson process
# with different intensity for each species

# ppm(lansing ~ marks * polynom(x,y,3), Poisson())
# fit nonstationary marked Poisson process
# with different log-cubic trend for each species
```

`polynom` *Polynomial in One or Two Variables*

Description

This function is used to represent a polynomial term in a model formula. It computes the homogeneous terms in the polynomial of degree `n` in one variable `x` or two variables `x,y`.

Usage

```
polynom(x, ...)
```

Arguments

`x`	A numerical vector.
`...`	Either a single integer `n` specifying the degree of the polynomial, or two arguments `y,n` giving another vector of data `y` and the degree of the polynomial.

Details

This function is typically used inside a model formula in order to specify the most general possible polynomial of order `n` involving one numerical variable `x` or two numerical variables `x,y`.

It is equivalent to `poly(, raw=TRUE)`.

If only one numerical vector argument x is given, the function computes the vectors x^k for k = 1, 2, ..., n. These vectors are combined into a matrix with n columns.

If two numerical vector arguments x, y are given, the function computes the vectors x^k * y^m for k >= 0 and m >= 0 satisfying 0 < k + m <= n. These vectors are combined into a matrix with one column for each homogeneous term.

Value

A numeric matrix, with rows corresponding to the entries of x, and columns corresponding to the terms in the polynomial.

Author(s)

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

See Also

poly, harmonic

Examples

```
x <- 1:4
y <- 10 * (0:3)
polynom(x, 3)
polynom(x, y, 3)
```

---

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 <r.turner@auckland.ac.nz>

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 <r.turner@auckland.ac.nz>

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 <r.turner@auckland.ac.nz>

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 <r.turner@auckland.ac.nz>

See Also

fasp, alltypes, pool.envelope, pool

Examples

```
data(amacrine)
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 <r.turner@auckland.ac.nz> 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 <r.turner@auckland.ac.nz>

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_iw[i] is the ith 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 <r.turner@auckland.ac.nz> 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"))
```

---

ppm *Fit Point Process Model to Data*

Description

Fits a point process model to an observed point pattern.

Usage

```
ppm(Q, ...)

## S3 method for class 'formula'
ppm(Q, interaction=NULL, ..., data=NULL, subset)
```

Arguments

`Q`	A `formula` in the R language describing the model to be fitted.
`interaction`	An object of class `"interact"` describing the point process interaction structure, or a function that makes such an object, or `NULL` indicating that a Poisson process (stationary or nonstationary) should be fitted.
`...`	Arguments passed to `ppm.ppp` or `ppm.quad` to control the model-fitting process.
`data`	Optional. The values of spatial covariates (other than the Cartesian coordinates) required by the model. Either a data frame, or a list whose entries are images, functions, windows, tessellations or single numbers. See Details.
`subset`	Optional. An expression (which may involve the names of the Cartesian coordinates `x` and `y` and the names of entries in `data`) defining a subset of the spatial domain, to which the model-fitting should be restricted. The result of evaluating the expression should be either a logical vector, or a window (object of class `"owin"`) or a logical-valued pixel image (object of class `"im"`).

Details

This function fits a point process model to an observed point pattern. The model may include spatial trend, interpoint interaction, and dependence on covariates.

The model fitted by `ppm` is either a Poisson point process (in which different points do not interact with each other) or a Gibbs point process (in which different points typically inhibit each other). For clustered point process models, use `kppm`.

The function `ppm` is generic, with methods for the classes `formula`, `ppp` and `quad`. This page describes the method for a `formula`.

The first argument is a formula in the R language describing the spatial trend model to be fitted. It has the general form pattern ~ trend where the left hand side pattern is usually the name of a spatial point pattern (object of class "ppp") to which the model should be fitted, or an expression which evaluates to a point pattern; and the right hand side trend is an expression specifying the spatial trend of the model.

Systematic effects (spatial trend and/or dependence on spatial covariates) are specified by the trend expression on the right hand side of the formula. The trend may involve the Cartesian coordinates x, y, the marks marks, the names of entries in the argument data (if supplied), or the names of objects that exist in the R session. The trend formula specifies the **logarithm** of the intensity of a Poisson process, or in general, the logarithm of the first order potential of the Gibbs process. The formula should not use any names beginning with .mpl as these are reserved for internal use. If the formula is pattern~1, then the model to be fitted is stationary (or at least, its first order potential is constant).

The symbol . in the trend expression stands for all the covariates supplied in the argument data. For example the formula pattern ~ . indicates an additive model with a main effect for each covariate in data.

Stochastic interactions between random points of the point process are defined by the argument interaction. This is an object of class "interact" which is initialised in a very similar way to the usage of family objects in glm and gam. The interaction models currently available are: AreaInter, BadGey, Concom, DiggleGatesStibbard, DiggleGratton, Fiksel, Geyer, Hardcore, HierHard, HierStrauss, HierStraussHard, Hybrid, LennardJones, MultiHard, MultiStrauss, MultiStraussHard, OrdThresh, Ord, Pairwise, PairPiece, Penttinen, Poisson, Saturated, SatPiece, Softcore, Strauss, StraussHard and Triplets. See the examples below. Note that it is possible to combine several interactions using Hybrid.

If interaction is missing or NULL, then the model to be fitted has no interpoint interactions, that is, it is a Poisson process (stationary or nonstationary according to trend). In this case the methods of maximum pseudolikelihood and maximum logistic likelihood coincide with maximum likelihood.

The fitted point process model returned by this function can be printed (by the print method print.ppm) to inspect the fitted parameter values. If a nonparametric spatial trend was fitted, this can be extracted using the predict method predict.ppm.

To fit a model involving spatial covariates other than the Cartesian coordinates x and y, the values of the covariates should either be supplied in the argument data, or should be stored in objects that exist in the R session. Note that it is not sufficient to have observed the covariate only at the points of the data point pattern; the covariate must also have been observed at other locations in the window.

If it is given, the argument data is typically a list, with names corresponding to variables in the trend formula. Each entry in the list is either

a pixel image, giving the values of a spatial covariate at a fine grid of locations. It should be an object of class "im", see im.object.

a function, which can be evaluated at any location (x,y) to obtain the value of the spatial covariate. It should be a function(x, y) or function(x, y, ...) in the R language. For marked point pattern data, the covariate can be a function(x, y, marks) or function(x, y, marks, ...). The first two arguments of the function should be the Cartesian coordinates x and y. The function may have additional arguments; if the function does not have default values for these additional arguments, then the user must supply values for them, in covfunargs. See the Examples.

a window, interpreted as a logical variable which is TRUE inside the window and FALSE outside it. This should be an object of class "owin".

a tessellation, interpreted as a factor covariate. For each spatial location, the factor value indicates which tile of the tessellation it belongs to. This should be an object of class "tess". (To make a covariate in which each tile of the tessellation has a numerical value, convert the tessellation to a function(x,y) using as.function.tess.)

a single number, indicating a covariate that is constant in this dataset.

The software will look up the values of each covariate at the required locations (quadrature points).

Note that, for covariate functions, only the *name* of the function appears in the trend formula. A covariate function is treated as if it were a single variable. The function arguments do not appear in the trend formula. See the Examples.

If data is a list, the list entries should have names corresponding to (some of) the names of covariates in the model formula trend. The variable names x, y and marks are reserved for the Cartesian coordinates and the mark values, and these should not be used for variables in data.

Alternatively, data may be a data frame giving the values of the covariates at specified locations. Then pattern should be a quadrature scheme (object of class "quad") giving the corresponding locations. See ppm.quad for details.

Value

An object of class "ppm" describing a fitted point process model.

See ppm.object for details of the format of this object and methods available for manipulating it.

Interaction parameters

Apart from the Poisson model, every point process model fitted by ppm has parameters that determine the strength and range of 'interaction' or dependence between points. These parameters are of two types:

regular parameters: A parameter ϕ is called *regular* if the log likelihood is a linear function of θ where $\theta = \theta(\psi)$ is some transformation of ψ. [Then θ is called the canonical parameter.]

irregular parameters Other parameters are called *irregular*.

Typically, regular parameters determine the 'strength' of the interaction, while irregular parameters determine the 'range' of the interaction. For example, the Strauss process has a regular parameter γ controlling the strength of interpoint inhibition, and an irregular parameter r determining the range of interaction.

The ppm command is only designed to estimate regular parameters of the interaction. It requires the values of any irregular parameters of the interaction to be fixed. For example, to fit a Strauss process model to the cells dataset, you could type ppm(cells ~ 1, Strauss(r=0.07)). Note that the value of the irregular parameter r must be given. The result of this command will be a fitted model in which the regular parameter γ has been estimated.

To determine the irregular parameters, there are several practical techniques, but no general statistical theory available. Useful techniques include maximum profile pseudolikelihood, which is implemented in the command profilepl, and Newton-Raphson maximisation, implemented in the experimental command ippm.

Some irregular parameters can be estimated directly from data: the hard-core radius in the model Hardcore and the matrix of hard-core radii in MultiHard can be estimated easily from data. In these cases, ppm allows the user to specify the interaction without giving the value of the irregular parameter. The user can give the hard core interaction as interaction=Hardcore() or even interaction=Hardcore, and the hard core radius will then be estimated from the data.

Technical Warnings and Error Messages

See ppm.ppp for some technical warnings about the weaknesses of the algorithm, and explanation of some common error messages.

Author(s)

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

References

Baddeley, A., Coeurjolly, J.-F., Rubak, E. and Waagepetersen, R. (2014) Logistic regression for spatial Gibbs point processes. *Biometrika* **101** (2) 377–392.

Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42** 283–322.

Berman, M. and Turner, T.R. (1992) Approximating point process likelihoods with GLIM. *Applied Statistics* **41**, 31–38.

Besag, J. (1975) Statistical analysis of non-lattice data. *The Statistician* **24**, 179-195.

Diggle, P.J., Fiksel, T., Grabarnik, P., Ogata, Y., Stoyan, D. and Tanemura, M. (1994) On parameter estimation for pairwise interaction processes. *International Statistical Review* **62**, 99-117.

Huang, F. and Ogata, Y. (1999) Improvements of the maximum pseudo-likelihood estimators in various spatial statistical models. *Journal of Computational and Graphical Statistics* **8**, 510–530.

Jensen, J.L. and Moeller, M. (1991) Pseudolikelihood for exponential family models of spatial point processes. *Annals of Applied Probability* **1**, 445–461.

Jensen, J.L. and Kuensch, H.R. (1994) On asymptotic normality of pseudo likelihood estimates for pairwise interaction processes, *Annals of the Institute of Statistical Mathematics* **46**, 475–486.

See Also

ppm.ppp and ppm.quad for more details on the fitting technique and edge correction.

ppm.object for details of how to print, plot and manipulate a fitted model.

ppp and quadscheme for constructing data.

Interactions: AreaInter, BadGey, Concom, DiggleGatesStibbard, DiggleGratton, Fiksel, Geyer, Hardcore, HierHard, HierStrauss, HierStraussHard, Hybrid, LennardJones, MultiHard, MultiStrauss, MultiStraussHard, OrdThresh, Ord, Pairwise, PairPiece, Penttinen, Poisson, Saturated, SatPiece, Softcore, Strauss, StraussHard and Triplets.

See profilepl for advice on fitting nuisance parameters in the interaction, and ippm for irregular parameters in the trend.

See valid.ppm and project.ppm for ensuring the fitted model is a valid point process.

See kppm for fitting Cox point process models and cluster point process models, and dppm for fitting determinantal point process models.

Examples

```
online <- interactive()
if(!online) {
   # reduce grid sizes for efficiency in tests
   spatstat.options(npixel=32, ndummy.min=16)
}

# fit the stationary Poisson process
# to point pattern 'nztrees'

ppm(nztrees ~ 1)

if(online) {
  Q <- quadscheme(nztrees)
  ppm(Q ~ 1)
  # equivalent.
}

fit1 <- ppm(nztrees ~ x)
# fit the nonstationary Poisson process
# with intensity function lambda(x,y) = exp(a + bx)
# where x,y are the Cartesian coordinates
# and a,b are parameters to be estimated

fit1
coef(fit1)
coef(summary(fit1))

ppm(nztrees ~ polynom(x,2))

# fit the nonstationary Poisson process
# with intensity function lambda(x,y) = exp(a + bx + cx^2)

if(online) {
  library(splines)
  ppm(nztrees ~ bs(x,df=3))
}
# Fits the nonstationary Poisson process
# with intensity function lambda(x,y) = exp(B(x))
# where B is a B-spline with df = 3

ppm(nztrees ~ 1, Strauss(r=10), rbord=10)

# Fit the stationary Strauss process with interaction range r=10
# using the border method with margin rbord=10

ppm(nztrees ~ x, Strauss(13), correction="periodic")

# Fit the nonstationary Strauss process with interaction range r=13
```

```
# and exp(first order potential) =  activity = beta(x,y) = exp(a+bx)
# using the periodic correction.

 # Compare Maximum Pseudolikelihood, Huang-Ogata and Variational Bayes fits:
 if(online) ppm(swedishpines ~ 1, Strauss(9))

 ppm(swedishpines ~ 1, Strauss(9), method="ho",
     nsim=if(!online) 8 else 99)

 ppm(swedishpines ~ 1, Strauss(9), method="VBlogi")

# COVARIATES
#
X <- rpoispp(20)
weirdfunction <- function(x,y){ 10 * x^2 + 5 * sin(10 * y) }
#
# (a) covariate values as function
ppm(X ~ y + weirdfunction)
#
# (b) covariate values in pixel image
Zimage <- as.im(weirdfunction, unit.square())
ppm(X ~ y + Z, covariates=list(Z=Zimage))
#
# (c) covariate values in data frame
Q <- quadscheme(X)
xQ <- x.quad(Q)
yQ <- y.quad(Q)
Zvalues <- weirdfunction(xQ,yQ)
ppm(Q ~  y + Z, data=data.frame(Z=Zvalues))
# Note Q not X

# COVARIATE FUNCTION WITH EXTRA ARGUMENTS
#
f <- function(x,y,a){ y - a }
ppm(X ~ x + f, covfunargs=list(a=1/2))

# COVARIATE: logical value TRUE inside window, FALSE outside
b <- owin(c(0.1, 0.6), c(0.1, 0.9))
ppm(X ~ b)

## MULTITYPE POINT PROCESSES ###
# fit stationary marked Poisson process
# with different intensity for each species
if(online) {
  ppm(lansing ~  marks, Poisson())
} else {
 ama <- amacrine[square(0.7)]
 a <- ppm(ama ~  marks, Poisson(), nd=16)
}

# fit nonstationary marked Poisson process
# with different log-cubic trend for each species
if(online) {
```

```
 ppm(lansing ~  marks * polynom(x,y,3), Poisson())
} else {
 b <- ppm(ama ~  marks * polynom(x,y,2), Poisson(), nd=16)
}
```

---

ppm.object *Class of Fitted Point Process Models*

Description

A class ppm to represent a fitted stochastic model for a point process. The output of ppm.

Details

An object of class ppm represents a stochastic point process model that has been fitted to a point pattern dataset. Typically it is the output of the model fitter, ppm.

The class ppm has methods for the following standard generic functions:

generic	method	description
print	print.ppm	print details
plot	plot.ppm	plot fitted model
predict	predict.ppm	fitted intensity and conditional intensity
fitted	fitted.ppm	fitted intensity
coef	coef.ppm	fitted coefficients of model
anova	anova.ppm	Analysis of Deviance
formula	formula.ppm	Extract model formula
terms	terms.ppm	Terms in the model formula
labels	labels.ppm	Names of estimable terms in the model formula
residuals	residuals.ppm	Point process residuals
simulate	simulate.ppm	Simulate the fitted model
update	update.ppm	Change or refit the model
vcov	vcov.ppm	Variance/covariance matrix of parameter estimates
model.frame	model.frame.ppm	Model frame
model.matrix	model.matrix.ppm	Design matrix
logLik	logLik.ppm	log *pseudo* likelihood
extractAIC	extractAIC.ppm	pseudolikelihood counterpart of AIC
nobs	nobs.ppm	number of observations

Objects of class ppm can also be handled by the following standard functions, without requiring a special method:

name	description
confint	Confidence intervals for parameters
step	Stepwise model selection
drop1	One-step model improvement
add1	One-step model improvement

The class ppm also has methods for the following generic functions defined in the **spatstat** package:

generic	method	description
as.interact	as.interact.ppm	Interpoint interaction structure
as.owin	as.owin.ppm	Observation window of data
berman.test	berman.test.ppm	Berman's test
envelope	envelope.ppm	Simulation envelopes
fitin	fitin.ppm	Fitted interaction
is.marked	is.marked.ppm	Determine whether the model is marked
is.multitype	is.multitype.ppm	Determine whether the model is multitype
is.poisson	is.poisson.ppm	Determine whether the model is Poisson
is.stationary	is.stationary.ppm	Determine whether the model is stationary
cdf.test	cdf.test.ppm	Spatial distribution test
quadrat.test	quadrat.test.ppm	Quadrat counting test
reach	reach.ppm	Interaction range of model
rmhmodel	rmhmodel.ppm	Model in a form that can be simulated
rmh	rmh.ppm	Perform simulation
unitname	unitname.ppm	Name of unit of length

Information about the data (to which the model was fitted) can be extracted using data.ppm, dummy.ppm and quad.ppm.

Internal format

If you really need to get at the internals, a ppm object contains at least the following entries:

coef	the fitted regular parameters (as returned by glm)
trend	the trend formula or NULL
interaction	the point process interaction family (an object of class "interact") or NULL
Q	the quadrature scheme used
maxlogpl	the maximised value of log pseudolikelihood
correction	name of edge correction method used

See ppm for explanation of these concepts. The irregular parameters (e.g. the interaction radius of the Strauss process) are encoded in the interaction entry. However see the Warnings.

Warnings

The internal representation of ppm objects may change slightly between releases of the **spatstat** package.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppm, coef.ppm, fitted.ppm, print.ppm, predict.ppm, plot.ppm.

Examples

```
fit <- ppm(cells ~ x, Strauss(0.1), correction="periodic")
fit
coef(fit)
# pred <- predict(fit)
pred <- predict(fit, ngrid=20, type="trend")
if(interactive()) {
plot(fit)
}
```

---

ppm.ppp *Fit Point Process Model to Point Pattern Data*

Description

Fits a point process model to an observed point pattern.

Usage

```
   ## S3 method for class 'ppp'
ppm(Q, trend=~1, interaction=Poisson(),
       ...,
       covariates=data,
       data=NULL,
       covfunargs = list(),
       subset,
       clipwin,
       correction="border",
       rbord=reach(interaction),
       use.gam=FALSE,
       method="mpl",
       forcefit=FALSE,
       emend=project,
       project=FALSE,
       prior.mean = NULL,
       prior.var = NULL,
       nd = NULL,
       eps = NULL,
       gcontrol=list(),
       nsim=100, nrmh=1e5, start=NULL, control=list(nrep=nrmh),
       verb=TRUE,
       callstring=NULL)
```

```
   ## S3 method for class 'quad'
ppm(Q, trend=~1, interaction=Poisson(),
       ...,
       covariates=data,
       data=NULL,
       covfunargs = list(),
       subset,
       clipwin,
       correction="border",
       rbord=reach(interaction),
       use.gam=FALSE,
       method="mpl",
       forcefit=FALSE,
       emend=project,
       project=FALSE,
       prior.mean = NULL,
       prior.var = NULL,
       nd = NULL,
       eps = NULL,
       gcontrol=list(),
       nsim=100, nrmh=1e5, start=NULL, control=list(nrep=nrmh),
       verb=TRUE,
       callstring=NULL)
```

Arguments

Q	A data point pattern (of class "ppp") to which the model will be fitted, or a quadrature scheme (of class "quad") containing this pattern.
trend	An R formula object specifying the spatial trend to be fitted. The default formula, ~1, indicates the model is stationary and no trend is to be fitted.
interaction	An object of class "interact" describing the point process interaction structure, or a function that makes such an object, or NULL indicating that a Poisson process (stationary or nonstationary) should be fitted.
...	Ignored.
data,covariates	The values of any spatial covariates (other than the Cartesian coordinates) required by the model. Either a data frame, or a list whose entries are images, functions, windows, tessellations or single numbers. See Details.
subset	Optional. An expression (which may involve the names of the Cartesian coordinates x and y and the names of entries in data) defining a subset of the spatial domain, to which the likelihood or pseudolikelihood should be restricted. See Details. The result of evaluating the expression should be either a logical vector, or a window (object of class "owin") or a logical-valued pixel image (object of class "im").
clipwin	Optional. A spatial window (object of class "owin") to which data will be restricted, before model-fitting is performed. See Details.

covfunargs	A named list containing the values of any additional arguments required by covariate functions.
correction	The name of the edge correction to be used. The default is "border" indicating the border correction. Other possibilities may include "Ripley", "isotropic", "periodic", "translate" and "none", depending on the interaction.
rbord	If correction = "border" this argument specifies the distance by which the window should be eroded for the border correction.
use.gam	Logical flag; if TRUE then computations are performed using gam instead of glm.
method	The method used to fit the model. Options are "mpl" for the method of Maximum PseudoLikelihood, "logi" for the Logistic Likelihood method, "VBlogi" for the Variational Bayes Logistic Likelihood method, and "ho" for the Huang-Ogata approximate maximum likelihood method.
forcefit	Logical flag for internal use. If forcefit=FALSE, some trivial models will be fitted by a shortcut. If forcefit=TRUE, the generic fitting method will always be used.
emend,project	(These are equivalent: project is an older name for emend.) Logical value. Setting emend=TRUE will ensure that the fitted model is always a valid point process by applying emend.ppm.
prior.mean	Optional vector of prior means for canonical parameters (for method="VBlogi"). See Details.
prior.var	Optional prior variance covariance matrix for canonical parameters (for method="VBlogi"). See Details.
nd	Optional. Integer or pair of integers. The dimension of the grid of dummy points (nd * nd or nd[1] * nd[2]) used to evaluate the integral in the pseudolikelihood. Incompatible with eps.
eps	Optional. A positive number, or a vector of two positive numbers, giving the horizontal and vertical spacing, respectively, of the grid of dummy points. Incompatible with nd.
gcontrol	Optional. List of parameters passed to glm.control (or passed to gam.control if use.gam=TRUE) controlling the model-fitting algorithm.
nsim	Number of simulated realisations to generate (for method="ho")
nrmh	Number of Metropolis-Hastings iterations for each simulated realisation (for method="ho")
start,control	Arguments passed to rmh controlling the behaviour of the Metropolis-Hastings algorithm (for method="ho")
verb	Logical flag indicating whether to print progress reports (for method="ho")
callstring	Internal use only.

Details

NOTE: This help page describes the **old syntax** of the function ppm, described in many older documents. This old syntax is still supported. However, if you are learning about ppm for the first time, we recommend you use the **new syntax** described in the help file for ppm.

This function fits a point process model to an observed point pattern. The model may include spatial trend, interpoint interaction, and dependence on covariates.

basic use: In basic use, Q is a point pattern dataset (an object of class "ppp") to which we wish to fit a model.

The syntax of ppm() is closely analogous to the R functions glm and gam. The analogy is:

glm	ppm
formula	trend
family	interaction

The point process model to be fitted is specified by the arguments trend and interaction which are respectively analogous to the formula and family arguments of glm().

Systematic effects (spatial trend and/or dependence on spatial covariates) are specified by the argument trend. This is an R formula object, which may be expressed in terms of the Cartesian coordinates x, y, the marks marks, or the variables in covariates (if supplied), or both. It specifies the **logarithm** of the first order potential of the process. The formula should not use any names beginning with .mpl as these are reserved for internal use. If trend is absent or equal to the default, ~1, then the model to be fitted is stationary (or at least, its first order potential is constant).

The symbol . in the trend expression stands for all the covariates supplied in the argument data. For example the formula ~ . indicates an additive model with a main effect for each covariate in data.

Stochastic interactions between random points of the point process are defined by the argument interaction. This is an object of class "interact" which is initialised in a very similar way to the usage of family objects in glm and gam. The models currently available are: AreaInter, BadGey, Concom, DiggleGatesStibbard, DiggleGratton, Fiksel, Geyer, Hardcore, HierHard, HierStrauss, HierStraussHard, Hybrid, LennardJones, MultiHard, MultiStrauss, MultiStraussHard, OrdThresh, Ord, Pairwise, PairPiece, Penttinen, Poisson, Saturated, SatPiece, Softcore, Strauss, StraussHard and Triplets. See the examples below. It is also possible to combine several interactions using Hybrid.

If interaction is missing or NULL, then the model to be fitted has no interpoint interactions, that is, it is a Poisson process (stationary or nonstationary according to trend). In this case the methods of maximum pseudolikelihood and maximum logistic likelihood coincide with maximum likelihood.

The fitted point process model returned by this function can be printed (by the print method print.ppm) to inspect the fitted parameter values. If a nonparametric spatial trend was fitted, this can be extracted using the predict method predict.ppm.

Models with covariates: To fit a model involving spatial covariates other than the Cartesian coordinates x and y, the values of the covariates should be supplied in the argument covariates. Note that it is not sufficient to have observed the covariate only at the points of the data point pattern; the covariate must also have been observed at other locations in the window.

Typically the argument covariates is a list, with names corresponding to variables in the trend formula. Each entry in the list is either

a pixel image, giving the values of a spatial covariate at a fine grid of locations. It should be an object of class "im", see im.object.

a function, which can be evaluated at any location (x,y) to obtain the value of the spatial covariate. It should be a function(x, y) or function(x, y, ...) in the R language. For marked point pattern data, the covariate can be a function(x, y, marks) or function(x, y, marks, ...). The first two arguments of the function should be the

Cartesian coordinates x and y. The function may have additional arguments; if the function does not have default values for these additional arguments, then the user must supply values for them, in `covfunargs`. See the Examples.

a window, interpreted as a logical variable which is `TRUE` inside the window and `FALSE` outside it. This should be an object of class `"owin"`.

a tessellation, interpreted as a factor covariate. For each spatial location, the factor value indicates which tile of the tessellation it belongs to. This should be an object of class `"tess"`.

a single number, indicating a covariate that is constant in this dataset.

The software will look up the values of each covariate at the required locations (quadrature points).

Note that, for covariate functions, only the *name* of the function appears in the trend formula. A covariate function is treated as if it were a single variable. The function arguments do not appear in the trend formula. See the Examples.

If `covariates` is a list, the list entries should have names corresponding to the names of covariates in the model formula `trend`. The variable names `x`, `y` and `marks` are reserved for the Cartesian coordinates and the mark values, and these should not be used for variables in `covariates`.

If `covariates` is a data frame, `Q` must be a quadrature scheme (see under Quadrature Schemes below). Then `covariates` must have as many rows as there are points in `Q`. The ith row of `covariates` should contain the values of spatial variables which have been observed at the ith point of `Q`.

Quadrature schemes: In advanced use, `Q` may be a 'quadrature scheme'. This was originally just a technicality but it has turned out to have practical uses, as we explain below.

Quadrature schemes are required for our implementation of the method of maximum pseudolikelihood. The definition of the pseudolikelihood involves an integral over the spatial window containing the data. In practice this integral must be approximated by a finite sum over a set of quadrature points. We use the technique of Baddeley and Turner (2000), a generalisation of the Berman-Turner (1992) device. In this technique the quadrature points for the numerical approximation include all the data points (points of the observed point pattern) as well as additional 'dummy' points.

Quadrature schemes are also required for the method of maximum logistic likelihood, which combines the data points with additional 'dummy' points.

A quadrature scheme is an object of class `"quad"` (see `quad.object`) which specifies both the data point pattern and the dummy points for the quadrature scheme, as well as the quadrature weights associated with these points. If `Q` is simply a point pattern (of class `"ppp"`, see `ppp.object`) then it is interpreted as specifying the data points only; a set of dummy points specified by `default.dummy()` is added, and the default weighting rule is invoked to compute the quadrature weights.

Finer quadrature schemes (i.e. those with more dummy points) generally yield a better approximation, at the expense of higher computational load.

An easy way to fit models using a finer quadrature scheme is to let `Q` be the original point pattern data, and use the argument `nd` to determine the number of dummy points in the quadrature scheme.

Complete control over the quadrature scheme is possible. See `quadscheme` for an overview. Use `quadscheme(X, D, method="dirichlet")` to compute quadrature weights based on the

Dirichlet tessellation, or `quadscheme(X, D, method="grid")` to compute quadrature weights by counting points in grid squares, where `X` and `D` are the patterns of data points and dummy points respectively. Alternatively use `pixelquad` to make a quadrature scheme with a dummy point at every pixel in a pixel image.

A practical advantage of quadrature schemes arises when we want to fit a model involving covariates (e.g. soil pH). Suppose we have only been able to observe the covariates at a small number of locations. Suppose `cov.dat` is a data frame containing the values of the covariates at the data points (i.e.\ `cov.dat[i,]` contains the observations for the `i`th data point) and `cov.dum` is another data frame (with the same columns as `cov.dat`) containing the covariate values at another set of points whose locations are given by the point pattern `Y`. Then setting `Q = quadscheme(X,Y)` combines the data points and dummy points into a quadrature scheme, and `covariates = rbind(cov.dat, cov.dum)` combines the covariate data frames. We can then fit the model by calling `ppm(Q, ..., covariates)`.

Model-fitting technique: There are several choices for the technique used to fit the model.

method="mpl" (the default): the model will be fitted by maximising the pseudolikelihood (Besag, 1975) using the Berman-Turner computational approximation (Berman and Turner, 1992; Baddeley and Turner, 2000). Maximum pseudolikelihood is equivalent to maximum likelihood if the model is a Poisson process. Maximum pseudolikelihood is biased if the interpoint interaction is very strong, unless there is a large number of dummy points. The default settings for `method='mpl'` specify a moderately large number of dummy points, striking a compromise between speed and accuracy.

method="logi": the model will be fitted by maximising the logistic likelihood (Baddeley et al, 2014). This technique is roughly equivalent in speed to maximum pseudolikelihood, but is believed to be less biased. Because it is less biased, the default settings for `method='logi'` specify a relatively small number of dummy points, so that this method is the fastest, in practice.

method="VBlogi": the model will be fitted in a Bayesian setup by maximising the posterior probability density for the canonical model parameters. This uses the variational Bayes approximation to the posterior derived from the logistic likelihood as described in Rajala (2014). The prior is assumed to be multivariate Gaussian with mean vector `prior.mean` and variance-covariance matrix `prior.var`.

method="ho": the model will be fitted by applying the approximate maximum likelihood method of Huang and Ogata (1999). See below. The Huang-Ogata method is slower than the other options, but has better statistical properties.

Note that `method='logi'`, `method='VBlogi'` and `method='ho'` involve randomisation, so that the results are subject to random variation.

Huang-Ogata method: If `method="ho"` then the model will be fitted using the Huang-Ogata (1999) approximate maximum likelihood method. First the model is fitted by maximum pseudolikelihood as described above, yielding an initial estimate of the parameter vector θ_0. From this initial model, `nsim` simulated realisations are generated. The score and Fisher information of the model at $\theta = \theta_0$ are estimated from the simulated realisations. Then one step of the Fisher scoring algorithm is taken, yielding an updated estimate θ_1. The corresponding model is returned.

Simulated realisations are generated using `rmh`. The iterative behaviour of the Metropolis-Hastings algorithm is controlled by the arguments `start` and `control` which are passed to `rmh`.

As a shortcut, the argument nrmh determines the number of Metropolis-Hastings iterations run to produce one simulated realisation (if control is absent). Also if start is absent or equal to NULL, it defaults to list(n.start=N) where N is the number of points in the data point pattern.

Edge correction Edge correction should be applied to the sufficient statistics of the model, to reduce bias. The argument correction is the name of an edge correction method. The default correction="border" specifies the border correction, in which the quadrature window (the domain of integration of the pseudolikelihood) is obtained by trimming off a margin of width rbord from the observation window of the data pattern. Not all edge corrections are implemented (or implementable) for arbitrary windows. Other options depend on the argument interaction, but these generally include correction="periodic" (the periodic or toroidal edge correction in which opposite edges of a rectangular window are identified) and correction="translate" (the translation correction, see Baddeley 1998 and Baddeley and Turner 2000). For pairwise interaction models there is also Ripley's isotropic correction, identified by correction="isotropic" or "Ripley".

Subsetting The arguments subset and clipwin specify that the model should be fitted to a restricted subset of the available data. These arguments are equivalent for Poisson point process models, but different for Gibbs models. If clipwin is specified, then all the available data will be restricted to this spatial region, and data outside this region will be discarded, before the model is fitted. If subset is specified, then no data are deleted, but the domain of integration of the likelihood or pseudolikelihood is restricted to the subset. For Poisson models, these two arguments have the same effect; but for a Gibbs model, interactions between points inside and outside the subset are taken into account, while interactions between points inside and outside the clipwin are ignored.

Value

An object of class "ppm" describing a fitted point process model.

See ppm.object for details of the format of this object and methods available for manipulating it.

Interaction parameters

Apart from the Poisson model, every point process model fitted by ppm has parameters that determine the strength and range of 'interaction' or dependence between points. These parameters are of two types:

regular parameters: A parameter ϕ is called *regular* if the log likelihood is a linear function of θ where $\theta = \theta(\psi)$ is some transformation of ψ. [Then θ is called the canonical parameter.]

irregular parameters Other parameters are called *irregular*.

Typically, regular parameters determine the 'strength' of the interaction, while irregular parameters determine the 'range' of the interaction. For example, the Strauss process has a regular parameter γ controlling the strength of interpoint inhibition, and an irregular parameter r determining the range of interaction.

The ppm command is only designed to estimate regular parameters of the interaction. It requires the values of any irregular parameters of the interaction to be fixed. For example, to fit a Strauss process model to the cells dataset, you could type ppm(cells, ~1, Strauss(r=0.07)). Note

that the value of the irregular parameter r must be given. The result of this command will be a fitted model in which the regular parameter γ has been estimated.

To determine the irregular parameters, there are several practical techniques, but no general statistical theory available. Useful techniques include maximum profile pseudolikelihood, which is implemented in the command profilepl, and Newton-Raphson maximisation, implemented in the experimental command ippm.

Some irregular parameters can be estimated directly from data: the hard-core radius in the model Hardcore and the matrix of hard-core radii in MultiHard can be estimated easily from data. In these cases, ppm allows the user to specify the interaction without giving the value of the irregular parameter. The user can give the hard core interaction as interaction=Hardcore() or even interaction=Hardcore, and the hard core radius will then be estimated from the data.

Error and Warning Messages

Some common error messages and warning messages are listed below, with explanations.

"System is computationally singular" The Fisher information matrix of the fitted model has a determinant close to zero, so that the matrix cannot be inverted, and the software cannot calculate standard errors or confidence intervals. This error is usually reported when the model is printed, because the print method calculates standard errors for the fitted parameters. Singularity usually occurs because the spatial coordinates in the original data were very large numbers (e.g. expressed in metres) so that the fitted coefficients were very small numbers. The simple remedy is to **rescale the data**, for example, to convert from metres to kilometres by X <- rescale(X, 1000), then re-fit the model. Singularity can also occur if the covariate values are very large numbers, or if the covariates are approximately collinear.

"Covariate values were NA or undefined at X% (M out of N) of the quadrature points" The covariate data (typically a pixel image) did not provide values of the covariate at some of the spatial locations in the observation window of the point pattern. This means that the spatial domain of the pixel image does not completely cover the observation window of the point pattern. If the percentage is small, this warning can be ignored - typically it happens because of rounding effects which cause the pixel image to be one-pixel-width narrower than the observation window. However if more than a few percent of covariate values are undefined, it would be prudent to check that the pixel images are correct, and are correctly registered in their spatial relation to the observation window.

"Some tiles with positive area do not contain any quadrature points: relative error = X%" A problem has arisen when creating the quadrature scheme used to fit the model. In the default rule for computing the quadrature weights, space is divided into rectangular tiles, and the number of quadrature points (data and dummy points) in each tile is counted. It is possible for a tile with non-zero area to contain no quadrature points; in this case, the quadrature scheme will contribute a bias to the model-fitting procedure. **A small relative error (less than 2 percent) is not important.** Relative errors of a few percent can occur because of the shape of the window. If the relative error is greater than about 5 percent, we recommend trying different parameters for the quadrature scheme, perhaps setting a larger value of nd to increase the number of dummy points. A relative error greater than 10 percent indicates a major problem with the input data: in this case, extract the quadrature scheme by applying quad.ppm to the fitted model, and inspect it. (The most likely cause of this problem is that the spatial coordinates of the original data were not handled correctly, for example, coordinates of the locations and the window boundary were incompatible.)

"Model is unidentifiable" It is not possible to estimate all the model parameters from this dataset. The error message gives a further explanation, such as "data pattern is empty". Choose a simpler model, or check the data.

"N data points are illegal (zero conditional intensity)" In a Gibbs model (i.e. with interaction between points), the conditional intensity may be zero at some spatial locations, indicating that the model forbids the presence of a point at these locations. However if the conditional intensity is zero *at a data point*, this means that the model is inconsistent with the data. Modify the interaction parameters so that the data point is not illegal (e.g. reduce the value of the hard core radius) or choose a different interaction.

Warnings

The implementation of the Huang-Ogata method is experimental; several bugs were fixed in **spatstat** 1.19-0.

See the comments above about the possible inefficiency and bias of the maximum pseudolikelihood estimator.

The accuracy of the Berman-Turner approximation to the pseudolikelihood depends on the number of dummy points used in the quadrature scheme. The number of dummy points should at least equal the number of data points.

The parameter values of the fitted model do not necessarily determine a valid point process. Some of the point process models are only defined when the parameter values lie in a certain subset. For example the Strauss process only exists when the interaction parameter γ is less than or equal to 1, corresponding to a value of `ppm()$theta[2]` less than or equal to `0`.

By default (if `emend=FALSE`) the algorithm maximises the pseudolikelihood without constraining the parameters, and does not apply any checks for sanity after fitting the model. This is because the fitted parameter value could be useful information for data analysis. To constrain the parameters to ensure that the model is a valid point process, set `emend=TRUE`. See also the functions `valid.ppm` and `emend.ppm`.

The `trend` formula should not use any variable names beginning with the prefixes `.mpl` or `Interaction` as these names are reserved for internal use. The data frame `covariates` should have as many rows as there are points in `Q`. It should not contain variables called `x`, `y` or `marks` as these names are reserved for the Cartesian coordinates and the marks.

If the model formula involves one of the functions `poly()`, `bs()` or `ns()` (e.g. applied to spatial coordinates `x` and `y`), the fitted coefficients can be misleading. The resulting fit is not to the raw spatial variates (`x`, `x^2`, `x*y`, etc.) but to a transformation of these variates. The transformation is implemented by `poly()` in order to achieve better numerical stability. However the resulting coefficients are appropriate for use with the transformed variates, not with the raw variates. This affects the interpretation of the constant term in the fitted model, `logbeta`. Conventionally, β is the background intensity, i.e. the value taken by the conditional intensity function when all predictors (including spatial or "trend" predictors) are set equal to 0. However the coefficient actually produced is the value that the log conditional intensity takes when all the predictors, including the *transformed* spatial predictors, are set equal to `0`, which is not the same thing.

Worse still, the result of `predict.ppm` can be completely wrong if the trend formula contains one of the functions `poly()`, `bs()` or `ns()`. This is a weakness of the underlying function `predict.glm`.

If you wish to fit a polynomial trend, we offer an alternative to `poly()`, namely `polynom()`, which avoids the difficulty induced by transformations. It is completely analogous to `poly` except that it

does not orthonormalise. The resulting coefficient estimates then have their natural interpretation and can be predicted correctly. Numerical stability may be compromised.

Values of the maximised pseudolikelihood are not comparable if they have been obtained with different values of rbord.

Author(s)

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

References

Baddeley, A., Coeurjolly, J.-F., Rubak, E. and Waagepetersen, R. (2014) Logistic regression for spatial Gibbs point processes. *Biometrika* **101** (2) 377–392.

Baddeley, A. and Turner, R. Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42** (2000) 283–322.

Berman, M. and Turner, T.R. Approximating point process likelihoods with GLIM. *Applied Statistics* **41** (1992) 31–38.

Besag, J. Statistical analysis of non-lattice data. *The Statistician* **24** (1975) 179-195.

Diggle, P.J., Fiksel, T., Grabarnik, P., Ogata, Y., Stoyan, D. and Tanemura, M. On parameter estimation for pairwise interaction processes. *International Statistical Review* **62** (1994) 99-117.

Huang, F. and Ogata, Y. Improvements of the maximum pseudo-likelihood estimators in various spatial statistical models. *Journal of Computational and Graphical Statistics* **8** (1999) 510-530.

Jensen, J.L. and Moeller, M. Pseudolikelihood for exponential family models of spatial point processes. *Annals of Applied Probability* **1** (1991) 445–461.

Jensen, J.L. and Kuensch, H.R. On asymptotic normality of pseudo likelihood estimates for pairwise interaction processes, *Annals of the Institute of Statistical Mathematics* **46** (1994) 475-486.

Rajala T. (2014) *A note on Bayesian logistic regression for spatial exponential family Gibbs point processes*, Preprint on ArXiv.org. https://arxiv.org/abs/1411.0539

See Also

ppm.object for details of how to print, plot and manipulate a fitted model.

ppp and quadscheme for constructing data.

Interactions: AreaInter, BadGey, Concom, DiggleGatesStibbard, DiggleGratton, Fiksel, Geyer, Hardcore, HierHard, HierStrauss, HierStraussHard, Hybrid, LennardJones, MultiHard, MultiStrauss, MultiStraussHard, OrdThresh, Ord, Pairwise, PairPiece, Penttinen, Poisson, Saturated, SatPiece, Softcore, Strauss, StraussHard and Triplets.

See profilepl for advice on fitting nuisance parameters in the interaction, and ippm for irregular parameters in the trend.

See valid.ppm and emend.ppm for ensuring the fitted model is a valid point process.

Examples

```
# fit the stationary Poisson process
# to point pattern 'nztrees'

ppm(nztrees)
ppm(nztrees ~ 1)
# equivalent.

Q <- quadscheme(nztrees)
ppm(Q)
# equivalent.

fit1 <- ppm(nztrees, ~ x)
# fit the nonstationary Poisson process
# with intensity function lambda(x,y) = exp(a + bx)
# where x,y are the Cartesian coordinates
# and a,b are parameters to be estimated

# For other examples, see help(ppm)
```

---

ppmInfluence *Leverage and Influence Measures for Spatial Point Process Model*

Description

Calculates all the leverage and influence measures described in influence.ppm, leverage.ppm and dfbetas.ppm.

Usage

```
ppmInfluence(fit,
             what = c("leverage", "influence", "dfbetas"),
             ...,
             iScore = NULL, iHessian = NULL, iArgs = NULL,
             drop = FALSE,
             fitname = NULL)
```

Arguments

fit	A fitted point process model of class "ppm".
what	Character vector specifying which quantities are to be calculated. Default is to calculate all quantities.
...	Ignored.
iScore,iHessian	Components of the score vector and Hessian matrix for the irregular parameters, if required. See Details.
iArgs	List of extra arguments for the functions iScore, iHessian if required.

drop	Logical. Whether to include (drop=FALSE) or exclude (drop=TRUE) contributions from quadrature points that were not used to fit the model.
fitname	Optional character string name for the fitted model fit.

Details

This function calculates all the leverage and influence measures described in influence.ppm, leverage.ppm and dfbetas.ppm.

When analysing large datasets, the user can call ppmInfluence to perform the calculations efficiently, then extract the leverage and influence values as desired. For example the leverage can be extracted either as result$leverage or leverage(result).

If the point process model trend has irregular parameters that were fitted (using ippm) then the influence calculation requires the first and second derivatives of the log trend with respect to the irregular parameters. The argument iScore should be a list, with one entry for each irregular parameter, of R functions that compute the partial derivatives of the log trend (i.e. log intensity or log conditional intensity) with respect to each irregular parameter. The argument iHessian should be a list, with p^2 entries where p is the number of irregular parameters, of R functions that compute the second order partial derivatives of the log trend with respect to each pair of irregular parameters.

Value

A list containing the leverage and influence measures specified by what. The result also belongs to the class "ppmInfluence".

Author(s)

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

See Also

leverage.ppm, influence.ppm, dfbetas.ppm

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X ~ x+y)
fI <- ppmInfluence(fit)

fitlev <- fI$leverage
fitlev <- leverage(fI)

fitinf <- fI$influence
fitinf <- influence(fI)

fitdfb <- fI$dfbetas
fitdfb <- dfbetas(fI)
```

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 <r.turner@auckland.ac.nz> 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)
```

---

predict.dppm *Prediction from a Fitted Determinantal Point Process Model*

Description

Given a fitted determinantal point process model, these functions compute the fitted intensity.

Usage

```
  ## S3 method for class 'dppm'
fitted(object, ...)

  ## S3 method for class 'dppm'
predict(object, ...)
```

Arguments

object	Fitted determinantal point process model. An object of class "dppm".
...	Arguments passed to fitted.ppm or predict.ppm respectively.

Details

These functions are methods for the generic functions fitted and predict. The argument object should be a determinantal point process model (object of class "dppm") obtained using the function dppm.

The *intensity* of the fitted model is computed, using fitted.ppm or predict.ppm respectively.

Value

The value of `fitted.dppm` is a numeric vector giving the fitted values at the quadrature points.

The value of `predict.dppm` is usually a pixel image (object of class `"im"`), but see `predict.ppm` for details.

Author(s)

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

See Also

`dppm`, `plot.dppm`, `fitted.ppm`, `predict.ppm`

Examples

```
if(interactive()) {
  fit <- dppm(swedishpines ~ x + y, dppGauss())
} else {
  fit <- dppm(redwood ~ x, dppGauss())
}
predict(fit)
```

---

predict.kppm *Prediction from a Fitted Cluster Point Process Model*

Description

Given a fitted cluster point process model, these functions compute the fitted intensity.

Usage

```
  ## S3 method for class 'kppm'
fitted(object, ...)

  ## S3 method for class 'kppm'
predict(object, ...)
```

Arguments

`object`	Fitted cluster point process model. An object of class `"kppm"`.
`...`	Arguments passed to `fitted.ppm` or `predict.ppm` respectively.

Details

These functions are methods for the generic functions `fitted` and `predict`. The argument `object` should be a cluster point process model (object of class `"kppm"`) obtained using the function `kppm`.

The *intensity* of the fitted model is computed, using `fitted.ppm` or `predict.ppm` respectively.

Value

The value of `fitted.kppm` is a numeric vector giving the fitted values at the quadrature points.

The value of `predict.kppm` is usually a pixel image (object of class `"im"`), but see `predict.ppm` for details.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`kppm`, `plot.kppm`, `vcov.kppm`, `fitted.ppm`, `predict.ppm`

Examples

```
data(redwood)
fit <- kppm(redwood ~ x, "Thomas")
predict(fit)
```

`predict.mppm` *Prediction for Fitted Multiple Point Process Model*

Description

Given a fitted multiple point process model obtained by `mppm`, evaluate the spatial trend and/or the conditional intensity of the model. By default, predictions are evaluated over a grid of locations, yielding pixel images of the trend and conditional intensity. Alternatively predictions may be evaluated at specified locations with specified values of the covariates.

Usage

```
## S3 method for class 'mppm'
predict(object, ..., newdata = NULL, type = c("trend", "cif"),
             ngrid = 40, locations=NULL, verbose=FALSE)
```

Arguments

`object`	The fitted model. An object of class `"mppm"` obtained from `mppm`.
`...`	Ignored.
`newdata`	Optional. New values of the covariates, for which the predictions should be computed. See Details.
`type`	Type of predicted values required. A character string or vector of character strings. Options are `"trend"` for the spatial trend (first-order term) and `"cif"` or `"lambda"` for the conditional intensity. Alternatively `type="all"` selects all options.

ngrid	Dimensions of the grid of spatial locations at which prediction will be performed (if locations=NULL). An integer or a pair of integers.
locations	Optional. The locations at which predictions should be performed. A list of point patterns, with one entry for each row of newdata.
verbose	Logical flag indicating whether to print progress reports.

Details

This function computes the spatial trend and the conditional intensity of a spatial point process model that has been fitted to several spatial point patterns. See Chapter 16 of Baddeley, Rubak and Turner (2015) for explanation and examples.

Note that by "spatial trend" we mean the (exponentiated) first order potential and not the intensity of the process. [For example if we fit the stationary Strauss process with parameters β and γ, then the spatial trend is constant and equal to β.] The conditional intensity $\lambda(u, X)$ of the fitted model is evaluated at each required spatial location u, with respect to the response point pattern X.

If newdata=NULL, predictions are computed for the original values of the covariates, to which the model was fitted. Otherwise newdata should be a hyperframe (see hyperframe) containing columns of covariates as required by the model. If type includes "cif", then newdata must also include a column of spatial point pattern responses, in order to compute the conditional intensity.

If locations=NULL, then predictions are performed at an ngrid by ngrid grid of locations in the window for each response point pattern. The result will be a hyperframe containing a column of images of the trend (if selected) and a column of images of the conditional intensity (if selected). The result can be plotted.

If locations is given, then it should be a list of point patterns (objects of class "ppp"). Predictions are performed at these points, and the results are returned as mark values attached to the locations. The result is a hyperframe containing columns called trend and/or cif. The column called trend contains marked point patterns in which the point locations are the locations and the mark value is the predicted trend. The column called cif contains marked point patterns in which the point locations are the locations and the mark value is the predicted conditional intensity.

Value

A hyperframe with columns named trend and/or cif.

If locations=NULL, the entries of the hyperframe are pixel images.

If locations is not null, the entries are marked point patterns constructed by attaching the predicted values to the locations point patterns.

Models that depend on row number

The point process model that is described by an mppm object may be a different point process for each row of the original hyperframe of data. This occurs if the model formula includes the variable id (representing row number) or if the model has a different interpoint interaction on each row.

If the point process model is different on each row of the original data, then either

- newdata is missing. Predictions are computed for each row of the original data using the point process model that applies on each row.

- newdata must have the same number of rows as the original data. Each row of newdata is assumed to be a replacement for the corresponding row of the original data. The prediction for row i of newdata will be computed for the point process model that applies to row i of the original data.
- newdata must include a column called id specifying the row number, and therefore identifying which of the point process models should apply. The predictions for row i of newdata will be computed for the point process model that applies to row k of the original data, where k = newdata$id[i].

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Ida-Maria Sintorn and Leanne Bischoff. Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A. and Turner, R. Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42** (2000) 283–322.

Baddeley, A., Bischof, L., Sintorn, I.-M., Haggarty, S., Bell, M. and Turner, R. Analysis of a designed experiment where the response is a spatial point pattern. In preparation.

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

See Also

mppm, fitted.mppm, hyperframe

Examples

```
h <- hyperframe(Bugs=waterstriders)
fit <- mppm(Bugs ~ x, data=h, interaction=Strauss(7))
# prediction on a grid
p <- predict(fit)
plot(p$trend)
# prediction at specified locations
loc <- with(h, runifpoint(20, Window(Bugs)))
p2 <- predict(fit, locations=loc)
plot(p2$trend)
```

---

predict.ppm *Prediction from a Fitted Point Process Model*

Description

Given a fitted point process model obtained by ppm, evaluate the spatial trend or the conditional intensity of the model at new locations.

Usage

```
   ## S3 method for class 'ppm'
predict(object, window=NULL, ngrid=NULL, locations=NULL,
   covariates=NULL,
   type=c("trend", "cif", "intensity", "count"),
   se=FALSE,
   interval=c("none", "confidence", "prediction"),
   level = 0.95,
   X=data.ppm(object), correction, ignore.hardcore=FALSE,
   ...,
   dimyx=NULL, eps=NULL,
   new.coef=NULL, check=TRUE, repair=TRUE)
```

Arguments

object	A fitted point process model, typically obtained from the model-fitting algorithm ppm. An object of class "ppm" (see ppm.object).
window	Optional. A window (object of class "owin") *delimiting* the locations where predictions should be computed. Defaults to the window of the original data used to fit the model object.
ngrid	Optional. Dimensions of a rectangular grid of locations inside window where the predictions should be computed. An integer, or an integer vector of length 2, specifying the number of grid points in the y and x directions. (Incompatible with locations. Equivalent to dimyx.)
locations	Optional. Data giving the exact x, y coordinates (and marks, if required) of locations at which predictions should be computed. Either a point pattern, or a data frame with columns named x and y, or a binary image mask, or a pixel image. (Incompatible with ngrid, dimyx and eps).
covariates	Values of external covariates required by the model. Either a data frame or a list of images. See Details.
type	Character string. Indicates which property of the fitted model should be predicted. Options are "trend" for the spatial trend, "cif" or "lambda" for the conditional intensity, "intensity" for the intensity, and "count" for the total number of points in window.
se	Logical value indicating whether to calculate standard errors as well.
interval	String (partially matched) indicating whether to produce estimates (interval="none", the default) or a confidence interval (interval="confidence") or a prediction interval (interval="prediction").
level	Coverage probability for the confidence or prediction interval.
X	Optional. A point pattern (object of class "ppp") to be taken as the data point pattern when calculating the conditional intensity. The default is to use the original data to which the model was fitted.
correction	Name of the edge correction to be used in calculating the conditional intensity. Options include "border" and "none". Other options may include "periodic", "isotropic" and "translate" depending on the model. The default correction is the one that was used to fit object.

`ignore.hardcore`
: Advanced use only. Logical value specifying whether to compute only the finite part of the interaction potential (effectively removing any hard core interaction terms).

`...`
: Ignored.

`dimyx`
: Equivalent to `ngrid`.

`eps`
: Width and height of pixels in the prediction grid. A numerical value, or numeric vector of length 2.

`new.coef`
: Numeric vector of parameter values to replace the fitted model parameters `coef(object)`.

`check`
: Logical value indicating whether to check the internal format of `object`. If there is any possibility that this object has been restored from a dump file, or has otherwise lost track of the environment where it was originally computed, set `check=TRUE`.

`repair`
: Logical value indicating whether to repair the internal format of `object`, if it is found to be damaged.

Details

This function computes properties of a fitted spatial point process model (object of class `"ppm"`). For a Poisson point process it can compute the fitted intensity function, or the expected number of points in a region. For a Gibbs point process it can compute the spatial trend (first order potential), conditional intensity, and approximate intensity of the process. Point estimates, standard errors, confidence intervals and prediction intervals are available.

Given a point pattern dataset, we may fit a point process model to the data using the model-fitting algorithm `ppm`. This returns an object of class `"ppm"` representing the fitted point process model (see `ppm.object`). The parameter estimates in this fitted model can be read off simply by printing the `ppm` object. The spatial trend, conditional intensity and intensity of the fitted model are evaluated using this function `predict.ppm`.

The default action is to create a rectangular grid of points in the observation window of the data point pattern, and evaluate the spatial trend at these locations.

The argument `type` specifies the values that are desired:

If `type="trend"`**:** the "spatial trend" of the fitted model is evaluated at each required spatial location u. See below.

If `type="cif"`**:** the conditional intensity $\lambda(u, X)$ of the fitted model is evaluated at each required spatial location u, with respect to the data point pattern X.

If `type="intensity"`**:** the intensity $\lambda(u)$ of the fitted model is evaluated at each required spatial location u.

If `type="count"`**:** the expected total number of points (or the expected number of points falling in `window`) is evaluated. If `window` is a tessellation, the expected number of points in each tile of the tessellation is evaluated.

The spatial trend, conditional intensity, and intensity are all equivalent if the fitted model is a Poisson point process. However, if the model is not a Poisson process, then they are all different. The "spatial trend" is the (exponentiated) first order potential, and not the intensity of the process. [For

example if we fit the stationary Strauss process with parameters β and γ, then the spatial trend is constant and equal to β, while the intensity is a smaller value.]

The default is to compute an estimate of the desired quantity. If `interval="confidence"` or `interval="prediction"`, the estimate is replaced by a confidence interval or prediction interval.

If `se=TRUE`, then a standard error is also calculated, and is returned together with the (point or interval) estimate.

The spatial locations where predictions are required, are determined by the (incompatible) arguments `ngrid` and `locations`.

- If the argument `ngrid` is present, then predictions are performed at a rectangular grid of locations in the window `window`. The result of prediction will be a pixel image or images.
- If `locations` is present, then predictions will be performed at the spatial locations given by this dataset. These may be an arbitrary list of spatial locations, or they may be a rectangular grid. The result of prediction will be either a numeric vector or a pixel image or images.
- If neither `ngrid` nor `locations` is given, then `ngrid` is assumed. The value of `ngrid` defaults to `spatstat.options("npixel")`, which is initialised to 128 when **spatstat** is loaded.

The argument `locations` may be a point pattern, a data frame or a list specifying arbitrary locations; or it may be a binary image mask (an object of class `"owin"` with type `"mask"`) or a pixel image (object of class `"im"`) specifying (a subset of) a rectangular grid of locations.

- If `locations` is a point pattern (object of class `"ppp"`), then prediction will be performed at the points of the point pattern. The result of prediction will be a vector of predicted values, one value for each point. If the model is a marked point process, then `locations` should be a marked point pattern, with marks of the same kind as the model; prediction will be performed at these marked points. The result of prediction will be a vector of predicted values, one value for each (marked) point.
- If `locations` is a data frame or list, then it must contain vectors `locations$x` and `locations$y` specifying the x, y coordinates of the prediction locations. Additionally, if the model is a marked point process, then `locations` must also contain a factor `locations$marks` specifying the marks of the prediction locations. These vectors must have equal length. The result of prediction will be a vector of predicted values, of the same length.
- If `locations` is a binary image mask, then prediction will be performed at each pixel in this binary image where the pixel value is `TRUE` (in other words, at each pixel that is inside the window). If the fitted model is an unmarked point process, then the result of prediction will be an image. If the fitted model is a marked point process, then prediction will be performed for each possible value of the mark at each such location, and the result of prediction will be a list of images, one for each mark value.
- If `locations` is a pixel image (object of class `"im"`), then prediction will be performed at each pixel in this image where the pixel value is defined (i.e.\ where the pixel value is not `NA`).

The argument `covariates` gives the values of any spatial covariates at the prediction locations. If the trend formula in the fitted model involves spatial covariates (other than the Cartesian coordinates `x`, `y`) then `covariates` is required. The format and use of `covariates` are analogous to those of the argument of the same name in `ppm`. It is either a data frame or a list of images.

- If `covariates` is a list of images, then the names of the entries should correspond to the names of covariates in the model formula `trend`. Each entry in the list must be an image object (of

class "im", see im.object). The software will look up the pixel values of each image at the quadrature points.

- If covariates is a data frame, then the ith row of covariates is assumed to contain covariate data for the ith location. When locations is a data frame, this just means that each row of covariates contains the covariate data for the location specified in the corresponding row of locations. When locations is a binary image mask, the row covariates[i,] must correspond to the location x[i],y[i] where x = as.vector(raster.x(locations)) and y = as.vector(raster.y(locations)).

Note that if you only want to use prediction in order to generate a plot of the predicted values, it may be easier to use plot.ppm which calls this function and plots the results.

Value

If total *is given:* a numeric vector or matrix.

If locations *is given and is a data frame:* a vector of predicted values for the spatial locations (and marks, if required) given in locations.

If ngrid *is given, or if* locations *is given and is a binary image mask or a pixel image:* If object is an unmarked point process, the result is a pixel image object (of class "im", see im.object) containing the predictions. If object is a multitype point process, the result is a list of pixel images, containing the predictions for each type at the same grid of locations.

The "predicted values" are either values of the spatial trend (if type="trend"), values of the conditional intensity (if type="cif" or type="lambda"), values of the intensity (if type="intensity") or numbers of points (if type="count").

If se=TRUE, then the result is a list with two entries, the first being the predicted values in the format described above, and the second being the standard errors in the same format.

Warnings

The current implementation invokes predict.glm so that **prediction is wrong** if the trend formula in object involves terms in ns(), bs() or poly(). This is a weakness of predict.glm itself!

Error messages may be very opaque, as they tend to come from deep in the workings of predict.glm. If you are passing the covariates argument and the function crashes, it is advisable to start by checking that all the conditions listed above are satisfied.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A. and Turner, R. Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42** (2000) 283–322.

Berman, M. and Turner, T.R. Approximating point process likelihoods with GLIM. *Applied Statistics* **41** (1992) 31–38.

See Also

ppm, ppm.object, plot.ppm, print.ppm, fitted.ppm, spatstat.options

Examples

```
m <- ppm(cells ~ polynom(x,y,2), Strauss(0.05))
trend <- predict(m, type="trend")
if(human <- interactive()) {
image(trend)
points(cells)
}
cif <- predict(m, type="cif")
if(human) {
persp(cif)
}
mj <- ppm(japanesepines ~ harmonic(x,y,2))
se <- predict(mj, se=TRUE) # image of standard error
ci <- predict(mj, interval="c") # two images, confidence interval

# prediction interval for total number of points
predict(mj, type="count", interval="p")

# prediction intervals for counts in tiles
predict(mj, window=quadrats(japanesepines, 3), type="count", interval="p")

# prediction at arbitrary locations
predict(mj, locations=data.frame(x=0.3, y=0.4))

X <- runifpoint(5, Window(japanesepines))
predict(mj, locations=X, se=TRUE)

# multitype
rr <- matrix(0.06, 2, 2)
ma <- ppm(amacrine ~ marks,  MultiStrauss(rr))
Z <- predict(ma)
Z <- predict(ma, type="cif")
predict(ma, locations=data.frame(x=0.8, y=0.5,marks="on"), type="cif")
```

---

predict.rppm *Make Predictions From a Recursively Partitioned Point Process Model*

Description

Given a model which has been fitted to point pattern data by recursive partitioning, compute the predicted intensity of the model.

Usage

```
## S3 method for class 'rppm'
predict(object, ...)

## S3 method for class 'rppm'
fitted(object, ...)
```

Arguments

object	Fitted point process model of class "rppm" produced by the function rppm.
...	Optional arguments passed to predict.ppm to specify the locations where prediction is required. (Ignored by fitted.rppm)

Details

These functions are methods for the generic functions fitted and predict. They compute the fitted intensity of a point process model. The argument object should be a fitted point process model of class "rppm" produced by the function rppm.

The fitted method computes the fitted intensity at the original data points, yielding a numeric vector with one entry for each data point.

The predict method computes the fitted intensity at any locations. By default, predictions are calculated at a regular grid of spatial locations, and the result is a pixel image giving the predicted intensity values at these locations.

Alternatively, predictions can be performed at other locations, or a finer grid of locations, or only at certain specified locations, using additional arguments ... which will be interpreted by predict.ppm. Common arguments are ngrid to increase the grid resolution, window to specify the prediction region, and locations to specify the exact locations of predictions. See predict.ppm for details of these arguments.

Predictions are computed by evaluating the explanatory covariates at each desired location, and applying the recursive partitioning rule to each set of covariate values.

Value

The result of fitted.rppm is a numeric vector.

The result of predict.rppm is a pixel image, a list of pixel images, or a numeric vector.

Author(s)

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

See Also

rppm, plot.rppm

Examples

```
fit <- rppm(unmark(gorillas) ~ vegetation, data=gorillas.extra)
plot(predict(fit))
lambdaX <- fitted(fit)
lambdaX[1:5]
# Mondriaan pictures
plot(predict(rppm(redwoodfull ~ x + y)))
points(redwoodfull)
```

`predict.slrm` *Predicted or Fitted Values from Spatial Logistic Regression*

Description

Given a fitted Spatial Logistic Regression model, this function computes the fitted probabilities for each pixel, or the fitted point process intensity, or the values of the linear predictor in each pixel.

Usage

```
 ## S3 method for class 'slrm'
predict(object, ..., type = "intensity",
                        newdata=NULL, window=NULL)
```

Arguments

`object`	a fitted spatial logistic regression model. An object of class `"slrm"`.
`...`	Optional arguments passed to `pixellate` determining the pixel resolution for the discretisation of the point pattern.
`type`	Character string (partially) matching one of `"probabilities"`, `"intensity"` or `"link"`.
`newdata`	Optional. List containing new covariate values for the prediction. See Details.
`window`	Optional. New window in which to predict. An object of class `"owin"`.

Details

This is a method for `predict` for spatial logistic regression models (objects of class `"slrm"`, usually obtained from the function `slrm`).

The argument `type` determines which quantity is computed. If `type="intensity"`), the value of the point process intensity is computed at each pixel. If `type="probabilities"`) the probability of the presence of a random point in each pixel is computed. If `type="link"`, the value of the linear predictor is computed at each pixel.

If `newdata = NULL` (the default), the algorithm computes fitted values of the model (based on the data that was originally used to fit the model `object`).

If `newdata` is given, the algorithm computes predicted values of the model, using the new values of the covariates provided by `newdata`. The argument `newdata` should be a list; names of entries in the list should correspond to variables appearing in the model formula of the `object`. Each list entry may be a pixel image or a single numeric value.

Value

A pixel image (object of class "im") containing the predicted values for each pixel.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> <adrian@maths.uwa.edu.au>
and Rolf Turner <r.turner@auckland.ac.nz>

See Also

slrm

Examples

```
X <- rpoispp(42)
fit <- slrm(X ~ x+y)
plot(predict(fit))

data(copper)
X <- copper$SouthPoints
Y <- copper$SouthLines
Z <- distmap(Y)
fitc <- slrm(X ~ Z)
pc <- predict(fitc)

Znew <- distmap(copper$Lines)[copper$SouthWindow]
pcnew <- predict(fitc, newdata=list(Z=Znew))
```

---

print.ppm *Print a Fitted Point Process Model*

Description

Default print method for a fitted point process model.

Usage

```
 ## S3 method for class 'ppm'
print(x,...,
     what=c("all", "model", "trend", "interaction", "se", "errors"))
```

Arguments

x	A fitted point process model, typically obtained from the model-fittingg algorithm ppm. An object of class "ppm".
what	Character vector (partially-matched) indicating what information should be printed.
...	Ignored.

Details

This is the `print` method for the class `"ppm"`. It prints information about the fitted model in a sensible format.

The argument `what` makes it possible to print only some of the information.

If `what` is missing, then by default, standard errors for the estimated coefficients of the model will be printed only if the model is a Poisson point process. To print the standard errors for a non-Poisson model, call `print.ppm` with the argument `what` given explicitly, or reset the default rule by typing `spatstat.options(print.ppm.SE="always")`.

Value

none.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`ppm.object` for details of the class `"ppm"`.

`ppm` for generating these objects.

`plot.ppm`, `predict.ppm`

Examples

```
# m <- ppm(cells ~1, Strauss(0.05))
# m
```

`profilepl` *Fit Models by Profile Maximum Pseudolikelihood or AIC*

Description

Fits point process models by maximising the profile likelihood, profile pseudolikelihood, profile composite likelihood or AIC.

Usage

```
profilepl(s, f, ..., aic=FALSE, rbord=NULL, verbose = TRUE, fast=TRUE)
```

Arguments

s	Data frame containing values of the irregular parameters over which the criterion will be computed.
f	Function (such as Strauss) that generates an interpoint interaction object, given values of the irregular parameters.
...	Data passed to ppm to fit the model.
aic	Logical value indicating whether to find the parameter values which minimise the AIC (aic=TRUE) or maximise the profile likelihood (aic=FALSE, the default).
rbord	Radius for border correction (same for all models). If omitted, this will be computed from the interactions.
verbose	Logical value indicating whether to print progress reports.
fast	Logical value indicating whether to use a faster, less accurate model-fitting technique when computing the profile pseudolikelihood. See Section on Speed and Accuracy.

Details

The model-fitting function ppm fits point process models to point pattern data. However, only the 'regular' parameters of the model can be fitted by ppm. The model may also depend on 'irregular' parameters that must be fixed in any call to ppm.

This function profilepl is a wrapper which finds the values of the irregular parameters that give the best fit. If aic=FALSE (the default), the best fit is the model which maximises the likelihood (if the models are Poisson processes) or maximises the pseudolikelihood or logistic likelihood. If aic=TRUE then the best fit is the model which minimises the Akaike Information Criterion AIC.ppm.

The argument s must be a data frame whose columns contain values of the irregular parameters over which the maximisation is to be performed.

An irregular parameter may affect either the interpoint interaction or the spatial trend.

interaction parameters: in a call to ppm, the argument interaction determines the interaction between points. It is usually a call to a function such as Strauss. The arguments of this call are irregular parameters. For example, the interaction radius parameter r of the Strauss process, determined by the argument r to the function Strauss, is an irregular parameter.

trend parameters: in a call to ppm, the spatial trend may depend on covariates, which are supplied by the argument covariates. These covariates may be functions written by the user, of the form function(x,y,...), and the extra arguments ... are irregular parameters.

The argument f determines the interaction for each model to be fitted. It would typically be one of the functions Poisson, AreaInter, BadGey, DiggleGatesStibbard, DiggleGratton, Fiksel, Geyer, Hardcore, LennardJones, OrdThresh, Softcore, Strauss or StraussHard. Alternatively it could be a function written by the user.

Columns of s which match the names of arguments of f will be interpreted as interaction parameters. Other columns will be interpreted as trend parameters.

The data frame s must provide values for each argument of f, except for the optional arguments, which are those arguments of f that have the default value NA.

To find the best fit, each row of s will be taken in turn. Interaction parameters in this row will be passed to f, resulting in an interaction object. Then ppm will be applied to the data ... using this interaction. Any trend parameters will be passed to ppm through the argument covfunargs. This results in a fitted point process model. The value of the log pseudolikelihood or AIC from this model is stored. After all rows of s have been processed in this way, the row giving the maximum value of log pseudolikelihood will be found.

The object returned by profilepl contains the profile pseudolikelihood (or profile AIC) function, the best fitting model, and other data. It can be plotted (yielding a plot of the log pseudolikelihood or AIC values against the irregular parameters) or printed (yielding information about the best fitting values of the irregular parameters).

In general, f may be any function that will return an interaction object (object of class "interact") that can be used in a call to ppm. Each argument of f must be a single value.

Value

An object of class "profilepl". There are methods for plot, print, summary, simulate, as.ppm, fitin and parameters for objects of this class.

The components of the object include

fit	Best-fitting model
param	The data frame s
iopt	Row index of the best-fitting parameters in s

To extract the best fitting model you can also use as.ppm.

Speed and Accuracy

Computation of the profile pseudolikelihood can be time-consuming. We recommend starting with a small experiment in which s contains only a few rows of values. This will indicate roughly the optimal values of the parameters. Then a full calculation using more finely spaced values can identify the exact optimal values.

It is normal that the procedure appears to slow down at the end. During the computation of the profile pseudolikelihood, the model-fitting procedure is accelerated by omitting some calculations that are not needed for computing the pseudolikelihood. When the optimal parameter values have been identified, they are used to fit the final model in its entirety. Fitting the final model can take longer than computing the profile pseudolikelihood.

If fast=TRUE (the default), then additional shortcuts are taken in order to accelerate the computation of the profile log pseudolikelihood. These shortcuts mean that the values of the profile log pseudolikelihood in the result ($prof) may not be equal to the values that would be obtained if the model was fitted normally. Currently this happens only for the area interaction AreaInter. It may be wise to do a small experiment with fast=TRUE and then a definitive calculation with fast=FALSE.

Author(s)

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

References

Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.

See Also

plot.profilepl

Examples

```
# one irregular parameter
rr <- data.frame(r=seq(0.05,0.15, by=0.01))

ps <- profilepl(rr, Strauss, cells)
ps
plot(ps)

# two irregular parameters
rs <- expand.grid(r=seq(0.05,0.15, by=0.01),sat=1:3)

pg <- profilepl(rs, Geyer, cells)
pg
as.ppm(pg)

## more information
summary(pg)

# multitype pattern with a common interaction radius
# RR <- data.frame(R=seq(0.03,0.05,by=0.01))
# MS <- function(R) { MultiStrauss(radii=diag(c(R,R))) }
# pm <- profilepl(RR, MS, amacrine ~marks)
```

---

prune.rppm *Prune a Recursively Partitioned Point Process Model*

Description

Given a model which has been fitted to point pattern data by recursive partitioning, apply pruning to reduce the complexity of the partition tree.

Usage

```
## S3 method for class 'rppm'
prune(tree, ...)
```

Arguments

tree	Fitted point process model of class "rppm" produced by the function rppm.
...	Arguments passed to prune.rpart to control the pruning procedure.

Details

This is a method for the generic function prune for the class "rppm". An object of this class is a point process model, fitted to point pattern data by recursive partitioning, by the function rppm.

The recursive partition tree will be pruned using prune.rpart. The result is another object of class "rppm".

Value

Object of class "rppm".

Author(s)

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

See Also

rppm, plot.rppm, predict.rppm.

Examples

```
# Murchison gold data
mur <- solapply(murchison, rescale, s=1000, unitname="km")
mur$dfault <- distfun(mur$faults)
fit <- rppm(gold ~ dfault + greenstone, data=mur)
fit
prune(fit, cp=0.1)
```

---

pseudoR2 *Calculate Pseudo-R-Squared for Point Process Model*

Description

Given a fitted point process model, calculate the pseudo-R-squared value, which measures the fraction of variation in the data that is explained by the model.

Usage

```
  pseudoR2(object, ...)

  ## S3 method for class 'ppm'
pseudoR2(object, ..., keepoffset=TRUE)

  ## S3 method for class 'slrm'
pseudoR2(object, ..., keepoffset=TRUE)
```

Arguments

object	Fitted point process model. An object of class "ppm" or "slrm".
keepoffset	Logical value indicating whether to retain offset terms in the model when computing the deviance difference. See Details.
...	Additional arguments passed to deviance.ppm or deviance.slrm.

Details

The function pseudoR2 is generic, with methods for fitted point process models of class "ppm" and "slrm".

This function computes McFadden's pseudo-Rsquared

$$R^2 = 1 - \frac{D}{D_0}$$

where D is the deviance of the fitted model object, and D_0 is the deviance of the null model. Deviance is defined as twice the negative log-likelihood or log-pseudolikelihood.

The null model is usually obtained by re-fitting the model using the trend formula ~1. However if the original model formula included offset terms, and if keepoffset=TRUE (the default), then the null model formula consists of these offset terms. This ensures that the pseudoR2 value is non-negative.

Value

A single numeric value.

Author(s)

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

See Also

deviance.ppm, deviance.slrm.

Examples

```
fit <- ppm(swedishpines ~ x+y)
pseudoR2(fit)

xcoord <- as.im(function(x,y) x, Window(swedishpines))
fut <- ppm(swedishpines ~ offset(xcoord/200) + y)
pseudoR2(fut)
```

psib *Sibling Probability of Cluster Point Process*

Description

Computes the sibling probability of a cluster point process model.

Usage

```
psib(object)

## S3 method for class 'kppm'
psib(object)
```

Arguments

object	Fitted cluster point process model (object of class "kppm").

Details

In a Poisson cluster process, two points are called *siblings* if they belong to the same cluster, that is, if they had the same parent point. If two points of the process are separated by a distance r, the probability that they are siblings is $p(r) = 1 - 1/g(r)$ where g is the pair correlation function of the process.

The value $p(0) = 1 - 1/g(0)$ is the probability that, if two points of the process are situated very close to each other, they came from the same cluster. This probability is an index of the strength of clustering, with high values suggesting strong clustering.

This concept was proposed in Baddeley, Rubak and Turner (2015, page 479) and Baddeley (2017).

Value

A single number.

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

kppm

Examples

```
fit <- kppm(redwood ~1, "Thomas")
psib(fit)
```

---

psst *Pseudoscore Diagnostic For Fitted Model against General Alternative*

Description

Given a point process model fitted to a point pattern dataset, and any choice of functional summary statistic, this function computes the pseudoscore test statistic of goodness-of-fit for the model.

Usage

```
psst(object, fun, r = NULL, breaks = NULL, ...,
     model=NULL,
     trend = ~1, interaction = Poisson(), rbord = reach(interaction),
     truecoef=NULL, hi.res=NULL, funargs = list(correction="best"),
     verbose=TRUE)
```

Arguments

object	Object to be analysed. Either a fitted point process model (object of class "ppm") or a point pattern (object of class "ppp") or quadrature scheme (object of class "quad").
fun	Summary function to be applied to each point pattern.
r	Optional. Vector of values of the argument r at which the function $S(r)$ should be computed. This argument is usually not specified. There is a sensible default.
breaks	Optional alternative to r for advanced use.
...	Ignored.
model	Optional. A fitted point process model (object of class "ppm") to be re-fitted to the data using update.ppm, if object is a point pattern. Overrides the arguments trend,interaction,rbord.
trend,interaction,rbord	Optional. Arguments passed to ppm to fit a point process model to the data, if object is a point pattern. See ppm for details.
truecoef	Optional. Numeric vector. If present, this will be treated as if it were the true coefficient vector of the point process model, in calculating the diagnostic. Incompatible with hi.res.
hi.res	Optional. List of parameters passed to quadscheme. If this argument is present, the model will be re-fitted at high resolution as specified by these parameters. The coefficients of the resulting fitted model will be taken as the true coefficients. Then the diagnostic will be computed for the default quadrature scheme, but using the high resolution coefficients.
funargs	List of additional arguments to be passed to fun.
verbose	Logical value determining whether to print progress reports during the computation.

Details

Let x be a point pattern dataset consisting of points $x_1, \ldots, x_n$ in a window W. Consider a point process model fitted to x, with conditional intensity $\lambda(u, x)$ at location u. For the purpose of testing goodness-of-fit, we regard the fitted model as the null hypothesis. Given a functional summary statistic S, consider a family of alternative models obtained by exponential tilting of the null model by S. The pseudoscore for the null model is

$$V(r) = \sum_i \Delta S(x_i, x, r) - \int_W \Delta S(u, x, r) \lambda(u, x) \mathrm{d}u$$

where the Δ operator is

$$\Delta S(u, x, r) = S(x \cup \{u\}, r) - S(x \setminus u, r)$$

the difference between the values of S for the point pattern with and without the point u.

According to the Georgii-Nguyen-Zessin formula, $V(r)$ should have mean zero if the model is correct (ignoring the fact that the parameters of the model have been estimated). Hence $V(r)$ can be used as a diagnostic for goodness-of-fit.

This algorithm computes $V(r)$ by direct evaluation of the sum and integral. It is computationally intensive, but it is available for any summary statistic $S(r)$.

The diagnostic $V(r)$ is also called the **pseudoresidual** of S. On the right hand side of the equation for $V(r)$ given above, the sum over points of x is called the **pseudosum** and the integral is called the **pseudocompensator**.

Value

A function value table (object of class `"fv"`), essentially a data frame of function values.

Columns in this data frame include `dat` for the pseudosum, `com` for the compensator and `res` for the pseudoresidual.

There is a plot method for this class. See `fv.object`.

Author(s)

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

References

Baddeley, A., Rubak, E. and Møller, J. (2011) Score, pseudo-score and residual diagnostics for spatial point process models. *Statistical Science* **26**, 613–646.

See Also

Special cases: `psstA`, `psstG`.

Alternative functions: `Kres`, `Gres`.

Examples

```
if(live <- interactive()) {
  fit0 <- ppm(cells ~ 1)
} else {
  fit0 <- ppm(cells ~ 1, nd=8)
}
G0 <- psst(fit0, Gest)
G0
if(live) plot(G0)
```

---

psstA *Pseudoscore Diagnostic For Fitted Model against Area-Interaction Alternative*

Description

Given a point process model fitted to a point pattern dataset, this function computes the pseudoscore diagnostic of goodness-of-fit for the model, against moderately clustered or moderately inhibited alternatives of area-interaction type.

Usage

```
psstA(object, r = NULL, breaks = NULL, ...,
      model = NULL,
      trend = ~1, interaction = Poisson(),
      rbord = reach(interaction), ppmcorrection = "border",
      correction = "all",
      truecoef = NULL, hi.res = NULL,
      nr=spatstat.options("psstA.nr"),
      ngrid=spatstat.options("psstA.ngrid"))
```

Arguments

object	Object to be analysed. Either a fitted point process model (object of class "ppm") or a point pattern (object of class "ppp") or quadrature scheme (object of class "quad").
r	Optional. Vector of values of the argument r at which the diagnostic should be computed. This argument is usually not specified. There is a sensible default.
breaks	This argument is for internal use only.
...	Extra arguments passed to quadscheme to determine the quadrature scheme, if object is a point pattern.
model	Optional. A fitted point process model (object of class "ppm") to be re-fitted to the data using update.ppm, if object is a point pattern. Overrides the arguments trend,interaction,rbord,ppmcorrection.
trend,interaction,rbord	Optional. Arguments passed to ppm to fit a point process model to the data, if object is a point pattern. See ppm for details.

ppmcorrection	Optional. Character string specifying the edge correction for the pseudolikelihood to be used in fitting the point process model. Passed to ppm.
correction	Optional. Character string specifying which diagnostic quantities will be computed. Options are "all" and "best". The default is to compute all diagnostic quantities.
truecoef	Optional. Numeric vector. If present, this will be treated as if it were the true coefficient vector of the point process model, in calculating the diagnostic. Incompatible with hi.res.
hi.res	Optional. List of parameters passed to quadscheme. If this argument is present, the model will be re-fitted at high resolution as specified by these parameters. The coefficients of the resulting fitted model will be taken as the true coefficients. Then the diagnostic will be computed for the default quadrature scheme, but using the high resolution coefficients.
nr	Optional. Number of r values to be used if r is not specified.
ngrid	Integer. Number of points in the square grid used to compute the approximate area.

Details

This function computes the pseudoscore test statistic which can be used as a diagnostic for goodness-of-fit of a fitted point process model.

Let x be a point pattern dataset consisting of points $x_1, \ldots, x_n$ in a window W. Consider a point process model fitted to x, with conditional intensity $\lambda(u, x)$ at location u. For the purpose of testing goodness-of-fit, we regard the fitted model as the null hypothesis. The alternative hypothesis is a family of hybrid models obtained by combining the fitted model with the area-interaction process (see AreaInter). The family of alternatives includes models that are slightly more regular than the fitted model, and others that are slightly more clustered than the fitted model.

The pseudoscore, evaluated at the null model, is

$$V(r) = \sum_i A(x_i, x, r) - \int_W A(u, x, r)\lambda(u, x)\mathrm{d}u$$

where

$$A(u, x, r) = B(x \cup \{u\}, r) - B(x \setminus u, r)$$

where $B(x, r)$ is the area of the union of the discs of radius r centred at the points of x (i.e. $B(x, r)$ is the area of the dilation of x by a distance r). Thus $A(u, x, r)$ is the *unclaimed area* associated with u, that is, the area of that part of the disc of radius r centred at the point u that is not covered by any of the discs of radius r centred at points of x.

According to the Georgii-Nguyen-Zessin formula, $V(r)$ should have mean zero if the model is correct (ignoring the fact that the parameters of the model have been estimated). Hence $V(r)$ can be used as a diagnostic for goodness-of-fit.

The diagnostic $V(r)$ is also called the **pseudoresidual** of S. On the right hand side of the equation for $V(r)$ given above, the sum over points of x is called the **pseudosum** and the integral is called the **pseudocompensator**.

Value

A function value table (object of class "fv"), essentially a data frame of function values.

Columns in this data frame include dat for the pseudosum, com for the compensator and res for the pseudoresidual.

There is a plot method for this class. See fv.object.

Warning

This computation can take a **very long time**.

To shorten the computation time, choose smaller values of the arguments nr and ngrid, or reduce the values of their defaults spatstat.options("psstA.nr") and spatstat.options("psstA.ngrid").

Computation time is roughly proportional to nr * npoints * ngrid^2 where npoints is the number of points in the point pattern.

Author(s)

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

References

Baddeley, A., Rubak, E. and Møller, J. (2011) Score, pseudo-score and residual diagnostics for spatial point process models. *Statistical Science* **26**, 613–646.

See Also

Alternative functions: psstG, psst, Gres, Kres.

Point process models: ppm.

Options: spatstat.options

Examples

```
if(live <- interactive()) {
  X <- rStrauss(200,0.1,0.05)
} else {
  pso <- spatstat.options(psstA.ngrid=16,psstA.nr=10,
         ndummy.min=16,npixel=32)
  X <- cells
}

plot(psstA(X))
plot(psstA(X, interaction=Strauss(0.05)))

if(!live) spatstat.options(pso)
```

psstG *Pseudoscore Diagnostic For Fitted Model against Saturation Alternative*

Description

Given a point process model fitted to a point pattern dataset, this function computes the pseudoscore diagnostic of goodness-of-fit for the model, against moderately clustered or moderately inhibited alternatives of saturation type.

Usage

```
psstG(object, r = NULL, breaks = NULL, ...,
      model=NULL,
      trend = ~1, interaction = Poisson(), rbord = reach(interaction),
      truecoef = NULL, hi.res = NULL)
```

Arguments

object	Object to be analysed. Either a fitted point process model (object of class "ppm") or a point pattern (object of class "ppp") or quadrature scheme (object of class "quad").
r	Optional. Vector of values of the argument r at which the diagnostic should be computed. This argument is usually not specified. There is a sensible default.
breaks	Optional alternative to r for advanced use.
...	Ignored.
model	Optional. A fitted point process model (object of class "ppm") to be re-fitted to the data using update.ppm, if object is a point pattern. Overrides the arguments trend,interaction,rbord,ppmcorrection.
trend,interaction,rbord	Optional. Arguments passed to ppm to fit a point process model to the data, if object is a point pattern. See ppm for details.
truecoef	Optional. Numeric vector. If present, this will be treated as if it were the true coefficient vector of the point process model, in calculating the diagnostic. Incompatible with hi.res.
hi.res	Optional. List of parameters passed to quadscheme. If this argument is present, the model will be re-fitted at high resolution as specified by these parameters. The coefficients of the resulting fitted model will be taken as the true coefficients. Then the diagnostic will be computed for the default quadrature scheme, but using the high resolution coefficients.

Details

This function computes the pseudoscore test statistic which can be used as a diagnostic for goodness-of-fit of a fitted point process model.

Consider a point process model fitted to x, with conditional intensity $\lambda(u,x)$ at location u. For the purpose of testing goodness-of-fit, we regard the fitted model as the null hypothesis. The alternative hypothesis is a family of hybrid models obtained by combining the fitted model with the Geyer saturation process (see Geyer) with saturation parameter 1. The family of alternatives includes models that are more regular than the fitted model, and others that are more clustered than the fitted model.

For any point pattern x, and any $r > 0$, let $S(x,r)$ be the number of points in x whose nearest neighbour (the nearest other point in x) is closer than r units. Then the pseudoscore for the null model is

$$V(r) = \sum_i \Delta S(x_i, x, r) - \int_W \Delta S(u, x, r) \lambda(u, x) \mathrm{d}u$$

where the Δ operator is

$$\Delta S(u, x, r) = S(x \cup \{u\}, r) - S(x \setminus u, r)$$

the difference between the values of S for the point pattern with and without the point u.

According to the Georgii-Nguyen-Zessin formula, $V(r)$ should have mean zero if the model is correct (ignoring the fact that the parameters of the model have been estimated). Hence $V(r)$ can be used as a diagnostic for goodness-of-fit.

The diagnostic $V(r)$ is also called the **pseudoresidual** of S. On the right hand side of the equation for $V(r)$ given above, the sum over points of x is called the **pseudosum** and the integral is called the **pseudocompensator**.

Value

A function value table (object of class `"fv"`), essentially a data frame of function values.

Columns in this data frame include `dat` for the pseudosum, `com` for the compensator and `res` for the pseudoresidual.

There is a plot method for this class. See `fv.object`.

Author(s)

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

References

Baddeley, A., Rubak, E. and Møller, J. (2011) Score, pseudo-score and residual diagnostics for spatial point process models. *Statistical Science* **26**, 613–646.

See Also

Alternative functions: `psstA`, `psst`, `Kres`, `Gres`.

Examples

```
if(live <- interactive()) {
  X <- rStrauss(200,0.1,0.05)
} else {
  pso <- spatstat.options(ndummy.min=16,npixel=32)
  X <- cells
}

plot(psstG(X))
plot(psstG(X, interaction=Strauss(0.05)))

if(!live) spatstat.options(pso)
```

---

qqplot.ppm *Q-Q Plot of Residuals from Fitted Point Process Model*

Description

Given a point process model fitted to a point pattern, produce a Q-Q plot based on residuals from the model.

Usage

```
qqplot.ppm(fit, nsim=100, expr=NULL, ..., type="raw",
           style="mean", fast=TRUE, verbose=TRUE, plot.it=TRUE,
           dimyx=NULL, nrep=if(fast) 5e4 else 1e5,
           control=update(default.rmhcontrol(fit), nrep=nrep),
           saveall=FALSE,
           monochrome=FALSE,
           limcol=if(monochrome) "black" else "red",
           maxerr=max(100, ceiling(nsim/10)),
           check=TRUE, repair=TRUE, envir.expr)
```

Arguments

fit	The fitted point process model, which is to be assessed using the Q-Q plot. An object of class "ppm". Smoothed residuals obtained from this fitted model will provide the "data" quantiles for the Q-Q plot.
nsim	The number of simulations from the "reference" point process model.
expr	Determines the simulation mechanism which provides the "theoretical" quantiles for the Q-Q plot. See Details.
...	Arguments passed to diagnose.ppm influencing the computation of residuals.
type	String indicating the type of residuals or weights to be used. Current options are "eem" for the Stoyan-Grabarnik exponential energy weights, "raw" for the raw residuals, "inverse" for the inverse-lambda residuals, and "pearson" for the Pearson residuals. A partial match is adequate.

style	Character string controlling the type of Q-Q plot. Options are "classical" and "mean". See Details.
fast	Logical flag controlling the speed and accuracy of computation. Use fast=TRUE for interactive use and fast=FALSE for publication standard plots. See Details.
verbose	Logical flag controlling whether the algorithm prints progress reports during long computations.
plot.it	Logical flag controlling whether the function produces a plot or simply returns a value (silently).
dimyx	Dimensions of the pixel grid on which the smoothed residual field will be calculated. A vector of two integers.
nrep	If control is absent, then nrep gives the number of iterations of the Metropolis-Hastings algorithm that should be used to generate one simulation of the fitted point process.
control	List of parameters controlling the Metropolis-Hastings algorithm rmh which generates each simulated realisation from the model (unless the model is Poisson). This list becomes the argument control of rmh.default. It overrides nrep.
saveall	Logical flag indicating whether to save all the intermediate calculations.
monochrome	Logical flag indicating whether the plot should be in black and white (monochrome=TRUE), or in colour (monochrome=FALSE).
limcol	String. The colour to be used when plotting the 95% limit curves.
maxerr	Maximum number of failures tolerated while generating simulated realisations. See Details.
check	Logical value indicating whether to check the internal format of fit. If there is any possibility that this object has been restored from a dump file, or has otherwise lost track of the environment where it was originally computed, set check=TRUE.
repair	Logical value indicating whether to repair the internal format of fit, if it is found to be damaged.
envir.expr	Optional. An environment in which the expression expr should be evaluated.

Details

This function generates a Q-Q plot of the residuals from a fitted point process model. It is an addendum to the suite of diagnostic plots produced by the function diagnose.ppm, kept separate because it is computationally intensive. The quantiles of the theoretical distribution are estimated by simulation.

In classical statistics, a Q-Q plot of residuals is a useful diagnostic for checking the distributional assumptions. Analogously, in spatial statistics, a Q-Q plot of the (smoothed) residuals from a fitted point process model is a useful way to check the interpoint interaction part of the model (Baddeley et al, 2005). The systematic part of the model (spatial trend, covariate effects, etc) is assessed using other plots made by diagnose.ppm.

The argument fit represents the fitted point process model. It must be an object of class "ppm" (typically produced by the maximum pseudolikelihood fitting algorithm ppm). Residuals will be

computed for this fitted model using residuals.ppm, and the residuals will be kernel-smoothed to produce a "residual field". The values of this residual field will provide the "data" quantiles for the Q-Q plot.

The argument expr is not usually specified. It provides a way to modify the "theoretical" or "reference" quantiles for the Q-Q plot.

In normal usage we set expr=NULL. The default is to generate nsim simulated realisations of the fitted model fit, re-fit this model to each of the simulated patterns, evaluate the residuals from these fitted models, and use the kernel-smoothed residual field from these fitted models as a sample from the reference distribution for the Q-Q plot.

In advanced use, expr may be an expression. It will be re-evaluated nsim times, and should include random computations so that the results are not identical each time. The result of evaluating expr should be either a point pattern (object of class "ppp") or a fitted point process model (object of class "ppm"). If the value is a point pattern, then the original fitted model fit will be fitted to this new point pattern using update.ppm, to yield another fitted model. Smoothed residuals obtained from these nsim fitted models will yield the "theoretical" quantiles for the Q-Q plot.

Alternatively expr can be a list of point patterns, or an envelope object that contains a list of point patterns (typically generated by calling envelope with savepatterns=TRUE). These point patterns will be used as the simulated patterns.

Simulation is performed (if expr=NULL) using the Metropolis-Hastings algorithm rmh. Each simulated realisation is the result of running the Metropolis-Hastings algorithm from an independent random starting state each time. The iterative and termination behaviour of the Metropolis-Hastings algorithm are governed by the argument control. See rmhcontrol for information about this argument. As a shortcut, the argument nrep determines the number of Metropolis-Hastings iterations used to generate each simulated realisation, if control is absent.

By default, simulations are generated in an expanded window. Use the argument control to change this, as explained in the section on *Warning messages*.

The argument type selects the type of residual or weight that will be computed. For options, see diagnose.ppm.

The argument style determines the type of Q-Q plot. It is highly recommended to use the default, style="mean".

style="classical" The quantiles of the residual field for the data (on the y axis) are plotted against the quantiles of the **pooled** simulations (on the x axis). This plot is biased, and therefore difficult to interpret, because of strong autocorrelations in the residual field and the large differences in sample size.

style="mean" The order statistics of the residual field for the data are plotted against the sample means, over the nsim simulations, of the corresponding order statistics of the residual field for the simulated datasets. Dotted lines show the 2.5 and 97.5 percentiles, over the nsim simulations, of each order statistic.

The argument fast is a simple way to control the accuracy and speed of computation. If fast=FALSE, the residual field is computed on a fine grid of pixels (by default 100 by 100 pixels, see below) and the Q-Q plot is based on the complete set of order statistics (usually 10,000 quantiles). If fast=TRUE, the residual field is computed on a coarse grid (at most 40 by 40 pixels) and the Q-Q plot is based on the *percentiles* only. This is about 7 times faster. It is recommended to use fast=TRUE for interactive data analysis and fast=FALSE for definitive plots for publication.

The argument dimyx gives full control over the resolution of the pixel grid used to calculate the smoothed residuals. Its interpretation is the same as the argument dimyx to the function as.mask. Note that dimyx[1] is the number of pixels in the y direction, and dimyx[2] is the number in the x direction. If dimyx is not present, then the default pixel grid dimensions are controlled by spatstat.options("npixel").

Since the computation is so time-consuming, qqplot.ppm returns a list containing all the data necessary to re-display the Q-Q plot. It is advisable to assign the result of qqplot.ppm to something (or use .Last.value if you forgot to.) The return value is an object of class "qqppm". There are methods for plot.qqppm and print.qqppm. See the Examples.

The argument saveall is usually set to FALSE. If saveall=TRUE, then the intermediate results of calculation for each simulated realisation are saved and returned. The return value includes a 3-dimensional array sim containing the smoothed residual field images for each of the nsim realisations. When saveall=TRUE, the return value is an object of very large size, and should not be saved on disk.

Errors may occur during the simulation process, because random data are generated. For example:

- one of the simulated patterns may be empty.
- one of the simulated patterns may cause an error in the code that fits the point process model.
- the user-supplied argument expr may have a bug.

Empty point patterns do not cause a problem for the code, but they are reported. Other problems that would lead to a crash are trapped; the offending simulated data are discarded, and the simulation is retried. The argument maxerr determines the maximum number of times that such errors will be tolerated (mainly as a safeguard against an infinite loop).

Value

An object of class "qqppm" containing the information needed to reproduce the Q-Q plot. Entries x and y are numeric vectors containing quantiles of the simulations and of the data, respectively.

Side Effects

Produces a Q-Q plot if plot.it is TRUE.

Warning messages

A warning message will be issued if any of the simulations trapped an error (a potential crash).

A warning message will be issued if all, or many, of the simulated point patterns are empty. This usually indicates a problem with the simulation procedure.

The default behaviour of qqplot.ppm is to simulate patterns on an expanded window (specified through the argument control) in order to avoid edge effects. The model's trend is extrapolated over this expanded window. If the trend is strongly inhomogeneous, the extrapolated trend may have very large (or even infinite) values. This can cause the simulation algorithm to produce empty patterns.

The only way to suppress this problem entirely is to prohibit the expansion of the window, by setting the control argument to something like control=list(nrep=1e6, expand=1). Here expand=1 means there will be no expansion. See rmhcontrol for more information about the argument control.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

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.

Stoyan, D. and Grabarnik, P. (1991) Second-order characteristics for stochastic structures connected with Gibbs point processes. *Mathematische Nachrichten*, 151:95–100.

See Also

diagnose.ppm, lurking, residuals.ppm, eem, ppm.object, ppm, rmh, rmhcontrol

Examples

```
data(cells)

fit <- ppm(cells, ~1, Poisson())
diagnose.ppm(fit)  # no suggestion of departure from stationarity
if(interactive()) {
   qqplot.ppm(fit, 80)  # strong evidence of non-Poisson interaction
 diagnose.ppm(fit, type="pearson")
 qqplot.ppm(fit, type="pearson")
}


# capture the plot coordinates
# mypreciousdata <- qqplot.ppm(fit, type="pearson")
# mypreciousdata <- qqplot.ppm(fit, 4, type="pearson")
# plot(mypreciousdata)

##     use the idiom .Last.value if you forgot to assign them
mypreciousdata <- .Last.value

#########################################################
# Q-Q plots based on fixed n
# The above QQ plots used simulations from the (fitted) Poisson process.
# But I want to simulate conditional on n, instead of Poisson
# Do this by setting rmhcontrol(p=1)
fixit <- list(p=1)
if(interactive()) {qqplot.ppm(fit, 100, control=fixit)}


#########################################################
# Inhomogeneous Poisson data
X <- rpoispp(function(x,y){1000 * exp(-3*x)}, 1000)
plot(X)
# Inhomogeneous Poisson model
fit <- ppm(X, ~x, Poisson())
```

```
if(interactive()) {qqplot.ppm(fit, 100)}

# conclusion: fitted inhomogeneous Poisson model looks OK

#####################################################
# Advanced use of 'expr' argument
#
# set the initial conditions in Metropolis-Hastings algorithm
#
expr <- expression(rmh(fit, start=list(n.start=42), verbose=FALSE))
if(interactive()) qqplot.ppm(fit, 100, expr)
```

---

quad.ppm *Extract Quadrature Scheme Used to Fit a Point Process Model*

Description

Given a fitted point process model, this function extracts the quadrature scheme used to fit the model.

Usage

```
quad.ppm(object, drop=FALSE, clip=FALSE)
```

Arguments

object	fitted point process model (an object of class "ppm" or "kppm" or "lppm").
drop	Logical value determining whether to delete quadrature points that were not used to fit the model.
clip	Logical value determining whether to erode the window, if object was fitted using the border correction. See Details.

Details

An object of class "ppm" represents a point process model that has been fitted to data. It is typically produced by the model-fitting algorithm ppm.

The maximum pseudolikelihood algorithm in ppm approximates the pseudolikelihood integral by a sum over a finite set of quadrature points, which is constructed by augmenting the original data point pattern by a set of "dummy" points. The fitted model object returned by ppm contains complete information about this quadrature scheme. See ppm or ppm.object for further information.

This function quad.ppm extracts the quadrature scheme. A typical use of this function would be to inspect the quadrature scheme (points and weights) to gauge the accuracy of the approximation to the exact pseudolikelihood.

Some quadrature points may not have been used in fitting the model. This happens if the border correction is used, and in other cases (e.g. when the value of a covariate is NA at these points). The

argument drop specifies whether these unused quadrature points shall be deleted (drop=TRUE) or retained (drop=FALSE) in the return value.

The quadrature scheme has a *window*, which by default is set to equal the window of the original data. However this window may be larger than the actual domain of integration of the pseudolikelihood or composite likelihood that was used to fit the model. If clip=TRUE then the window of the quadrature scheme is set to the actual domain of integration. This option only has an effect when the model was fitted using the border correction; then the window is obtained by eroding the original data window by the border correction distance.

See ppm.object for a list of all operations that can be performed on objects of class "ppm". See quad.object for a list of all operations that can be performed on objects of class "quad".

This function can also be applied to objects of class "kppm" and "lppm".

Value

A quadrature scheme (object of class "quad").

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppm.object, quad.object, ppm

Examples

```
fit <- ppm(cells ~1, Strauss(r=0.1))
Q <- quad.ppm(fit)
# plot(Q)
npoints(Q$data)
npoints(Q$dummy)
```

---

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 'ppm'
quadrat.test(X, nx=5, ny=nx,
                          alternative=c("two.sided", "regular", "clustered"),
                           method=c("Chisq", "MonteCarlo"),
                           conditional=TRUE, CR=1, df.est=NULL,
                           ...,
                           xbreaks=NULL, ybreaks=NULL, tess=NULL,
                           nsim=1999)

## S3 method for class 'slrm'
quadrat.test(X, nx=5, ny=nx,
                           alternative=c("two.sided", "regular", "clustered"),
                           method=c("Chisq", "MonteCarlo"),
                           conditional=TRUE, CR=1, 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 <r.turner@auckland.ac.nz>

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.
# quadrat.test(swedishpines, method="M", nsim=4999)
#  quadrat.test(swedishpines, method="M", nsim=4999, conditional=FALSE)



# quadrat counts
qS <- quadratcount(simdat, 4, 3)
quadrat.test(qS)

# fitted model: inhomogeneous Poisson
fitx <- ppm(simdat ~ x)
quadrat.test(fitx)

# an equivalent test (results differ due to discretisation effects):
quadrat.test(simdat, lambda=predict(fitx), df.est=length(coef(fitx)))

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.mppm` *Chi-Squared Test for Multiple Point Process Model Based on Quadrat Counts*

Description

Performs a chi-squared goodness-of-fit test of a Poisson point process model fitted to multiple point patterns.

Usage

```
  ## S3 method for class 'mppm'
quadrat.test(X, ...)
```

Arguments

`X`	An object of class `"mppm"` representing a point process model fitted to multiple point patterns. It should be a Poisson model.
`...`	Arguments passed to `quadrat.test.ppm` which determine the size of the quadrats.

Details

This function performs a χ^2 test of goodness-of-fit for a Poisson point process model, based on quadrat counts. It can also be used to perform a test of Complete Spatial Randomness for a list of point patterns.

The function `quadrat.test` is generic, with methods for point patterns (class `"ppp"`), point process models (class `"ppm"`) and multiple point process models (class `"mppm"`).

For this function, the argument `X` should be a multiple point process model (object of class `"mppm"`) obtained by fitting a point process model to a list of point patterns using the function `mppm`.

To perform the test, the data point patterns are extracted from `X`. For each point pattern

- the window of observation is divided into rectangular tiles, and the number of data points in each tile is counted, as described in `quadratcount`.
- The expected number of points in each quadrat is calculated, as determined by the fitted model.

Then we perform a single χ^2 test of goodness-of-fit based on these observed and expected counts.

Value

An object of class "htest". Printing the object gives comprehensible output about the outcome of the test. The p-value of the test is stored in the component p.value.

The return value also belongs to the special class "quadrat.test". Plotting the object will display, for each window, the position of the quadrats, annotated by their observed and expected counts and the Pearson residuals. See the examples.

The return value also has an attribute "components" which is a list containing the results of χ^2 tests of goodness-of-fit for each individual point pattern.

Testing Complete Spatial Randomness

If the intention is to test Complete Spatial Randomness (CSR) there are two options:

- CSR with the same intensity of points in each point pattern;
- CSR with a different, unrelated intensity of points in each point pattern.

In the first case, suppose P is a list of point patterns we want to test. Then fit the multiple model fit1 <- mppm(P, ~1) which signifies a Poisson point process model with a constant intensity. Then apply quadrat.test(fit1).

In the second case, fit the model codefit2 <- mppm(P, ~id) which signifies a Poisson point process with a different constant intensity for each point pattern. Then apply quadrat.test(fit2).

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Ida-Maria Sintorn and Leanne Bischoff. Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

References

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

See Also

mppm, quadrat.test

Examples

```
H <- hyperframe(X=waterstriders)
# Poisson with constant intensity for all patterns
fit1 <- mppm(X~1, H)
quadrat.test(fit1, nx=2)

# uniform Poisson with different intensity for each pattern
fit2 <- mppm(X ~ id, H)
quadrat.test(fit2, nx=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 <r.turner@auckland.ac.nz>

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

```
data(humberside)
qH <- quadrat.test(split(humberside), 2, 3)
plot(qH)
qH
```

---

quantile.density *Quantiles of a Density Estimate*

Description

Given a kernel estimate of a probability density, compute quantiles.

Usage

```
## S3 method for class 'density'
quantile(x, probs = seq(0, 1, 0.25), names = TRUE,
        ..., warn = TRUE)
```

Arguments

x	Object of class "density" computed by a method for density
probs	Numeric vector of probabilities for which the quantiles are required.
names	Logical value indicating whether to attach names (based on probs) to the result.
...	Ignored.
warn	Logical value indicating whether to issue a warning if the density estimate x had to be renormalised because it was computed in a restricted interval.

Details

This function calculates quantiles of the probability distribution whose probability density has been estimated and stored in the object x. The object x must belong to the class "density", and would typically have been obtained from a call to the function density.

The probability density is first normalised so that the total probability is equal to 1. A warning is issued if the density estimate was restricted to an interval (i.e. if x was created by a call to density which included either of the arguments from and to).

Next, the density estimate is numerically integrated to obtain an estimate of the cumulative distribution function $F(x)$. Then for each desired probability p, the algorithm finds the corresponding quantile q.

The quantile q corresponding to probability p satisfies $F(q) = p$ up to the resolution of the grid of values contained in x. The quantile is computed from the right, that is, q is the smallest available value of x such that $F(x) \geq p$.

Value

A numeric vector containing the quantiles.

Author(s)

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

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

quantile, quantile.ewcdf, quantile.im, CDF.

Examples

```
dd <- density(runif(10))
quantile(dd)
```

---

ranef.mppm *Extract Random Effects from Point Process Model*

Description

Given a point process model fitted to a list of point patterns, extract the fixed effects of the model. A method for ranef.

Usage

```
 ## S3 method for class 'mppm'
ranef(object, ...)
```

Arguments

object	A fitted point process model (an object of class "mppm").
...	Ignored.

Details

This is a method for the generic function ranef.

The argument object must be a fitted point process model (object of class "mppm") produced by the fitting algorithm mppm). This represents a point process model that has been fitted to a list of several point pattern datasets. See mppm for information.

This function extracts the coefficients of the random effects of the model.

Value

A data frame, or list of data frames, as described in the help for ranef.lme.

Author(s)

Adrian Baddeley, Ida-Maria Sintorn and Leanne Bischoff. Implemented in **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

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

See Also

fixef.mppm, coef.mppm

Examples

```
H <- hyperframe(Y = waterstriders)
# Tweak data to exaggerate differences
H$Y[[1]] <- rthin(H$Y[[1]], 0.3)

m1 <- mppm(Y ~ id,  data=H, Strauss(7))
ranef(m1)
m2 <- mppm(Y ~ 1,  random=~1|id, data=H, Strauss(7))
ranef(m2)
```

---

range.fv *Range of Function Values*

Description

Compute the range, maximum, or minimum of the function values in a summary function.

Usage

```
  ## S3 method for class 'fv'
range(..., na.rm = TRUE, finite = na.rm)

  ## S3 method for class 'fv'
max(..., na.rm = TRUE, finite = na.rm)

  ## S3 method for class 'fv'
min(..., na.rm = TRUE, finite = na.rm)
```

Arguments

...	One or more function value tables (objects of class "fv" representing summary functions) or other data.
na.rm	Logical. Whether to ignore NA values.
finite	Logical. Whether to ignore values that are infinite, NaN or NA.

Details

These are methods for the generic range, max and min. They compute the range, maximum, and minimum of the *function* values that would be plotted on the y axis by default.

For more complicated calculations, use with.fv.

Value

Numeric vector of length 2.

Author(s)

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

, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

with.fv

Examples

```
G <- Gest(cells)
range(G)
max(G)
min(G)
```

---

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 <r.turner@auckland.ac.nz>.

See Also

compatible, pool

---

rdpp *Simulation of a Determinantal Point Process*

Description

Generates simulated realisations from a determinantal point process.

Usage

```
rdpp(eig, index, basis = "fourierbasis",
     window = boxx(rep(list(0:1), ncol(index))),
     reject_max = 10000, progress = 0, debug = FALSE, ...)
```

Arguments

eig	vector of values between 0 and 1 specifying the non-zero eigenvalues for the process.
index	data.frame or matrix (or something acceptable to as.matrix) specifying indices of the basis functions.
basis	character string giving the name of the basis.
window	window (of class "owin", "box3" or "boxx") giving the domain of the point process.
reject_max	integer giving the maximal number of trials for rejection sampling.
progress	integer giving the interval for making a progress report. The value zero turns reporting off.
debug	logical value indicating whether debug informationb should be outputted.
...	Ignored.

Value

A point pattern (object of class "ppp").

Author(s)

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

Examples

```
index <- expand.grid(-2:2,-2:2)
eig <- exp(-rowSums(index^2))
X <- rdpp(eig, index)
X
## To simulate a det. projection p. p. with the given indices set eig=1:
XX <- rdpp(1, index)
XX
```

---

reach *Interaction Distance of a Point Process Model*

Description

Computes the interaction distance of a point process model.

Usage

```
  ## S3 method for class 'ppm'
reach(x, ..., epsilon=0)

  ## S3 method for class 'interact'
reach(x, ...)

  ## S3 method for class 'fii'
reach(x, ..., epsilon)
```

Arguments

x	Either a fitted point process model (object of class "ppm"), an interpoint interaction (object of class "interact"), a fitted interpoint interaction (object of class "fii") or a point process model for simulation (object of class "rmhmodel").
epsilon	Numerical threshold below which interaction is treated as zero. See details.
...	Other arguments are ignored.

Details

The function reach computes the 'interaction distance' or 'interaction range' of a point process model.

The definition of the interaction distance depends on the type of point process model. This help page explains the interaction distance for a Gibbs point process. For other kinds of models, see reach.kppm and reach.dppm.

For a Gibbs point process model, the interaction distance is the shortest distance D such that any two points in the process which are separated by a distance greater than D do not interact with each other.

For example, the interaction range of a Strauss process (see Strauss or rStrauss) with parameters β, γ, r is equal to r, unless $\gamma = 1$ in which case the model is Poisson and the interaction range is 0. The interaction range of a Poisson process is zero. The interaction range of the Ord threshold process (see OrdThresh) is infinite, since two points *may* interact at any distance apart.

The function reach is generic, with methods for the case where x is

- a fitted point process model (object of class "ppm", usually obtained from the model-fitting function ppm);
- an interpoint interaction structure (object of class "interact") created by one of the functions Poisson, Strauss, StraussHard, MultiStrauss, MultiStraussHard, Softcore, DiggleGratton, Pairwise, PairPiece, Geyer, LennardJones, Saturated, OrdThresh or Ord;
- a fitted interpoint interaction (object of class "fii") extracted from a fitted point process model by the command fitin;
- a point process model for simulation (object of class "rmhmodel"), usually obtained from rmhmodel.

When x is an "interact" object, reach(x) returns the maximum possible interaction range for any point process model with interaction structure given by x. For example, reach(Strauss(0.2)) returns 0.2.

When x is a "ppm" object, reach(x) returns the interaction range for the point process model represented by x. For example, a fitted Strauss process model with parameters beta,gamma,r will return either 0 or r, depending on whether the fitted interaction parameter gamma is equal or not equal to 1.

For some point process models, such as the soft core process (see Softcore), the interaction distance is infinite, because the interaction terms are positive for all pairs of points. A practical solution is to compute the distance at which the interaction contribution from a pair of points falls below a threshold epsilon, on the scale of the log conditional intensity. This is done by setting the argument epsilon to a positive value.

Value

The interaction distance, or NA if this cannot be computed from the information given.

Other types of models

Methods for reach are also defined for point process models of class "kppm" and "dppm". Their technical definition is different from this one. See reach.kppm and reach.dppm.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppm, Poisson, Strauss, StraussHard, MultiStrauss, MultiStraussHard, Softcore, DiggleGratton, Pairwise, PairPiece, Geyer, LennardJones, Saturated, OrdThresh, Ord.

reach.rmhmodel

See reach.kppm and reach.dppm for other types of point process models.

Examples

```
reach(Poisson())
# returns 0

reach(Strauss(r=7))
# returns 7
fit <- ppm(swedishpines ~ 1, Strauss(r=7))
reach(fit)
# returns 7

reach(OrdThresh(42))
# returns Inf

reach(MultiStrauss(matrix(c(1,3,3,1),2,2)))
# returns 3
```

reach.dppm *Range of Interaction for a Determinantal Point Process Model*

Description

Returns the range of interaction for a determinantal point process model.

Usage

```
  ## S3 method for class 'dppm'
reach(x, ...)

  ## S3 method for class 'detpointprocfamily'
reach(x, ...)
```

Arguments

x	Model of class "detpointprocfamily" or "dppm".
...	Additional arguments passed to the range function of the given model.

Details

The range of interaction for a determinantal point process model may defined as the smallest number R such that $g(r) = 1$ for all $r \geq R$, where g is the pair correlation function. For many models the range is infinite, but one may instead use a value where the pair correlation function is sufficiently close to 1. For example in the Matérn model this defaults to finding R such that $g(R) = 0.99$.

Value

Numeric

Author(s)

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

Examples

```
reach(dppMatern(lambda=100, alpha=.01, nu=1, d=2))
```

reach.kppm *Range of Interaction for a Cox or Cluster Point Process Model*

Description

Returns the range of interaction for a Cox or cluster point process model.

Usage

```
  ## S3 method for class 'kppm'
reach(x, ..., epsilon)
```

Arguments

x	Fitted point process model of class "kppm".
epsilon	Optional numerical value. Differences smaller than epsilon are treated as zero.
...	Additional arguments passed to the range function of the given model.

Details

The range of interaction for a fitted point process model of class "kppm" may defined as the smallest number R such that $g(r) = 1$ for all $r \geq R$, where g is the pair correlation function.

For many models the range is infinite, but one may instead use a value where the pair correlation function is sufficiently close to 1. The argument epsilon specifies the tolerance; there is a sensible default.

Value

Numeric

Author(s)

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

Examples

```
fit <- kppm(redwood ~ 1)
reach(fit)
```

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))
```

`reduced.sample` *Reduced Sample Estimator using Histogram Data*

Description

Compute the Reduced Sample estimator of a survival time distribution function, from histogram data

Usage

```
reduced.sample(nco, cen, ncc, show=FALSE, uppercen=0)
```

Arguments

`nco`	vector of counts giving the histogram of uncensored observations (those survival times that are less than or equal to the censoring time)
`cen`	vector of counts giving the histogram of censoring times
`ncc`	vector of counts giving the histogram of censoring times for the uncensored observations only
`uppercen`	number of censoring times greater than the rightmost histogram breakpoint (if there are any)
`show`	Logical value controlling the amount of detail returned by the function value (see below)

Details

This function is needed mainly for internal use in **spatstat**, but may be useful in other applications where you want to form the reduced sample estimator from a huge dataset.

Suppose T_i are the survival times of individuals $i = 1, \ldots, M$ with unknown distribution function $F(t)$ which we wish to estimate. Suppose these times are right-censored by random censoring times C_i. Thus the observations consist of right-censored survival times $\tilde{T}_i = \min(T_i, C_i)$ and non-censoring indicators $D_i = 1\{T_i \le C_i\}$ for each i.

If the number of observations M is large, it is efficient to use histograms. Form the histogram `cen` of all censoring times C_i. That is, `obs[k]` counts the number of values C_i in the interval `(breaks[k],breaks[k+1]]` for $k > 1$ and `[breaks[1],breaks[2]]` for $k = 1$. Also form the histogram `nco` of all uncensored times, i.e. those $\tilde{T}_i$ such that $D_i = 1$, and the histogram of all censoring times for which the survival time is uncensored, i.e. those C_i such that $D_i = 1$. These three histograms are the arguments passed to `kaplan.meier`.

The return value rs is the reduced-sample estimator of the distribution function $F(t)$. Specifically, rs[k] is the reduced sample estimate of F(breaks[k+1]). The value is exact, i.e. the use of histograms does not introduce any approximation error.

Note that, for the results to be valid, either the histogram breaks must span the censoring times, or the number of censoring times that do not fall in a histogram cell must have been counted in uppercen.

Value

If show = FALSE, a numeric vector giving the values of the reduced sample estimator. If show=TRUE, a list with three components which are vectors of equal length,

rs	Reduced sample estimate of the survival time c.d.f. $F(t)$
numerator	numerator of the reduced sample estimator
denominator	denominator of the reduced sample estimator

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

kaplan.meier, km.rs

---

reload.or.compute *Compute Unless Previously Saved*

Description

If the designated file does not yet exist, evaluate the expression and save the results in the file. If the file already exists, re-load the results from the file.

Usage

```
reload.or.compute(filename, expr, objects = NULL,
                  context = parent.frame(),
                  destination = parent.frame(),
                  force=FALSE, verbose=TRUE)
```

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.
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, then expr will be evaluated and the results will be saved in filename. The optional argument objects specifies which results should be saved to the file: the default is to save all objects that were created by evaluating the expression.

If the file called filename already exists, then it will be loaded. The optional argument objects specifies the names of objects that should be present in the file; a warning is issued if any of them are missing.

The resulting objects can be assigned into any desired destination. 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 <r.turner@auckland.ac.nz>

Examples

```
## Not run:
if(FALSE) {
 reload.or.compute("mydata.rda", {
    x <- very.long.computation()
    y <- 42
```

```
      })
  }

## End(Not run)
```

`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 <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

`relrisk.ppp`, `relrisk.ppm`.

relrisk.ppm *Parametric Estimate of Spatially-Varying Relative Risk*

Description

Given a point process model fitted to a multitype point pattern, this function computes the fitted spatially-varying probability of each type of point, or the ratios of such probabilities, according to the fitted model. Optionally the standard errors of the estimates are also computed.

Usage

```
## S3 method for class 'ppm'
relrisk(X, ...,
                    at = c("pixels", "points"),
                    relative = FALSE, se = FALSE,
                    casecontrol = TRUE, control = 1, case,
                    ngrid = NULL, window = NULL)
```

Arguments

X	A fitted point process model (object of class "ppm").
...	Ignored.
at	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").
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.
se	Logical value indicating whether to compute standard errors as well.
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.
ngrid	Optional. Dimensions of a rectangular grid of locations inside window where the predictions should be computed. An integer, or an integer vector of length 2, specifying the number of grid points in the y and x directions. (Applies only when at="pixels".)
window	Optional. A window (object of class "owin") *delimiting* the locations where predictions should be computed. Defaults to the window of the original data used to fit the model object. (Applies only when at="pixels".)

Details

The command relrisk is generic and can be used to estimate relative risk in different ways.

This function relrisk.ppm is the method for fitted point process models (class "ppm"). It computes *parametric* estimates of relative risk, using the fitted model.

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.

Probabilities and risks are computed from the fitted intensity of the model, using predict.ppm. If se=TRUE then standard errors will also be computed, based on asymptotic theory, using vcov.ppm.

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.

Author(s)

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

See Also

There is another method relrisk.ppp for point pattern datasets which computes *nonparametric* estimates of relative risk by kernel smoothing.

See also relrisk, relrisk.ppp, ppm

Examples

```
fit <- ppm(chorley ~ marks * (x+y))
rr <- relrisk(fit, relative=TRUE, control="lung", se=TRUE)
plot(rr$estimate)
plot(rr$SE)
rrX <- relrisk(fit, at="points", relative=TRUE, control="lung")
```

---

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,
           adjust=1, edge=TRUE, diggle=FALSE, se=FALSE,
           casecontrol=TRUE, control=1, case)
```

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.
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.
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.

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.

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 (Diggle, 2003). 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.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>.

References

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

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")
}
```

---

repul.dppm *Repulsiveness Index of a Determinantal Point Process Model*

Description

Computes a measure of the degree of repulsion between points in a determinantal point process model.

Usage

```
repul(model, ...)

## S3 method for class 'dppm'
repul(model, ...)
```

Arguments

model	A fitted point process model of determinantal type (object of class "dppm").
...	Ignored.

Details

The repulsiveness index μ of a determinantal point process model was defined by Lavancier, Møller and Rubak (2015) as

$$\mu = \lambda \int (1 - g(x))\, dx$$

where λ is the intensity of the model and $g(x)$ is the pair correlation function, and the integral is taken over all two-dimensional vectors x.

Values of μ are dimensionless. Larger values of μ indicate stronger repulsion between points.

If the model is stationary, the result is a single number.

If the model is not stationary, the result is a pixel image (obtained by multiplying the spatially-varying intensity by the integral defined above).

Value

A numeric value or a pixel image.

Author(s)

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

References

Lavancier, F., Møller, J. and Rubak, E. (2015), Determinantal point process models and statistical inference. *Journal of Royal Statistical Society: Series B (Statistical Methodology)*, **77**, 853–877.

See Also

dppm

Examples

```
jpines <- residualspaper$Fig1

fit <- dppm(jpines ~ 1, dppGauss)
repul(fit)
```

`residuals.dppm` *Residuals for Fitted Determinantal Point Process Model*

Description

Given a determinantal point process model fitted to a point pattern, compute residuals.

Usage

```
  ## S3 method for class 'dppm'
residuals(object, ...)
```

Arguments

`object`	The fitted determinatal point process model (an object of class `"dppm"`) for which residuals should be calculated.
`...`	Arguments passed to `residuals.ppm`.

Details

This function extracts the intensity component of the model using `as.ppm` and then applies `residuals.ppm` to compute the residuals.

Use `plot.msr` to plot the residuals directly.

Value

An object of class `"msr"` representing a signed measure or vector-valued measure (see `msr`). This object can be plotted.

Author(s)

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

See Also

`msr`, `dppm`

Examples

```
fit <- dppm(swedishpines ~ x, dppGauss, method="c")
rr <- residuals(fit)
```

`residuals.kppm` *Residuals for Fitted Cox or Cluster Point Process Model*

Description

Given a Cox or cluster point process model fitted to a point pattern, compute residuals.

Usage

```
  ## S3 method for class 'kppm'
residuals(object, ...)
```

Arguments

`object`	The fitted point process model (an object of class `"kppm"`) for which residuals should be calculated.
`...`	Arguments passed to `residuals.ppm`.

Details

This function extracts the intensity component of the model using `as.ppm` and then applies `residuals.ppm` to compute the residuals.

Use `plot.msr` to plot the residuals directly.

Value

An object of class `"msr"` representing a signed measure or vector-valued measure (see `msr`). This object can be plotted.

Author(s)

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

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

`msr`, `kppm`

Examples

```
fit <- kppm(redwood ~ x, "Thomas")
rr <- residuals(fit)
```

`residuals.mppm` *Residuals for Point Process Model Fitted to Multiple Point Patterns*

Description

Given a point process model fitted to multiple point patterns, compute residuals for each pattern.

Usage

```
## S3 method for class 'mppm'
residuals(object, type = "raw", ...,
                              fittedvalues = fitted.mppm(object))
```

Arguments

`object`	Fitted point process model (object of class `"mppm"`).
`...`	Ignored.
`type`	Type of residuals: either `"raw"`, `"pearson"` or `"inverse"`. Partially matched.
`fittedvalues`	Advanced use only. Fitted values of the model to be used in the calculation.

Details

Baddeley et al (2005) defined residuals for the fit of a point process model to spatial point pattern data. For an explanation of these residuals, see the help file for `residuals.ppm`.

This function computes the residuals for a point process model fitted to *multiple* point patterns. The `object` should be an object of class `"mppm"` obtained from `mppm`.

The return value is a list. The number of entries in the list equals the number of point patterns in the original data. Each entry in the list has the same format as the output of `residuals.ppm`. That is, each entry in the list is a signed measure (object of class `"msr"`) giving the residual measure for the corresponding point pattern.

Value

A list of signed measures (objects of class `"msr"`) giving the residual measure for each of the original point patterns. See Details.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Ida-Maria Sintorn and Leanne Bischoff. Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

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.

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

See Also

mppm, residuals.mppm

Examples

```
fit <- mppm(Bugs ~ x, hyperframe(Bugs=waterstriders))
r <- residuals(fit)
# compute total residual for each point pattern
rtot <- sapply(r, integral.msr)
# standardise the total residuals
areas <- sapply(windows.mppm(fit), area.owin)
rtot/sqrt(areas)
```

---

residuals.ppm *Residuals for Fitted Point Process Model*

Description

Given a point process model fitted to a point pattern, compute residuals.

Usage

```
  ## S3 method for class 'ppm'
residuals(object, type="raw", ...,
                check=TRUE, drop=FALSE,
                fittedvalues=NULL,
                new.coef=NULL, dropcoef=FALSE,
                quad=NULL)
```

Arguments

object	The fitted point process model (an object of class "ppm") for which residuals should be calculated.
type	String indicating the type of residuals to be calculated. Current options are "raw", "inverse", "pearson" and "score". A partial match is adequate.
...	Ignored.
check	Logical value indicating whether to check the internal format of object. If there is any possibility that this object has been restored from a dump file, or has otherwise lost track of the environment where it was originally computed, set check=TRUE.

drop	Logical value determining whether to delete quadrature points that were not used to fit the model. See quad.ppm for explanation.
fittedvalues	Vector of fitted values for the conditional intensity at the quadrature points, from which the residuals will be computed. For expert use only.
new.coef	Optional. Numeric vector of coefficients for the model, replacing coef(object). See the section on Modified Residuals below.
dropcoef	Internal use only.
quad	Optional. Data specifying how to re-fit the model. A list of arguments passed to quadscheme. See the section on Modified Residuals below.

Details

This function computes several kinds of residuals for the fit of a point process model to a spatial point pattern dataset (Baddeley et al, 2005). Use plot.msr to plot the residuals directly, or diagnose.ppm to produce diagnostic plots based on these residuals.

The argument object must be a fitted point process model (object of class "ppm"). Such objects are produced by the maximum pseudolikelihood fitting algorithm ppm. This fitted model object contains complete information about the original data pattern.

Residuals are attached both to the data points and to some other points in the window of observation (namely, to the dummy points of the quadrature scheme used to fit the model). If the fitted model is correct, then the sum of the residuals over all (data and dummy) points in a spatial region B has mean zero. For further explanation, see Baddeley et al (2005).

The type of residual is chosen by the argument type. Current options are

"raw": the raw residuals

$$r_j = z_j - w_j \lambda_j$$

at the quadrature points u_j, where z_j is the indicator equal to 1 if u_j is a data point and 0 if u_j is a dummy point; w_j is the quadrature weight attached to u_j; and

$$\lambda_j = \hat{\lambda}(u_j, x)$$

is the conditional intensity of the fitted model at u_j. These are the spatial analogue of the martingale residuals of a one-dimensional counting process.

"inverse": the 'inverse-lambda' residuals (Baddeley et al, 2005)

$$r_j^{(I)} = \frac{r_j}{\lambda_j} = \frac{z_j}{\lambda_j} - w_j$$

obtained by dividing the raw residuals by the fitted conditional intensity. These are a counterpart of the exponential energy marks (see eem).

"pearson": the Pearson residuals (Baddeley et al, 2005)

$$r_j^{(P)} = \frac{r_j}{\sqrt{\lambda_j}} = \frac{z_j}{\sqrt{\lambda_j}} - w_j \sqrt{\lambda_j}$$

obtained by dividing the raw residuals by the square root of the fitted conditional intensity. The Pearson residuals are standardised, in the sense that if the model (true and fitted) is Poisson, then the sum of the Pearson residuals in a spatial region B has variance equal to the area of B.

`"score"`: the score residuals (Baddeley et al, 2005)

$$r_j = (z_j - w_j \lambda_j) x_j$$

obtained by multiplying the raw residuals r_j by the covariates x_j for quadrature point j. The score residuals always sum to zero.

The result of `residuals.ppm` is a measure (object of class `"msr"`). Use `plot.msr` to plot the residuals directly, or `diagnose.ppm` to produce diagnostic plots based on these residuals. Use `integral.msr` to compute the total residual.

By default, the window of the measure is the same as the original window of the data. If `drop=TRUE` then the window is the domain of integration of the pseudolikelihood or composite likelihood. This only matters when the model `object` was fitted using the border correction: in that case, if `drop=TRUE` the window of the residuals is the erosion of the original data window by the border correction distance `rbord`.

Value

An object of class `"msr"` representing a signed measure or vector-valued measure (see `msr`). This object can be plotted.

Modified Residuals

Sometimes we want to modify the calculation of residuals by using different values for the model parameters. This capability is provided by the arguments `new.coef` and `quad`.

If `new.coef` is given, then the residuals will be computed by taking the model parameters to be `new.coef`. This should be a numeric vector of the same length as the vector of fitted model parameters `coef(object)`.

If `new.coef` is missing and `quad` is given, then the model parameters will be determined by refitting the model using a new quadrature scheme specified by `quad`. Residuals will be computed for the original model `object` using these new parameter values.

The argument `quad` should normally be a list of arguments in `name=value` format that will be passed to `quadscheme` (together with the original data points) to determine the new quadrature scheme. It may also be a quadrature scheme (object of class `"quad"`) to which the model should be fitted, or a point pattern (object of class `"ppp"`) specifying the *dummy points* in a new quadrature scheme.

Author(s)

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

and Rolf Turner `<r.turner@auckland.ac.nz>`

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.

Baddeley, A., Møller, J. and Pakes, A.G. (2008) Properties of residuals for spatial point processes. *Annals of the Institute of Statistical Mathematics* **60**, 627–649.

See Also

msr, diagnose.ppm, ppm.object, ppm

Examples

```
fit <- ppm(cells, ~x, Strauss(r=0.15))

# Pearson residuals
rp <- residuals(fit, type="pe")
rp

# simulated data
X <- rStrauss(100,0.7,0.05)
# fit Strauss model
fit <- ppm(X, ~1, Strauss(0.05))
res.fit <- residuals(fit)

# check that total residual is 0
integral.msr(residuals(fit, drop=TRUE))

# true model parameters
truecoef <- c(log(100), log(0.7))
res.true <- residuals(fit, new.coef=truecoef)
```

---

residuals.slrm *Residuals for Fitted Spatial Logistic Regression Model*

Description

Given a spatial logistic regression model fitted to a point pattern, compute the residuals for each pixel.

Usage

```
  ## S3 method for class 'slrm'
residuals(object,
                          type=c("raw", "deviance", "pearson", "working",
                                 "response", "partial", "score"),
                          ...)
```

Arguments

object	The fitted point process model (an object of class "ppm") for which residuals should be calculated.
type	String (partially matched) indicating the type of residuals to be calculated.
...	Ignored.

Details

This function computes several kinds of residuals for the fit of a spatial logistic regression model to a spatial point pattern dataset.

The argument `object` must be a fitted spatial logistic regression model (object of class `"slrm"`). Such objects are created by the fitting algorithm `slrm`.

The residuals are computed for each pixel that was used to fit the original model. The residuals are returned as a pixel image (if the residual values are scalar), or a list of pixel images (if the residual values are vectors).

The type of residual is chosen by the argument `type`.

For a given pixel, suppose p is the fitted probability of presence of a point, and y is the presence indicator (equal to 1 if the pixel contains any data points, and equal to 0 otherwise). Then

- `type="raw"` or `type="response"` specifies the response residual
$$r = y - p$$
- `type="pearson"` is the Pearson residual
$$r_P = \frac{y-p}{\sqrt{p(1-p)}}$$
- `type="deviance"` is the deviance residual
$$r_D = (-1)^{y+1}\sqrt{-2(ylogp + (1-y)log(1-p))}$$
- `type="score"` specifies the score residuals
$$r_S = (y-p)x$$
where `x` is the vector of canonical covariate values for the pixel
- `type="working"` specifies the working residuals as defined in `residuals.glm`
- `type="partial"` specifies the partial residuals as defined in `residuals.glm`

Value

A pixel image (if the residual values are scalar), or a list of pixel images (if the residual values are vectors).

Author(s)

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

See Also

`residuals.glm`, `residuals.ppm`

Examples

```
d <- if(interactive()) 128 else 32
H <- unmark(humberside)
fit <- slrm(H ~ x + y, dimyx=d)

plot(residuals(fit))

plot(residuals(fit, type="score"))
```

`response` *Extract the Values of the Response from a Fitted Model*

Description

Given a fitted model (of any kind) extract the values of the response variable.

Usage

```
response(object)

## S3 method for class 'lm'
response(object)

## S3 method for class 'glm'
response(object)

## S3 method for class 'ppm'
response(object)

## S3 method for class 'kppm'
response(object)

## S3 method for class 'dppm'
response(object)

## S3 method for class 'slrm'
response(object)

## S3 method for class 'mppm'
response(object)
```

Arguments

`object`	A fitted model (object of class `"lm"`, `"glm"`, `"ppm"`, `"kppm"`, `"dppm"`, `"slrm"` or `"mppm"` or some other class).

Details

For fitted linear models of class "lm" and fitted generalized linear models of class "glm", the numerical values of the response variable are extracted if they are available, and otherwise NULL is returned.

For fitted point process models of class "ppm", "kppm", "dppm", "slrm" or "lppm", the original data point pattern is extracted.

For a fitted point process model of class "mppm", the list of original data point patterns is extracted.

Value

For response.lm and response.glm, a numeric vector, or NULL.

For response.ppm, response.kppm, response.dppm and response.slrm a two-dimensional spatial point pattern (class "ppp").

For response.mppm, a list of two-dimensional spatial point patterns (objects of class "ppp"). The list also belongs to classes "solist" and "ppplist".

Author(s)

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

Examples

```
fit <- ppm(cells ~ x)
response(fit)
```

---

rex *Richardson Extrapolation*

Description

Performs Richardson Extrapolation on a sequence of approximate values.

Usage

```
rex(x, r = 2, k = 1, recursive = FALSE)
```

Arguments

x	A numeric vector or matrix, whose columns are successive estimates or approximations to a vector of parameters.
r	A number greater than 1. The ratio of successive step sizes. See Details.
k	Integer. The order of convergence assumed. See Details.
recursive	Logical value indicating whether to perform one step of Richardson extrapolation (recursive=FALSE, the default) or repeat the extrapolation procedure until a best estimate is obtained (recursive=TRUE.

Details

Richardson extrapolation is a general technique for improving numerical approximations, often used in numerical integration (Brezinski and Zaglia, 1991). It can also be used to improve parameter estimates in statistical models (Baddeley and Turner, 2014).

The successive columns of x are assumed to have been obtained using approximations with step sizes $a, a/r, a/r^2, \ldots$ where a is the initial step size (which does not need to be specified).

Estimates based on a step size s are assumed to have an error of order s^k.

Thus, the default values r=2 and k=1 imply that the errors in the second column of x should be roughly $(1/r)^k = 1/2$ as large as the errors in the first column, and so on.

Value

A matrix whose columns contain a sequence of improved estimates.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>.

References

Baddeley, A. and Turner, R. (2014) Bias correction for parameter estimates of spatial point process models. *Journal of Statistical Computation and Simulation* **84**, 1621–1643. DOI: 10.1080/00949655.2012.755976

Brezinski, C. and Zaglia, M.R. (1991) *Extrapolation Methods. Theory and Practice*. North-Holland.

See Also

bc

Examples

```
# integrals of sin(x) and cos(x) from 0 to pi
# correct answers: 2, 0
est <- function(nsteps) {
  xx <- seq(0, pi, length=nsteps)
  ans <- pi * c(mean(sin(xx)), mean(cos(xx)))
  names(ans) <- c("sin", "cos")
  ans
}
X <- cbind(est(10), est(20), est(40))
X
rex(X)
rex(X, recursive=TRUE)

# fitted Gibbs point process model
fit0 <- ppm(cells ~ 1, Strauss(0.07), nd=16)
fit1 <- update(fit0, nd=32)
fit2 <- update(fit0, nd=64)
co <- cbind(coef(fit0), coef(fit1), coef(fit2))
```

```
co
rex(co, k=2, recursive=TRUE)
```

---

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

```
data(bei)
attach(bei.extra)
plot(rho2hat(bei, elev, grad))
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", "piecewise"),
       subset=NULL,
       dimyx=NULL, eps=NULL,
       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", "piecewise"),
       subset=NULL,
       dimyx=NULL, eps=NULL,
       n = 512, bw = "nrd0", adjust=1, from = NULL, to = NULL,
       bwref=bw,
       covname, confidence=0.95, positiveCI, breaks=NULL)

## S3 method for class 'ppm'
rhohat(object, covariate, ...,
       weights=NULL,
       method=c("ratio", "reweight", "transform"),
       horvitz=FALSE,
       smoother=c("kernel", "local", "decreasing", "increasing", "piecewise"),
       subset=NULL,
       dimyx=NULL, eps=NULL,
       n = 512, bw = "nrd0", adjust=1, from = NULL, to = NULL,
       bwref=bw,
       covname, confidence=0.95, positiveCI, breaks=NULL)

## S3 method for class 'slrm'
rhohat(object, covariate, ...,
       weights=NULL,
       method=c("ratio", "reweight", "transform"),
       horvitz=FALSE,
       smoother=c("kernel", "local", "decreasing", "increasing", "piecewise"),
       subset=NULL,
       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 smoothing method. See Details.
horvitz	Logical value indicating whether to use Horvitz-Thompson weights. See Details.
smoother	Character string determining the smoothing algorithm. See Details.
subset	Optional. A spatial window (object of class "owin") specifying a subset of the data, from which the estimate should be calculated.
dimyx,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) or `smoother="local"`, the nonparametric estimator is a *smoothing estimator* of $\rho(z)$, effectively a kind of density estimator (Baddeley et al, 2012). The estimated function $\rho(z)$ will be a smooth function of z. Confidence bands are also computed, assuming a Poisson point process. See the section on *Smooth estimates*.
- If `smoother="increasing"` or `smoother="decreasing"`, we use the *nonparametric maximum likelihood estimator* of $\rho(z)$ described by Sager (1982). This 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. See the section on *Monotone 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.

See Baddeley (2018) for a comparison of these estimation techniques.

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` 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.

Author(s)

Smoothing algorithm by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Ya-Mei Chang, Yong Song, and Rolf Turner <r.turner@auckland.ac.nz>.

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")

rhoM <- rhohat(X, "x", smoother="increasing")

plot(rhoM, add=TRUE, .y ~ .x, col=6)
legend("top", lty=c(3, 1), col=c(4, 6), lwd=c(2, 1),
       legend=c("true", "increasing"))



fit <- ppm(X ~x)
rr <- rhohat(fit, "y")
```

`rmh.ppm` *Simulate from a Fitted Point Process Model*

Description

Given a point process model fitted to data, generate a random simulation of the model, using the Metropolis-Hastings algorithm.

Usage

```
  ## S3 method for class 'ppm'
rmh(model, start=NULL,
                    control=default.rmhcontrol(model, w=w),
                    ...,
                    w = NULL,
                    project=TRUE,
                    nsim=1, drop=TRUE, saveinfo=TRUE,
                    verbose=TRUE, new.coef=NULL)
```

Arguments

`model`	A fitted point process model (object of class `"ppm"`, see `ppm.object`) which it is desired to simulate. This fitted model is usually the result of a call to `ppm`. See **Details** below.
`start`	Data determining the initial state of the Metropolis-Hastings algorithm. See `rmhstart` for description of these arguments. Defaults to `list(x.start=data.ppm(model))`

control	Data controlling the iterative behaviour of the Metropolis-Hastings algorithm. See rmhcontrol for description of these arguments.
...	Further arguments passed to rmhcontrol, or to rmh.default, or to covariate functions in the model.
w	Optional. Window in which the simulations should be generated. Default is the window of the original data.
project	Logical flag indicating what to do if the fitted model is invalid (in the sense that the values of the fitted coefficients do not specify a valid point process). If project=TRUE the closest valid model will be simulated; if project=FALSE an error will occur.
nsim	Number of simulated point patterns that should be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a single point pattern.
saveinfo	Logical value indicating whether to save auxiliary information.
verbose	Logical flag indicating whether to print progress reports.
new.coef	New values for the canonical parameters of the model. A numeric vector of the same length as coef(model).

Details

This function generates simulated realisations from a point process model that has been fitted to point pattern data. It is a method for the generic function rmh for the class "ppm" of fitted point process models. To simulate other kinds of point process models, see rmh or rmh.default.

The argument model describes the fitted model. It must be an object of class "ppm" (see ppm.object), and will typically be the result of a call to the point process model fitting function ppm.

The current implementation enables simulation from any fitted model involving the interactions AreaInter, DiggleGratton, DiggleGatesStibbard, Geyer, Hardcore, MultiStrauss, MultiStraussHard, PairPiece, Poisson, Strauss, StraussHard and Softcore, including nonstationary models. See the examples.

It is also possible to simulate *hybrids* of several such models. See Hybrid and the examples.

It is possible that the fitted coefficients of a point process model may be "illegal", i.e. that there may not exist a mathematically well-defined point process with the given parameter values. For example, a Strauss process with interaction parameter $\gamma > 1$ does not exist, but the model-fitting procedure used in ppm will sometimes produce values of γ greater than 1. In such cases, if project=FALSE then an error will occur, while if project=TRUE then rmh.ppm will find the nearest legal model and simulate this model instead. (The nearest legal model is obtained by projecting the vector of coefficients onto the set of valid coefficient vectors. The result is usually the Poisson process with the same fitted intensity.)

The arguments start and control are lists of parameters determining the initial state and the iterative behaviour, respectively, of the Metropolis-Hastings algorithm.

The argument start is passed directly to rmhstart. See rmhstart for details of the parameters of the initial state, and their default values.

The argument control is first passed to rmhcontrol. Then if any additional arguments ... are given, update.rmhcontrol is called to update the parameter values. See rmhcontrol for details of the iterative behaviour parameters, and default.rmhcontrol for their default values.

Note that if you specify expansion of the simulation window using the parameter expand (so that the model will be simulated on a window larger than the original data window) then the model must be capable of extrapolation to this larger window. This is usually not possible for models which depend on external covariates, because the domain of a covariate image is usually the same as the domain of the fitted model.

After extracting the relevant information from the fitted model object model, rmh.ppm invokes the default rmh algorithm rmh.default, unless the model is Poisson. If the model is Poisson then the Metropolis-Hastings algorithm is not needed, and the model is simulated directly, using one of rpoispp, rmpoispp, rpoint or rmpoint.

See rmh.default for further information about the implementation, or about the Metropolis-Hastings algorithm.

Value

A point pattern (an object of class "ppp"; see ppp.object) or a list of point patterns.

Warnings

See Warnings in rmh.default.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

simulate.ppm, rmh, rmhmodel, rmhcontrol, default.rmhcontrol, update.rmhcontrol, rmhstart, rmh.default, ppp.object, ppm,

Interactions: AreaInter, DiggleGratton, DiggleGatesStibbard, Geyer, Hardcore, Hybrid, MultiStrauss, MultiStraussHard, PairPiece, Poisson, Strauss, StraussHard, Softcore

Examples

```
live <- interactive()
op <- spatstat.options()
spatstat.options(rmh.nrep=1e5)
Nrep <- 1e5

X <- swedishpines
if(live) plot(X, main="Swedish Pines data")

# Poisson process
fit <- ppm(X, ~1, Poisson())
Xsim <- rmh(fit)
if(live) plot(Xsim, main="simulation from fitted Poisson model")

# Strauss process
fit <- ppm(X, ~1, Strauss(r=7))
Xsim <- rmh(fit)
```

```
if(live) plot(Xsim, main="simulation from fitted Strauss model")

if(live) {
  # Strauss process simulated on a larger window
  # then clipped to original window
  Xsim <- rmh(fit, control=list(nrep=Nrep, expand=1.1, periodic=TRUE))
  Xsim <- rmh(fit, nrep=Nrep, expand=2, periodic=TRUE)
}

if(live) {
  X <- rSSI(0.05, 100)
  # piecewise-constant pairwise interaction function
  fit <- ppm(X, ~1, PairPiece(seq(0.02, 0.1, by=0.01)))
  Xsim <- rmh(fit)
}

 # marked point pattern
 Y <- amacrine

if(live) {
  # marked Poisson models
  fit <- ppm(Y)
  fit <- ppm(Y,~marks)
  fit <- ppm(Y,~polynom(x,2))
  fit <- ppm(Y,~marks+polynom(x,2))
  fit <- ppm(Y,~marks*polynom(x,y,2))
  Ysim <- rmh(fit)
}

# multitype Strauss models
MS <- MultiStrauss(radii=matrix(0.07, ncol=2, nrow=2),
                   types = levels(Y$marks))
if(live) {
 fit <- ppm(Y ~marks, MS)
 Ysim <- rmh(fit)
}

fit <- ppm(Y ~ marks*polynom(x,y,2), MS)
Ysim <- rmh(fit)
if(live) plot(Ysim, main="simulation from fitted inhomogeneous Multitype Strauss")

spatstat.options(op)

if(live) {
  # Hybrid model
  fit <- ppm(redwood, ~1, Hybrid(A=Strauss(0.02), B=Geyer(0.1, 2)))
  Y <- rmh(fit)
}
```

---

rmhmodel.ppm *Interpret Fitted Model for Metropolis-Hastings Simulation.*

Description

Converts a fitted point process model into a format that can be used to simulate the model by the Metropolis-Hastings algorithm.

Usage

```
  ## S3 method for class 'ppm'
rmhmodel(model, w, ..., verbose=TRUE, project=TRUE,
                        control=rmhcontrol(),
                        new.coef=NULL)
```

Arguments

model	Fitted point process model (object of class "ppm").
w	Optional. Window in which the simulations should be generated.
...	Ignored.
verbose	Logical flag indicating whether to print progress reports while the model is being converted.
project	Logical flag indicating what to do if the fitted model does not correspond to a valid point process. See Details.
control	Parameters determining the iterative behaviour of the simulation algorithm. Passed to rmhcontrol.
new.coef	New values for the canonical parameters of the model. A numeric vector of the same length as coef(model).

Details

The generic function rmhmodel takes a description of a point process model in some format, and converts it into an object of class "rmhmodel" so that simulations of the model can be generated using the Metropolis-Hastings algorithm rmh.

This function rmhmodel.ppm is the method for the class "ppm" of fitted point process models.

The argument model should be a fitted point process model (object of class "ppm") typically obtained from the model-fitting function ppm. This will be converted into an object of class "rmhmodel".

The optional argument w specifies the window in which the pattern is to be generated. If specified, it must be in a form which can be coerced to an object of class owin by as.owin.

Not all fitted point process models obtained from ppm can be simulated. We have not yet implemented simulation code for the LennardJones and OrdThresh models.

It is also possible that a fitted point process model obtained from ppm may not correspond to a valid point process. For example a fitted model with the Strauss interpoint interaction may have any value of the interaction parameter γ; however the Strauss process is not well-defined for $\gamma > 1$ (Kelly and Ripley, 1976).

The argument project determines what to do in such cases. If project=FALSE, a fatal error will occur. If project=TRUE, the fitted model parameters will be adjusted to the nearest values which do correspond to a valid point process. For example a Strauss process with $\gamma > 1$ will be projected to a Strauss process with $\gamma = 1$, equivalent to a Poisson process.

Value

An object of class "rmhmodel", which is essentially a list of parameter values for the model.

There is a print method for this class, which prints a sensible description of the model chosen.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

References

Diggle, P. J. (2003) *Statistical Analysis of Spatial Point Patterns* (2nd ed.) Arnold, London.

Diggle, P.J. and Gratton, R.J. (1984) Monte Carlo methods of inference for implicit statistical models. *Journal of the Royal Statistical Society, series B* **46**, 193 – 212.

Geyer, C.J. (1999) Likelihood Inference for Spatial Point Processes. Chapter 3 in O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. Van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*, Chapman and Hall / CRC, Monographs on Statistics and Applied Probability, number 80. Pages 79–140.

Kelly, F.P. and Ripley, B.D. (1976) On Strauss's model for clustering. *Biometrika* **63**, 357–360.

See Also

rmhmodel, rmhmodel.list, rmhmodel.default, rmh, rmhcontrol, rmhstart, ppm, AreaInter, BadGey, DiggleGatesStibbard, DiggleGratton, Fiksel, Geyer, Hardcore, Hybrid, LennardJones, MultiStrauss, MultiStraussHard, PairPiece, Penttinen, Poisson, Softcore, Strauss, StraussHard and Triplets.

Examples

```
fit1 <- ppm(cells ~1, Strauss(0.07))
mod1 <- rmhmodel(fit1)

fit2 <- ppm(cells ~x, Geyer(0.07, 2))
mod2 <- rmhmodel(fit2)

fit3 <- ppm(cells ~x, Hardcore(0.07))
mod3 <- rmhmodel(fit3)

# Then rmh(mod1), etc
```

---

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, ..., high = TRUE)

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

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

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

Arguments

X	Point pattern (object of class "ppp" or "lpp") or fitted point process model (object of class "ppm", "kppm", "slrm" or "lppm").
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 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.

Details

This command computes Receiver Operating Characteristic curve. The area under the ROC is computed by auc.

For a point pattern X and a 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.

For a fitted point process model, the ROC shows the ability of the fitted model intensity to separate the spatial domain into areas of high and low density of points. The ROC is **not** a diagnostic for the goodness-of-fit of the model (Lobo et al, 2007).

(For spatial logistic regression models (class "slrm") replace "intensity" by "probability of presence" in the text above.)

Value

Function value table (object of class "fv") which can be plotted to show the ROC curve.

Author(s)

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

References

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

auc

Examples

```
plot(roc(swedishpines, "x"))
fit <- ppm(swedishpines ~ x+y)
plot(roc(fit))
```

---

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 <r.turner@auckland.ac.nz>

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 <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

Examples

```
if(interactive()) {
  Z <- setcov(square(1))
  plot(rotmean(Z))
  plot(rotmean(Z, result="im"))
} else {
  Z <- setcov(square(1), dimyx=32)
  f <- rotmean(Z)
}
```

---

rppm *Recursively Partitioned Point Process Model*

Description

Fits a recursive partition model to point pattern data.

Usage

```
rppm(..., rpargs=list())
```

Arguments

...	Arguments passed to ppm specifying the point pattern data and the explanatory covariates.
rpargs	Optional list of arguments passed to rpart controlling the recursive partitioning procedure.

Details

This function attempts to find a simple rule for predicting low and high intensity regions of points in a point pattern, using explanatory covariates.

The arguments ... specify the point pattern data and explanatory covariates in the same way as they would be in the function ppm.

The recursive partitioning algorithm rpart is then used to find a partitioning rule.

Value

An object of class "rppm". There are methods for print, plot, fitted, predict and prune for this class.

Author(s)

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

References

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984) *Classification and Regression Trees*. Wadsworth.

See Also

plot.rppm, predict.rppm, prune.rppm.

Examples

```
# New Zealand trees data: trees planted along border
# Use covariates 'x', 'y'
nzfit <- rppm(nztrees ~ x + y)
nzfit
prune(nzfit, cp=0.035)
# Murchison gold data: numeric and logical covariates
mur <- solapply(murchison, rescale, s=1000, unitname="km")
mur$dfault <- distfun(mur$faults)
#
mfit <- rppm(gold ~ dfault + greenstone, data=mur)
mfit
# Gorillas data: factor covariates
#          (symbol '.' indicates 'all variables')
gfit <- rppm(unmark(gorillas) ~ . , data=gorillas.extra)
gfit
```

SatPiece *Piecewise Constant Saturated Pairwise Interaction Point Process Model*

Description

Creates an instance of a saturated pairwise interaction point process model with piecewise constant potential function. The model can then be fitted to point pattern data.

Usage

```
SatPiece(r, sat)
```

Arguments

r	vector of jump points for the potential function
sat	vector of saturation values, or a single saturation value

Details

This is a generalisation of the Geyer saturation point process model, described in Geyer, to the case of multiple interaction distances. It can also be described as the saturated analogue of a pairwise interaction process with piecewise-constant pair potential, described in PairPiece.

The saturated point process with interaction radii $r_1, \ldots, r_k$, saturation thresholds $s_1, \ldots, s_k$, intensity parameter β and interaction parameters $\gamma_1, \ldots, gamma_k$, is the point process in which each point x_i in the pattern X contributes a factor

$$\beta \gamma_1^{v_1(x_i,X)} \ldots gamma_k^{v_k(x_i,X)}$$

to the probability density of the point pattern, where

$$v_j(x_i, X) = \min(s_j, t_j(x_i, X))$$

where $t_j(x_i, X)$ denotes the number of points in the pattern X which lie at a distance between r_{j-1} and r_j from the point x_i. We take $r_0 = 0$ so that $t_1(x_i, X)$ is the number of points of X that lie within a distance r_1 of the point x_i.

`SatPiece` is used to fit this model to data. The function `ppm()`, which fits point process models to point pattern data, requires an argument of class `"interact"` describing the interpoint interaction structure of the model to be fitted. The appropriate description of the piecewise constant Saturated pairwise interaction is yielded by the function `SatPiece()`. See the examples below.

Simulation of this point process model is not yet implemented. This model is not locally stable (the conditional intensity is unbounded).

The argument `r` specifies the vector of interaction distances. The entries of `r` must be strictly increasing, positive numbers.

The argument `sat` specifies the vector of saturation parameters. It should be a vector of the same length as `r`, and its entries should be nonnegative numbers. Thus `sat[1]` corresponds to the distance range from `0` to `r[1]`, and `sat[2]` to the distance range from `r[1]` to `r[2]`, etc. Alternatively `sat` may be a single number, and this saturation value will be applied to every distance range.

Infinite values of the saturation parameters are also permitted; in this case $v_j(x_i, X) = t_j(x_i, X)$ and there is effectively no 'saturation' for the distance range in question. If all the saturation parameters are set to `Inf` then the model is effectively a pairwise interaction process, equivalent to PairPiece (however the interaction parameters γ obtained from SatPiece are the square roots of the parameters γ obtained from PairPiece).

If `r` is a single number, this model is virtually equivalent to the Geyer process, see Geyer.

Value

An object of class `"interact"` describing the interpoint interaction structure of a point process.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

in collaboration with Hao Wang and Jeff Picka

See Also

ppm, pairsat.family, Geyer, PairPiece, BadGey.

Examples

```
SatPiece(c(0.1,0.2), c(1,1))
# prints a sensible description of itself
SatPiece(c(0.1,0.2), 1)

ppm(cells ~1, SatPiece(c(0.07, 0.1, 0.13), 2))
# fit a stationary piecewise constant Saturated pairwise interaction process

# ppm(cells ~polynom(x,y,3), SatPiece(c(0.07, 0.1, 0.13), 2))
# nonstationary process with log-cubic polynomial trend
```

Saturated *Saturated Pairwise Interaction model*

Description

Experimental.

Usage

```
Saturated(pot, name)
```

Arguments

pot	An S language function giving the user-supplied pairwise interaction potential.
name	Character string.

Details

This is experimental. It constructs a member of the "saturated pairwise" family pairsat.family.

Value

An object of class "interact" describing the interpoint interaction structure of a point process.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppm, pairsat.family, Geyer, SatPiece, ppm.object

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 <r.turner@auckland.ac.nz>

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 `<r.turner@auckland.ac.nz>`

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 `j`th column of `B` expresses the `j`th 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 <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

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`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 <r.turner@auckland.ac.nz> 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

```
data(shapley)
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)
```

`simulate.dppm` *Simulation of Determinantal Point Process Model*

Description

Generates simulated realisations from a determinantal point process model.

Usage

```
  ## S3 method for class 'dppm'
simulate(object, nsim = 1, seed = NULL, ...,
    W = NULL, trunc = 0.99, correction = "periodic", rbord = reach(object))

  ## S3 method for class 'detpointprocfamily'
simulate(object, nsim = 1, seed = NULL, ...,
    W = NULL, trunc = 0.99, correction = "periodic", rbord = reach(object))
```

Arguments

`object`	Determinantal point process model. An object of class `"detpointprocfamily"` or `"dppm"`.
`nsim`	Number of simulated realisations.
`seed`	an object specifying whether and how to initialise the random number generator. Either `NULL` or an integer that will be used in a call to `set.seed` before simulating the point patterns.
`...`	Arguments passed on to `rdpp`.
`W`	Object specifying the window of simulation (defaults to a unit box if nothing else is sensible – see Details). Can be any single argument acceptable to `as.boxx` (e.g. an `"owin"`, `"box3"` or `"boxx"` object).
`trunc`	Numeric value specifying how the model truncation is preformed. See Details.
`correction`	Character string specifying the type of correction to use. The options are `"periodic"` (default) and `"border"`. See Details.
`rbord`	Numeric value specifying the extent of the border correction if this correction is used. See Details.

Details

These functions are methods for the generic function simulate for the classes "detpointprocfamily" and "dppm" of determinantal point process models.

The return value is a list of nsim point patterns. It also carries an attribute "seed" that captures the initial state of the random number generator. This follows the convention used in simulate.lm (see simulate). It can be used to force a sequence of simulations to be repeated exactly, as shown in the examples for simulate.

The exact simulation of a determinantal point process model involves an infinite series, which typically has no analytical solution. In the implementation a truncation is performed. The truncation trunc can be specified either directly as a positive integer or as a fraction between 0 and 1. In the latter case the truncation is chosen such that the expected number of points in a simulation is trunc times the theoretical expected number of points in the model. The default is 0.99.

The window of the returned point pattern(s) can be specified via the argument W. For a fitted model (of class "dppm") it defaults to the observation window of the data used to fit the model. For inhomogeneous models it defaults to the window of the intensity image. Otherwise it defaults to a unit box. For non-rectangular windows simulation is done in the containing rectangle and then restricted to the window. For inhomogeneous models a stationary model is first simulated using the maximum intensity and then the result is obtained by thinning.

The default is to use periodic edge correction for simulation such that opposite edges are glued together. If border correction is used then the simulation is done in an extended window. Edge effects are theoretically completely removed by doubling the size of the window in each spatial dimension, but for practical purposes much less extension may be sufficient. The numeric rbord determines the extent of the extra space added to the window.

Value

A list of length nsim containing simulated point patterns. If the patterns are two-dimensional, then they are objects of class "ppp", and the list has class "solist". Otherwise, the patterns are objects of class "ppx" and the list has class "anylist".

The return value also carries an attribute "seed" that captures the initial state of the random number generator. See Details.

Author(s)

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

References

Lavancier, F. Møller, J. and Rubak, E. (2015) Determinantal point process models and statistical inference *Journal of the Royal Statistical Society, Series B* **77**, 853–977.

See Also

rdpp, simulate

Examples

```
if(interactive()) {
   nsim <- 2
   lam <- 100
} else {
   nsim <- 1
   lam <- 30
}
model <- dppGauss(lambda=lam, alpha=.05, d=2)
simulate(model, nsim)
```

`simulate.kppm` *Simulate a Fitted Cluster Point Process Model*

Description

Generates simulated realisations from a fitted cluster point process model.

Usage

```
  ## S3 method for class 'kppm'
simulate(object, nsim = 1, seed=NULL, ...,
         window=NULL, covariates=NULL,
         n.cond = NULL, w.cond = NULL,
         verbose=TRUE, retry=10,
         drop=FALSE)
```

Arguments

`object`	Fitted cluster point process model. An object of class `"kppm"`.
`nsim`	Number of simulated realisations.
`seed`	an object specifying whether and how to initialise the random number generator. Either `NULL` or an integer that will be used in a call to `set.seed` before simulating the point patterns.
`...`	Additional arguments passed to the relevant random generator. See Details.
`window`	Optional. Window (object of class `"owin"`) in which the model should be simulated.
`covariates`	Optional. A named list containing new values for the covariates in the model.
`n.cond`	Optional. Integer specifying a fixed number of points. See the section on *Conditional Simulation*.
`w.cond`	Optional. Conditioning region. A window (object of class `"owin"`) specifying the region which must contain exactly `n.cond` points. See the section on *Conditional Simulation*.
`verbose`	Logical. Whether to print progress reports (when `nsim > 1`).

retry	Number of times to repeat the simulation if it fails (e.g. because of insufficient memory).
drop	Logical. If nsim=1 and drop=TRUE, the result will be a point pattern, rather than a list containing a point pattern.

Details

This function is a method for the generic function simulate for the class "kppm" of fitted cluster point process models.

Simulations are performed by rThomas, rMatClust, rCauchy, rVarGamma or rLGCP depending on the model.

Additional arguments ... are passed to the relevant function performing the simulation. For example the argument saveLambda is recognised by all of the simulation functions.

The return value is a list of point patterns. It also carries an attribute "seed" that captures the initial state of the random number generator. This follows the convention used in simulate.lm (see simulate). It can be used to force a sequence of simulations to be repeated exactly, as shown in the examples for simulate.

Value

A list of length nsim containing simulated point patterns (objects of class "ppp"). (For conditional simulation, the length of the result may be shorter than nsim).

The return value also carries an attribute "seed" that captures the initial state of the random number generator. See Details.

Conditional Simulation

If n.cond is specified, it should be a single integer. Simulation will be conditional on the event that the pattern contains exactly n.cond points (or contains exactly n.cond points inside the region w.cond if it is given).

Conditional simulation uses the rejection algorithm described in Section 6.2 of Moller, Syversveen and Waagepetersen (1998). There is a maximum number of proposals which will be attempted. Consequently the return value may contain fewer than nsim point patterns.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> 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.

Møller, J., Syversveen, A. and Waagepetersen, R. (1998) Log Gaussian Cox Processes. *Scandinavian Journal of Statistics* **25**, 451–482.

See Also

kppm, rThomas, rMatClust, rCauchy, rVarGamma, rLGCP, simulate.ppm, simulate

Examples

```
if(offline <- !interactive()) {
  spatstat.options(npixel=32, ndummy.min=16)
}

fit <- kppm(redwood ~x, "Thomas")
simulate(fit, 2)

simulate(fit, n.cond=60)

if(offline) reset.spatstat.options()
```

---

simulate.mppm *Simulate a Point Process Model Fitted to Several Point Patterns*

Description

Generates simulated realisations from a point process model that was fitted to several point patterns.

Usage

```
  ## S3 method for class 'mppm'
simulate(object, nsim=1, ..., verbose=TRUE)
```

Arguments

object	Point process model fitted to several point patterns. An object of class "mppm".
nsim	Number of simulated realisations (of each original pattern).
...	Further arguments passed to simulate.ppm to control the simulation.
verbose	Logical value indicating whether to print progress reports.

Details

This function is a method for the generic function simulate for the class "mppm" of fitted point process models for replicated point pattern data.

The result is a hyperframe with n rows and nsim columns, where n is the number of original point pattern datasets to which the model was fitted. Each column of the hyperframe contains a simulated version of the original data.

For each of the original point pattern datasets, the fitted model for this dataset is extracted using subfits, then nsim simulated realisations of this model are generated using simulate.ppm, and these are stored in the corresponding row of the output.

Value

A hyperframe.

Author(s)

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

See Also

mppm, simulate.ppm.

Examples

```
H <- hyperframe(Bugs=waterstriders)
fit <- mppm(Bugs ~ id, H)
y <- simulate(fit, nsim=2)
y
plot(y[1,,drop=TRUE], main="Simulations for Waterstriders pattern 1")
plot(y[,1,drop=TRUE], main="Simulation 1 for each Waterstriders pattern")
```

---

simulate.ppm *Simulate a Fitted Gibbs Point Process Model*

Description

Generates simulated realisations from a fitted Gibbs or Poisson point process model.

Usage

```
  ## S3 method for class 'ppm'
simulate(object, nsim=1, ...,
                         singlerun = FALSE,
                         start = NULL,
                         control = default.rmhcontrol(object, w=w),
                         w = window,
                         window = NULL,
                         project=TRUE, new.coef=NULL,
                         verbose=FALSE, progress=(nsim > 1),
                         drop=FALSE)
```

Arguments

object	Fitted point process model. An object of class "ppm".
nsim	Number of simulated realisations.
singlerun	Logical. Whether to generate the simulated realisations from a single long run of the Metropolis-Hastings algorithm (singlerun=TRUE) or from separate, independent runs of the algorithm (singlerun=FALSE, the default).

start	Data determining the initial state of the Metropolis-Hastings algorithm. See rmhstart for description of these arguments. Defaults to list(n.start=npoints(data.ppm(object))) meaning that the initial state of the algorithm has the same number of points as the original dataset.
control	Data controlling the running of the Metropolis-Hastings algorithm. See rmhcontrol for description of these arguments.
w, window	Optional. The window in which the model is defined. An object of class "owin".
...	Further arguments passed to rmhcontrol, or to rmh.default, or to covariate functions in the model.
project	Logical flag indicating what to do if the fitted model is invalid (in the sense that the values of the fitted coefficients do not specify a valid point process). If project=TRUE the closest valid model will be simulated; if project=FALSE an error will occur.
verbose	Logical flag indicating whether to print progress reports from rmh.ppm during the simulation of each point pattern.
progress	Logical flag indicating whether to print progress reports for the sequence of simulations.
new.coef	New values for the canonical parameters of the model. A numeric vector of the same length as coef(object).
drop	Logical. If nsim=1 and drop=TRUE, the result will be a point pattern, rather than a list containing a point pattern.

Details

This function is a method for the generic function simulate for the class "ppm" of fitted point process models.

Simulations are performed by rmh.ppm.

If singlerun=FALSE (the default), the simulated patterns are the results of independent runs of the Metropolis-Hastings algorithm. If singlerun=TRUE, a single long run of the algorithm is performed, and the state of the simulation is saved every nsave iterations to yield the simulated patterns.

In the case of a single run, the behaviour is controlled by the parameters nsave,nburn,nrep. These are described in rmhcontrol. They may be passed in the ... arguments or included in control. It is sufficient to specify two of the three parameters nsave,nburn,nrep.

Value

A list of length nsim containing simulated point patterns (objects of class "ppp"). It also belongs to the class "solist", so that it can be plotted, and the class "timed", so that the total computation time is recorded.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppm, simulate.kppm, simulate

Examples

```
fit <- ppm(japanesepines, ~1, Strauss(0.1))
simulate(fit, 2)
simulate(fit, 2, singlerun=TRUE, nsave=1e4, nburn=1e4)
```

---

simulate.slrm *Simulate a Fitted Spatial Logistic Regression Model*

Description

Generates simulated realisations from a fitted spatial logistic regresson model

Usage

```
  ## S3 method for class 'slrm'
simulate(object, nsim = 1, seed=NULL, ...,
         window=NULL, covariates=NULL, verbose=TRUE, drop=FALSE)
```

Arguments

object	Fitted spatial logistic regression model. An object of class "slrm".
nsim	Number of simulated realisations.
seed	an object specifying whether and how to initialise the random number generator. Either NULL or an integer that will be used in a call to set.seed before simulating the point patterns.
...	Ignored.
window	Optional. Window (object of class "owin") in which the model should be simulated.
covariates	Optional. A named list containing new values for the covariates in the model.
verbose	Logical. Whether to print progress reports (when nsim > 1).
drop	Logical. If nsim=1 and drop=TRUE, the result will be a point pattern, rather than a list containing a point pattern.

Details

This function is a method for the generic function simulate for the class "slrm" of fitted spatial logistic regression models.

Simulations are performed by rpoispp after the intensity has been computed by predict.slrm.

The return value is a list of point patterns. It also carries an attribute "seed" that captures the initial state of the random number generator. This follows the convention used in simulate.lm (see simulate). It can be used to force a sequence of simulations to be repeated exactly, as shown in the examples for simulate.

Value

A list of length nsim containing simulated point patterns (objects of class "ppp").

The return value also carries an attribute "seed" that captures the initial state of the random number generator. See Details.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

slrm, rpoispp, simulate.ppm, simulate.kppm, simulate

Examples

```
X <- copper$SouthPoints
fit <- slrm(X ~ 1)
simulate(fit, 2)
fitxy <- slrm(X ~ x+y)
simulate(fitxy, 2, window=square(2))
```

---

slrm *Spatial Logistic Regression*

Description

Fits a spatial logistic regression model to a spatial point pattern.

Usage

```
slrm(formula, ..., data = NULL, offset = TRUE, link = "logit",
                   dataAtPoints=NULL, splitby=NULL)
```

Arguments

formula	The model formula. See Details.
...	Optional arguments passed to as.mask determining the pixel resolution for the discretisation of the point pattern.
data	Optional. A list containing data required in the formula. The names of entries in the list should correspond to variable names in the formula. The entries should be point patterns, pixel images or windows.
offset	Logical flag indicating whether the model formula should be augmented by an offset equal to the logarithm of the pixel area.
link	The link function for the regression model. A character string, specifying a link function for binary regression.
dataAtPoints	Optional. Exact values of the covariates at the data points. A data frame, with column names corresponding to variables in the formula, with one row for each point in the point pattern dataset.
splitby	Optional. Character string identifying a window. The window will be used to split pixels into sub-pixels.

Details

This function fits a Spatial Logistic Regression model (Tukey, 1972; Agterberg, 1974) to a spatial point pattern dataset. The logistic function may be replaced by another link function.

The formula specifies the form of the model to be fitted, and the data to which it should be fitted. The formula must be an R formula with a left and right hand side.

The left hand side of the formula is the name of the point pattern dataset, an object of class "ppp".

The right hand side of the formula is an expression, in the usual R formula syntax, representing the functional form of the linear predictor for the model.

Each variable name that appears in the formula may be

- one of the reserved names x and y, referring to the Cartesian coordinates;
- the name of an entry in the list data, if this argument is given;
- the name of an object in the parent environment, that is, in the environment where the call to slrm was issued.

Each object appearing on the right hand side of the formula may be

- a pixel image (object of class "im") containing the values of a covariate;
- a window (object of class "owin"), which will be interpreted as a logical covariate which is TRUE inside the window and FALSE outside it;
- a function in the R language, with arguments x,y, which can be evaluated at any location to obtain the values of a covariate.

See the Examples below.

The fitting algorithm discretises the point pattern onto a pixel grid. The value in each pixel is 1 if there are any points of the point pattern in the pixel, and 0 if there are no points in the pixel. The dimensions of the pixel grid will be determined as follows:

- The pixel grid will be determined by the extra arguments ... if they are specified (for example the argument dimyx can be used to specify the number of pixels).
- Otherwise, if the right hand side of the formula includes the names of any pixel images containing covariate values, these images will determine the pixel grid for the discretisation. The covariate image with the finest grid (the smallest pixels) will be used.
- Otherwise, the default pixel grid size is given by spatstat.options("npixel").

The covariates are evaluated at the centre of each pixel. If dataAtPoints is given, then the covariate values at the corresponding pixels are overwritten by the entries of dataAtPoints (and the spatial coordinates are overwritten by the exact spatial coordinates of the data points).

If link="logit" (the default), the algorithm fits a Spatial Logistic Regression model. This model states that the probability p that a given pixel contains a data point, is related to the covariates through

$$\log \frac{p}{1-p} = \eta$$

where η is the linear predictor of the model (a linear combination of the covariates, whose form is specified by the formula).

If link="cloglog" then the algorithm fits a model stating that

$$\log(-\log(1-p)) = \eta$$

.

If offset=TRUE (the default), the model formula will be augmented by adding an offset term equal to the logarithm of the pixel area. This ensures that the fitted parameters are approximately independent of pixel size. If offset=FALSE, the offset is not included, and the traditional form of Spatial Logistic Regression is fitted.

Value

An object of class "slrm" representing the fitted model.

There are many methods for this class, including methods for print, fitted, predict, anova, coef, logLik, terms, update, formula and vcov. Automated stepwise model selection is possible using step. Confidence intervals for the parameters can be computed using confint.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <r.turner@auckland.ac.nz>.

References

Agterberg, F.P. (1974) Automatic contouring of geological maps to detect target areas for mineral exploration. *Journal of the International Association for Mathematical Geology* **6**, 373–395.

Baddeley, A., Berman, M., Fisher, N.I., Hardegen, A., Milne, R.K., Schuhmacher, D., Shah, R. and Turner, R. (2010) Spatial logistic regression and change-of-support for spatial Poisson point processes. *Electronic Journal of Statistics* **4**, 1151–1201. DOI: 10.1214/10-EJS581

Tukey, J.W. (1972) Discussion of paper by F.P. Agterberg and S.C. Robinson. *Bulletin of the International Statistical Institute* **44** (1) p. 596. Proceedings, 38th Congress, International Statistical Institute.

See Also

anova.slrm, coef.slrm, fitted.slrm, logLik.slrm, plot.slrm, predict.slrm, vcov.slrm

Examples

```
if(offline <- !interactive()) op <- spatstat.options(npixel=32)

     X <- copper$SouthPoints
     slrm(X ~ 1)
     slrm(X ~ x+y)

     slrm(X ~ x+y, link="cloglog")
     # specify a grid of 2-km-square pixels
     slrm(X ~ 1, eps=2)

     Y <- copper$SouthLines
     Z <- distmap(Y)
     slrm(X ~ Z)
     slrm(X ~ Z, dataAtPoints=list(Z=nncross(X,Y,what="dist")))

     mur <- murchison
     mur$dfault <- distfun(mur$faults)
     slrm(gold ~ dfault, data=mur)
     slrm(gold ~ dfault + greenstone, data=mur)
     slrm(gold ~ dfault, data=mur, splitby="greenstone")

if(offline) spatstat.options(op)
```

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 <r.turner@auckland.ac.nz>

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 <r.turner@auckland.ac.nz>

See Also

Smooth, with.fv, fv.object, smooth.spline, smooth.spline

Examples

```
data(cells)
G <- Gest(cells)
plot(G)
plot(Smooth(G, df=9), add=TRUE)
```

Smooth.msr *Smooth a Signed or Vector-Valued Measure*

Description

Apply kernel smoothing to a signed measure or vector-valued measure.

Usage

```
 ## S3 method for class 'msr'
Smooth(X, ..., drop=TRUE)
```

Arguments

X	Object of class "msr" representing a signed measure or vector-valued measure.
...	Arguments passed to density.ppp controlling the smoothing bandwidth and the pixel resolution.
drop	Logical. If TRUE (the default), the result of smoothing a scalar-valued measure is a pixel image. If FALSE, the result of smoothing a scalar-valued measure is a list containing one pixel image.

Details

This function applies kernel smoothing to a signed measure or vector-valued measure X. The Gaussian kernel is used.

The object X would typically have been created by residuals.ppm or msr.

Value

A pixel image or a list of pixel images. For scalar-valued measures, a pixel image (object of class "im") provided drop=TRUE. For vector-valued measures (or if drop=FALSE), a list of pixel images; the list also belongs to the class "solist" so that it can be printed and plotted.

Author(s)

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

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.

Baddeley, A., Møller, J. and Pakes, A.G. (2008) Properties of residuals for spatial point processes. *Annals of the Institute of Statistical Mathematics* **60**, 627–649.

See Also

Smooth, msr, plot.msr

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X, ~x+y)
rp <- residuals(fit, type="pearson")
rs <- residuals(fit, type="score")

plot(Smooth(rp))
plot(Smooth(rs))
```

`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",
                     adjust=1, varcov=NULL,
                     edge=TRUE, diggle=FALSE,
                     kernel="gaussian", scalekernel=is.character(kernel),
                     geometric=FALSE)

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"`).
`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.

geometric — Logical value indicating whether to perform geometric mean smoothing instead of arithmetic mean smoothing. See Details.

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. 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 <r.turner@auckland.ac.nz> 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 <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

Smooth

Examples

```
f <- Smoothfun(longleaf)
f
f(120, 80)
plot(f)
```

Softcore *The Soft Core Point Process Model*

Description

Creates an instance of the Soft Core point process model which can then be fitted to point pattern data.

Usage

```
Softcore(kappa, sigma0=NA)
```

Arguments

kappa	The exponent κ of the Soft Core interaction
sigma0	Optional. Initial estimate of the parameter σ. A positive number.

Details

The (stationary) Soft Core point process with parameters β and σ and exponent κ is the pairwise interaction point process in which each point contributes a factor β to the probability density of the point pattern, and each pair of points contributes a factor

$$\exp\left\{-\left(\frac{\sigma}{d}\right)^{2/\kappa}\right\}$$

to the density, where d is the distance between the two points. See the Examples for a plot of this interaction curve.

Thus the process has probability density

$$f(x_1, \ldots, x_n) = \alpha\beta^{n(x)} \exp\left\{-\sum_{i<j}\left(\frac{\sigma}{||x_i - x_j||}\right)^{2/\kappa}\right\}$$

where $x_1, \ldots, x_n$ represent the points of the pattern, $n(x)$ is the number of points in the pattern, α is the normalising constant, and the sum on the right hand side is over all unordered pairs of points of the pattern.

This model describes an "ordered" or "inhibitive" process, with the strength of inhibition decreasing smoothly with distance. The interaction is controlled by the parameters σ and κ.

- The *spatial scale* of interaction is controlled by the parameter σ, which is a positive real number interpreted as a distance, expressed in the same units of distance as the spatial data. The parameter σ is the distance at which the pair potential reaches the threshold value 0.37.
- The *shape* of the interaction function is controlled by the exponent κ which is a dimensionless number in the range $(0, 1)$, with larger values corresponding to a flatter shape (or a more gradual decay rate). The process is well-defined only for κ in $(0, 1)$. The limit of the model as $\kappa \to 0$ is the hard core process with hard core distance $h = \sigma$.
- The "strength" of the interaction is determined by both of the parameters σ and κ. The larger the value of κ, the wider the range of distances over which the interaction has an effect. If σ is very small, the interaction is very weak for all practical purposes (theoretically if $\sigma = 0$ the model reduces to the Poisson point process).

The nonstationary Soft Core process is similar except that the contribution of each individual point x_i is a function $\beta(x_i)$ of location, rather than a constant beta.

The function `ppm()`, which fits point process models to point pattern data, requires an argument of class `"interact"` describing the interpoint interaction structure of the model to be fitted. The appropriate description of the Soft Core process pairwise interaction is yielded by the function `Softcore()`. See the examples below.

The main argument is the exponent `kappa`. When `kappa` is fixed, the model becomes an exponential family with canonical parameters $\log \beta$ and

$$\log \gamma = \frac{2}{\kappa} \log \sigma$$

The canonical parameters are estimated by `ppm()`, not fixed in `Softcore()`.

The optional argument `sigma0` can be used to improve numerical stability. If `sigma0` is given, it should be a positive number, and it should be a rough estimate of the parameter σ.

Value

An object of class `"interact"` describing the interpoint interaction structure of the Soft Core process with exponent κ.

Author(s)

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

References

Ogata, Y, and Tanemura, M. (1981). Estimation of interaction potentials of spatial point patterns through the maximum likelihood procedure. *Annals of the Institute of Statistical Mathematics*, B **33**, 315–338.

Ogata, Y, and Tanemura, M. (1984). Likelihood analysis of spatial point patterns. *Journal of the Royal Statistical Society, series B* **46**, 496–518.

See Also

`ppm`, `pairwise.family`, `ppm.object`

Examples

```
# fit the stationary Soft Core process to `cells'
fit5 <- ppm(cells ~1, Softcore(kappa=0.5), correction="isotropic")

# study shape of interaction and explore effect of parameters
fit2 <- update(fit5, Softcore(kappa=0.2))
fit8 <- update(fit5, Softcore(kappa=0.8))
plot(fitin(fit2), xlim=c(0, 0.4),
     main="Pair potential (sigma = 0.1)",
     xlab=expression(d), ylab=expression(h(d)), legend=FALSE)
plot(fitin(fit5), add=TRUE, col=4)
plot(fitin(fit8), add=TRUE, col=3)
legend("bottomright", col=c(1,4,3), lty=1,
       legend=expression(kappa==0.2, kappa==0.5, kappa==0.8))
```

---

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 <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

See Also

ewcdf, cdf.test

Examples

```
 with(bei.extra, {
   plot(spatialcdf(grad))
   fit <- ppm(bei ~ elev)
   plot(spatialcdf(grad, predict(fit)))
   plot(A <- spatialcdf(grad, fit))
   A(0.1)
})
plot(spatialcdf("x", W=letterR))
```

`split.msr` *Divide a Measure into Parts*

Description

Decomposes a measure into components, each component being a measure.

Usage

```
  ## S3 method for class 'msr'
split(x, f, drop = FALSE, ...)
```

Arguments

`x`	Measure (object of class `"msr"`) to be decomposed.
`f`	Factor or tessellation determining the decomposition. Argument passed to `split.ppp`. See Details.
`drop`	Logical value indicating whether empty components should be retained in the list (`drop=FALSE`, the default) or deleted (`drop=TRUE`).
`...`	Ignored.

Details

An object of class `"msr"` represents a signed (i.e. real-valued) or vector-valued measure in the **spatstat** package. See `msr` for explanation.

This function is a method for the generic `split`. It divides the measure `x` into components, each of which is a measure.

A measure `x` is represented in **spatstat** by a finite set of sample points with values attached to them. The function `split.msr` divides this pattern of sample points into several sub-patterns of points using `split.ppp`. For each sub-pattern, the values attached to these points are extracted from `x`, and these values and sample points determine a measure, which is a component or piece of the original `x`.

The argument `f` can be missing, if the sample points of `x` are multitype points. In this case, `x` represents a measure associated with marked spatial locations, and the command `split(x)` separates `x` into a list of component measures, one for each possible mark.

Otherwise the argument `f` is passed to `split.ppp`. It should be either a factor (of length equal to the number of sample points of `x`) or a tessellation (object of class `"tess"` representing a division of space into tiles) as documented under `split.ppp`.

Value

A list, each of whose entries is a measure (object of class `"msr"`).

Author(s)

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

See Also

msr, [.msr, with.msr

Examples

```
## split by tessellation
a <- residuals(ppm(cells ~ x))
aa <- split(a, dirichlet(runifpoint(4)))
aa
sapply(aa, integral)

## split by type of point
b <- residuals(ppm(amacrine ~ marks + x))
bb <- split(b)
bb
```

---

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))
```

---

stieltjes *Compute Integral of Function Against Cumulative Distribution*

Description

Computes the Stieltjes integral of a function f with respect to a function M.

Usage

```
stieltjes(f, M, ...)
```

Arguments

f	The integrand. A function in the R language.
M	The cumulative function against which f will be integrated. An object of class "fv" or "stepfun".
...	Additional arguments passed to f.

Details

This command computes the Stieltjes integral

$$I = \int f(x) dM(x)$$

of a real-valued function $f(x)$ with respect to a nondecreasing function $M(x)$.

One common use of the Stieltjes integral is to find the mean value of a random variable from its cumulative distribution function $F(x)$. The mean value is the Stieltjes integral of $f(x) = x$ with respect to $F(x)$.

The argument f should be a function in the R language. It should accept a numeric vector argument x and should return a numeric vector of the same length.

The argument M should be either a step function (object of class "stepfun") or a function value table (object of class "fv", see fv.object). Objects of class "stepfun" are returned by ecdf, ewcdf, spatialcdf and other utilities. Objects of class "fv" are returned by the commands Kest, Gest, etc.

Value

A list containing the value of the Stieltjes integral computed using each of the versions of the function M.

Author(s)

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

See Also

fv.object, Gest

Examples

```
# estimate cdf of nearest neighbour distance in redwood data
G <- Gest(redwood)
# compute estimate of mean nearest neighbour distance
stieltjes(function(x){x}, G)
# estimated probability of a distance in the interval [0.1,0.2]
stieltjes(function(x,a,b){ (x >= a) & (x <= b)}, G, a=0.1, b=0.2)

# stepfun example
H <- spatialcdf(bei.extra$elev, normalise=TRUE)
stieltjes(function(x){x}, H)
```

---

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 <r.turner@auckland.ac.nz>

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"))
```

`Strauss` *The Strauss Point Process Model*

Description

Creates an instance of the Strauss point process model which can then be fitted to point pattern data.

Usage

```
Strauss(r)
```

Arguments

`r`	The interaction radius of the Strauss process

Details

The (stationary) Strauss process with interaction radius r and parameters β and γ is the pairwise interaction point process in which each point contributes a factor β to the probability density of the point pattern, and each pair of points closer than r units apart contributes a factor γ to the density.

Thus the probability density is

$$f(x_1, \ldots, x_n) = \alpha \beta^{n(x)} \gamma^{s(x)}$$

where $x_1, \ldots, x_n$ represent the points of the pattern, $n(x)$ is the number of points in the pattern, $s(x)$ is the number of distinct unordered pairs of points that are closer than r units apart, and α is the normalising constant.

The interaction parameter γ must be less than or equal to 1 so that this model describes an "ordered" or "inhibitive" pattern.

The nonstationary Strauss process is similar except that the contribution of each individual point x_i is a function $\beta(x_i)$ of location, rather than a constant beta.

The function `ppm()`, which fits point process models to point pattern data, requires an argument of class `"interact"` describing the interpoint interaction structure of the model to be fitted. The appropriate description of the Strauss process pairwise interaction is yielded by the function `Strauss()`. See the examples below.

Note the only argument is the interaction radius `r`. When `r` is fixed, the model becomes an exponential family. The canonical parameters $\log(\beta)$ and $\log(\gamma)$ are estimated by `ppm()`, not fixed in `Strauss()`.

Value

An object of class `"interact"` describing the interpoint interaction structure of the Strauss process with interaction radius r.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

References

Kelly, F.P. and Ripley, B.D. (1976) On Strauss's model for clustering. *Biometrika* **63**, 357–360.

Strauss, D.J. (1975) A model for clustering. *Biometrika* **62**, 467–475.

See Also

ppm, pairwise.family, ppm.object

Examples

```
Strauss(r=0.1)
# prints a sensible description of itself

# ppm(cells ~1, Strauss(r=0.07))
# fit the stationary Strauss process to `cells'

ppm(cells ~polynom(x,y,3), Strauss(r=0.07))
# fit a nonstationary Strauss process with log-cubic polynomial trend
```

StraussHard *The Strauss / Hard Core Point Process Model*

Description

Creates an instance of the "Strauss/ hard core" point process model which can then be fitted to point pattern data.

Usage

```
StraussHard(r, hc=NA)
```

Arguments

r	The interaction radius of the Strauss interaction
hc	The hard core distance. Optional.

Details

A Strauss/hard core process with interaction radius r, hard core distance $h < r$, and parameters β and γ, is a pairwise interaction point process in which

- distinct points are not allowed to come closer than a distance h apart
- each pair of points closer than r units apart contributes a factor γ to the probability density.

This is a hybrid of the Strauss process and the hard core process.

The probability density is zero if any pair of points is closer than h units apart, and otherwise equals

$$f(x_1, \ldots, x_n) = \alpha \beta^{n(x)} \gamma^{s(x)}$$

where $x_1, \ldots, x_n$ represent the points of the pattern, $n(x)$ is the number of points in the pattern, $s(x)$ is the number of distinct unordered pairs of points that are closer than r units apart, and α is the normalising constant.

The interaction parameter γ may take any positive value (unlike the case for the Strauss process). If $\gamma < 1$, the model describes an "ordered" or "inhibitive" pattern. If $\gamma > 1$, the model is "ordered" or "inhibitive" up to the distance h, but has an "attraction" between points lying at distances in the range between h and r.

If $\gamma = 1$, the process reduces to a classical hard core process with hard core distance h. If $\gamma = 0$, the process reduces to a classical hard core process with hard core distance r.

The function `ppm()`, which fits point process models to point pattern data, requires an argument of class `"interact"` describing the interpoint interaction structure of the model to be fitted. The appropriate description of the Strauss/hard core process pairwise interaction is yielded by the function `StraussHard()`. See the examples below.

The canonical parameter $\log(\gamma)$ is estimated by `ppm()`, not fixed in `StraussHard()`.

If the hard core distance argument `hc` is missing or `NA`, it will be estimated from the data when `ppm` is called. The estimated value of `hc` is the minimum nearest neighbour distance multiplied by $n/(n+1)$, where n is the number of data points.

Value

An object of class `"interact"` describing the interpoint interaction structure of the "Strauss/hard core" process with Strauss interaction radius r and hard core distance `hc`.

Author(s)

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

and Rolf Turner `<r.turner@auckland.ac.nz>`

References

Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.

Ripley, B.D. (1981) *Spatial statistics*. John Wiley and Sons.

Strauss, D.J. (1975) A model for clustering. *Biometrika* **62**, 467–475.

See Also

`ppm`, `pairwise.family`, `ppm.object`

Examples

```
StraussHard(r=1,hc=0.02)
# prints a sensible description of itself

data(cells)

# ppm(cells, ~1, StraussHard(r=0.1, hc=0.05))
# fit the stationary Strauss/hard core  process to `cells'

ppm(cells, ~ polynom(x,y,3), StraussHard(r=0.1, hc=0.05))
# fit a nonstationary Strauss/hard core process
# with log-cubic polynomial trend
```

`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 `<r.turner@auckland.ac.nz>` 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
```

---

subfits *Extract List of Individual Point Process Models*

Description

Takes a Gibbs point process model that has been fitted to several point patterns simultaneously, and produces a list of fitted point process models for the individual point patterns.

Usage

```
subfits(object, what="models", verbose=FALSE, new.coef=NULL)
subfits.old(object, what="models", verbose=FALSE, new.coef=NULL)
subfits.new(object, what="models", verbose=FALSE)
```

Arguments

object	An object of class "mppm" representing a point process model fitted to several point patterns.
what	What should be returned. Either "models" to return the fitted models, or "interactions" to return the fitted interactions only.
verbose	Logical flag indicating whether to print progress reports.
new.coef	Advanced use only. Numeric vector or matrix of coefficients to replaced the fitted coefficients coef(object).

Details

object is assumed to have been generated by mppm. It represents a point process model that has been fitted to a list of several point patterns, with covariate data.

For each of the *individual* point pattern datasets, this function derives the corresponding fitted model for that dataset only (i.e. a point process model for the ith point pattern, that is consistent with object).

If what="models", the result is a list of point process models (a list of objects of class "ppm"), one model for each point pattern dataset in the original fit. If what="interactions", the result is a list of fitted interpoint interactions (a list of objects of class "fii").

Two different algorithms are provided, as subfits.old and subfits.new. Currently subfits is the same as the old algorithm subfits.old because the newer algorithm is too memory-hungry.

Value

A list of point process models (a list of objects of class "ppm") or a list of fitted interpoint interactions (a list of objects of class "fii").

Author(s)

Adrian Baddeley, Ida-Maria Sintorn and Leanne Bischoff. Implemented in **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

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

See Also

mppm, ppm

Examples

```
H <- hyperframe(Wat=waterstriders)
fit <- mppm(Wat~x, data=H)
subfits(fit)

H$Wat[[3]] <- rthin(H$Wat[[3]], 0.1)
fit2 <- mppm(Wat~x, data=H, random=~1|id)
subfits(fit2)
```

---

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.

---

suffstat *Sufficient Statistic of Point Process Model*

Description

The canonical sufficient statistic of a point process model is evaluated for a given point pattern.

Usage

```
suffstat(model, X=data.ppm(model))
```

Arguments

model	A fitted point process model (object of class "ppm").
X	A point pattern (object of class "ppp").

Details

The canonical sufficient statistic of model is evaluated for the point pattern X. This computation is useful for various Monte Carlo methods.

Here model should be a point process model (object of class "ppm", see ppm.object), typically obtained from the model-fitting function ppm. The argument X should be a point pattern (object of class "ppp").

Every point process model fitted by ppm has a probability density of the form

$$f(x) = Z(\theta) \exp(\theta^T S(x))$$

where x denotes a typical realisation (i.e. a point pattern), θ is the vector of model coefficients, $Z(\theta)$ is a normalising constant, and $S(x)$ is a function of the realisation x, called the "canonical sufficient statistic" of the model.

For example, the stationary Poisson process has canonical sufficient statistic $S(x) = n(x)$, the number of points in x. The stationary Strauss process with interaction range r (and fitted with no edge correction) has canonical sufficient statistic $S(x) = (n(x), s(x))$ where $s(x)$ is the number of pairs of points in x which are closer than a distance r to each other.

`suffstat(model, X)` returns the value of $S(x)$, where S is the canonical sufficient statistic associated with `model`, evaluated when x is the given point pattern `X`. The result is a numeric vector, with entries which correspond to the entries of the coefficient vector `coef(model)`.

The sufficient statistic S does not depend on the fitted coefficients of the model. However it does depend on the irregular parameters which are fixed in the original call to ppm, for example, the interaction range `r` of the Strauss process.

The sufficient statistic also depends on the edge correction that was used to fit the model. For example in a Strauss process,

- If the model is fitted with `correction="none"`, the sufficient statistic is $S(x) = (n(x), s(x))$ where $n(x)$ is the number of points and $s(x)$ is the number of pairs of points which are closer than r units apart.
- If the model is fitted with `correction="periodic"`, the sufficient statistic is the same as above, except that distances are measured in the periodic sense.
- If the model is fitted with `correction="translate"`, then $n(x)$ is unchanged but $s(x)$ is replaced by a weighted sum (the sum of the translation correction weights for all pairs of points which are closer than r units apart).
- If the model is fitted with `correction="border"` (the default), then points lying less than r units from the boundary of the observation window are treated as fixed. Thus $n(x)$ is replaced by the number $n_r(x)$ of points lying at least r units from the boundary of the observation window, and $s(x)$ is replaced by the number $s_r(x)$ of pairs of points, which are closer than r units apart, and at least one of which lies more than r units from the boundary of the observation window.

Non-finite values of the sufficient statistic (`NA` or `-Inf`) may be returned if the point pattern `X` is not a possible realisation of the model (i.e. if `X` has zero probability of occurring under `model` for all values of the canonical coefficients θ).

Value

A numeric vector of sufficient statistics. The entries correspond to the model coefficients `coef(model)`.

Author(s)

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

See Also

ppm

Examples

```
fitS <- ppm(swedishpines~1, Strauss(7))
suffstat(fitS)
X <- rpoispp(intensity(swedishpines), win=Window(swedishpines))
suffstat(fitS, X)
```

---

summary.dppm *Summarizing a Fitted Determinantal Point Process Model*

Description

summary method for class "dppm".

Usage

```
  ## S3 method for class 'dppm'
summary(object, ..., quick=FALSE)

  ## S3 method for class 'summary.dppm'
print(x, ...)
```

Arguments

object	A fitted determinantal point process model (object of class "dppm").
quick	Logical value controlling the scope of the summary.
...	Arguments passed to summary.ppm or print.summary.ppm controlling the treatment of the trend component of the model.
x	Object of class "summary.dppm" as returned by summary.dppm.

Details

This is a method for the generic summary for the class "dppm". An object of class "dppm" describes a fitted determinantal point process model. See dppm.

summary.dppm extracts information about the type of model that has been fitted, the data to which the model was fitted, and the values of the fitted coefficients.

print.summary.dppm prints this information in a comprehensible format.

In normal usage, print.summary.dppm is invoked implicitly when the user calls summary.dppm without assigning its value to anything. See the examples.

Value

summary.dppm returns an object of class "summary.dppm", while print.summary.dppm returns NULL.

The result of summary.dppm includes at least the following components:

Xname	character string name of the original point pattern data

stationary	logical value indicating whether the model is stationary
trend	Object of class summary.ppm summarising the trend
repul	Repulsiveness index

Author(s)

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

Examples

```
  jpines <- residualspaper$Fig1

fit <- dppm(jpines ~ 1, dppGauss)
summary(fit)
```

---

summary.kppm *Summarizing a Fitted Cox or Cluster Point Process Model*

Description

summary method for class "kppm".

Usage

```
  ## S3 method for class 'kppm'
summary(object, ..., quick=FALSE)

  ## S3 method for class 'summary.kppm'
print(x, ...)
```

Arguments

object	A fitted Cox or cluster point process model (object of class "kppm").
quick	Logical value controlling the scope of the summary.
...	Arguments passed to summary.ppm or print.summary.ppm controlling the treatment of the trend component of the model.
x	Object of class "summary.kppm" as returned by summary.kppm.

Details

This is a method for the generic summary for the class "kppm". An object of class "kppm" describes a fitted Cox or cluster point process model. See kppm.

summary.kppm extracts information about the type of model that has been fitted, the data to which the model was fitted, and the values of the fitted coefficients.

print.summary.kppm prints this information in a comprehensible format.

In normal usage, print.summary.kppm is invoked implicitly when the user calls summary.kppm without assigning its value to anything. See the examples.

You can also type coef(summary(object)) to extract a table of the fitted coefficients of the point process model object together with standard errors and confidence limits.

Value

summary.kppm returns an object of class "summary.kppm", while print.summary.kppm returns NULL.

The result of summary.kppm includes at least the following components:

Xname	character string name of the original point pattern data
stationary	logical value indicating whether the model is stationary
clusters	the clusters argument to kppm
modelname	character string describing the model
isPCP	TRUE if the model is a Poisson cluster process, FALSE if it is a log-Gaussian Cox process
lambda	Estimated intensity: numeric value, or pixel image
mu	Mean cluster size: numeric value, pixel image, or NULL
clustpar	list of fitted parameters for the cluster model
clustargs	list of fixed parameters for the cluster model, if any
callstring	character string representing the original call to kppm

Author(s)

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

Examples

```
fit <- kppm(redwood ~ 1, "Thomas")
summary(fit)
coef(summary(fit))
```

---

summary.ppm *Summarizing a Fitted Point Process Model*

Description

summary method for class "ppm".

Usage

```
  ## S3 method for class 'ppm'
summary(object, ..., quick=FALSE, fine=FALSE)
  ## S3 method for class 'summary.ppm'
print(x, ...)
```

Arguments

object	A fitted point process model.
...	Ignored.
quick	Logical flag controlling the scope of the summary.
fine	Logical value passed to vcov.ppm determining whether to compute the quick, coarse estimate of variance (fine=FALSE, the default) or the slower, finer estimate (fine=TRUE).
x	Object of class "summary.ppm" as returned by summary.ppm.

Details

This is a method for the generic summary for the class "ppm". An object of class "ppm" describes a fitted point process model. See ppm.object) for details of this class.

summary.ppm extracts information about the type of model that has been fitted, the data to which the model was fitted, and the values of the fitted coefficients. (If quick=TRUE then only the information about the type of model is extracted.)

print.summary.ppm prints this information in a comprehensible format.

In normal usage, print.summary.ppm is invoked implicitly when the user calls summary.ppm without assigning its value to anything. See the examples.

You can also type coef(summary(object)) to extract a table of the fitted coefficients of the point process model object together with standard errors and confidence limits.

Value

summary.ppm returns an object of class "summary.ppm", while print.summary.ppm returns NULL.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

Examples

```
# invent some data
X <- rpoispp(42)
# fit a model to it
fit <- ppm(X ~ x, Strauss(r=0.1))
# summarize the fitted model
summary(fit)
# `quick' option
summary(fit, quick=TRUE)
# coefficients with standard errors and CI
coef(summary(fit))
coef(summary(fit, fine=TRUE))

# save the full summary
s <- summary(fit)
```

```
# print it
print(s)
s
# extract stuff
names(s)
coef(s)
s$args$correction
s$name
s$trend$value

# multitype pattern
# data(demopat)
# fit <- ppm(demopat, ~marks, Poisson())
# summary(fit)

# model with external covariates
fitX <- ppm(X, ~Z, covariates=list(Z=function(x,y){x+y}))
summary(fitX)
```

`thomas.estK` *Fit the Thomas Point Process by Minimum Contrast*

Description

Fits the Thomas point process to a point pattern dataset by the Method of Minimum Contrast using the K function.

Usage

```
thomas.estK(X, startpar=c(kappa=1,scale=1), lambda=NULL,
            q = 1/4, p = 2, rmin = NULL, rmax = NULL, ...)
```

Arguments

`X`	Data to which the Thomas model will be fitted. Either a point pattern or a summary statistic. See Details.
`startpar`	Vector of starting values for the parameters of the Thomas process.
`lambda`	Optional. An estimate of the intensity of the point process.
`q,p`	Optional. Exponents for the contrast criterion.
`rmin, rmax`	Optional. The interval of r values for the contrast criterion.
`...`	Optional arguments passed to `optim` to control the optimisation algorithm. See Details.

Details

This algorithm fits the Thomas point process model to a point pattern dataset by the Method of Minimum Contrast, using the K function.

The argument `X` can be either

a point pattern: An object of class `"ppp"` representing a point pattern dataset. The K function of the point pattern will be computed using `Kest`, and the method of minimum contrast will be applied to this.

a summary statistic: An object of class `"fv"` containing the values of a summary statistic, computed for a point pattern dataset. The summary statistic should be the K function, and this object should have been obtained by a call to `Kest` or one of its relatives.

The algorithm fits the Thomas point process to `X`, by finding the parameters of the Thomas model which give the closest match between the theoretical K function of the Thomas process and the observed K function. For a more detailed explanation of the Method of Minimum Contrast, see `mincontrast`.

The Thomas point process is described in Møller and Waagepetersen (2003, pp. 61–62). It is a cluster process formed by taking a pattern of parent points, generated according to a Poisson process with intensity κ, and around each parent point, generating a random number of offspring points, such that the number of offspring of each parent is a Poisson random variable with mean μ, and the locations of the offspring points of one parent are independent and isotropically Normally distributed around the parent point with standard deviation σ which is equal to the parameter `scale`. The named vector of stating values can use either `sigma2` (σ^2) or `scale` as the name of the second component, but the latter is recommended for consistency with other cluster models.

The theoretical K-function of the Thomas process is

$$K(r) = \pi r^2 + \frac{1}{\kappa}(1 - \exp(-\frac{r^2}{4\sigma^2})).$$

The theoretical intensity of the Thomas process is $\lambda = \kappa\mu$.

In this algorithm, the Method of Minimum Contrast is first used to find optimal values of the parameters κ and σ^2. Then the remaining parameter μ is inferred from the estimated intensity λ.

If the argument `lambda` is provided, then this is used as the value of λ. Otherwise, if `X` is a point pattern, then λ will be estimated from `X`. If `X` is a summary statistic and `lambda` is missing, then the intensity λ cannot be estimated, and the parameter μ will be returned as `NA`.

The remaining arguments `rmin,rmax,q,p` control the method of minimum contrast; see `mincontrast`.

The Thomas process can be simulated, using `rThomas`.

Homogeneous or inhomogeneous Thomas process models can also be fitted using the function `kppm`.

The optimisation algorithm can be controlled through the additional arguments `"..."` which are passed to the optimisation function `optim`. For example, to constrain the parameter values to a certain range, use the argument `method="L-BFGS-B"` to select an optimisation algorithm that respects box constraints, and use the arguments `lower` and `upper` to specify (vectors of) minimum and maximum values for each parameter.

Value

An object of class "minconfit". There are methods for printing and plotting this object. It contains the following main components:

par	Vector of fitted parameter values.
fit	Function value table (object of class "fv") containing the observed values of the summary statistic (observed) and the theoretical values of the summary statistic computed from the fitted model parameters.

Author(s)

Rasmus Waagepetersen <rw@math.auc.dk> Adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

References

Diggle, P. J., Besag, J. and Gleaves, J. T. (1976) Statistical analysis of spatial point patterns by means of distance methods. *Biometrics* **32** 659–667.

Møller, J. and Waagepetersen, R. (2003). Statistical Inference and Simulation for Spatial Point Processes. Chapman and Hall/CRC, Boca Raton.

Thomas, M. (1949) A generalisation of Poisson's binomial limit for use in ecology. *Biometrika* **36**, 18–25.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

kppm, lgcp.estK, matclust.estK, mincontrast, Kest, rThomas to simulate the fitted model.

Examples

```
data(redwood)
u <- thomas.estK(redwood, c(kappa=10, scale=0.1))
u
plot(u)
```

---

thomas.estpcf *Fit the Thomas Point Process by Minimum Contrast*

Description

Fits the Thomas point process to a point pattern dataset by the Method of Minimum Contrast using the pair correlation function.

Usage

```
thomas.estpcf(X, startpar=c(kappa=1,scale=1), lambda=NULL,
              q = 1/4, p = 2, rmin = NULL, rmax = NULL, ..., pcfargs=list())
```

Arguments

X	Data to which the Thomas model will be fitted. Either a point pattern or a summary statistic. See Details.
startpar	Vector of starting values for the parameters of the Thomas process.
lambda	Optional. An estimate of the intensity of the point process.
q,p	Optional. Exponents for the contrast criterion.
rmin, rmax	Optional. The interval of r values for the contrast criterion.
...	Optional arguments passed to optim to control the optimisation algorithm. See Details.
pcfargs	Optional list containing arguments passed to pcf.ppp to control the smoothing in the estimation of the pair correlation function.

Details

This algorithm fits the Thomas point process model to a point pattern dataset by the Method of Minimum Contrast, using the pair correlation function pcf.

The argument X can be either

a point pattern: An object of class "ppp" representing a point pattern dataset. The pair correlation function of the point pattern will be computed using pcf, and the method of minimum contrast will be applied to this.

a summary statistic: An object of class "fv" containing the values of a summary statistic, computed for a point pattern dataset. The summary statistic should be the pair correlation function, and this object should have been obtained by a call to pcf or one of its relatives.

The algorithm fits the Thomas point process to X, by finding the parameters of the Thomas model which give the closest match between the theoretical pair correlation function of the Thomas process and the observed pair correlation function. For a more detailed explanation of the Method of Minimum Contrast, see mincontrast.

The Thomas point process is described in Møller and Waagepetersen (2003, pp. 61–62). It is a cluster process formed by taking a pattern of parent points, generated according to a Poisson process with intensity κ, and around each parent point, generating a random number of offspring points, such that the number of offspring of each parent is a Poisson random variable with mean μ, and the locations of the offspring points of one parent are independent and isotropically Normally distributed around the parent point with standard deviation σ which is equal to the parameter scale. The named vector of stating values can use either sigma2 (σ^2) or scale as the name of the second component, but the latter is recommended for consistency with other cluster models.

The theoretical pair correlation function of the Thomas process is

$$g(r) = 1 + \frac{1}{4\pi\kappa\sigma^2} \exp(-\frac{r^2}{4\sigma^2})).$$

The theoretical intensity of the Thomas process is $\lambda = \kappa\mu$.

In this algorithm, the Method of Minimum Contrast is first used to find optimal values of the parameters κ and σ^2. Then the remaining parameter μ is inferred from the estimated intensity λ.

If the argument lambda is provided, then this is used as the value of λ. Otherwise, if X is a point pattern, then λ will be estimated from X. If X is a summary statistic and lambda is missing, then the intensity λ cannot be estimated, and the parameter μ will be returned as NA.

The remaining arguments rmin,rmax,q,p control the method of minimum contrast; see mincontrast.

The Thomas process can be simulated, using rThomas.

Homogeneous or inhomogeneous Thomas process models can also be fitted using the function kppm.

The optimisation algorithm can be controlled through the additional arguments "..." which are passed to the optimisation function optim. For example, to constrain the parameter values to a certain range, use the argument method="L-BFGS-B" to select an optimisation algorithm that respects box constraints, and use the arguments lower and upper to specify (vectors of) minimum and maximum values for each parameter.

Value

An object of class "minconfit". There are methods for printing and plotting this object. It contains the following main components:

par	Vector of fitted parameter values.
fit	Function value table (object of class "fv") containing the observed values of the summary statistic (observed) and the theoretical values of the summary statistic computed from the fitted model parameters.

Author(s)

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

References

Diggle, P. J., Besag, J. and Gleaves, J. T. (1976) Statistical analysis of spatial point patterns by means of distance methods. *Biometrics* **32** 659–667.

Møller, J. and Waagepetersen, R. (2003). Statistical Inference and Simulation for Spatial Point Processes. Chapman and Hall/CRC, Boca Raton.

Thomas, M. (1949) A generalisation of Poisson's binomial limit for use in ecology. *Biometrika* **36**, 18–25.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

thomas.estK mincontrast, pcf, rThomas to simulate the fitted model.

Examples

```
data(redwood)
u <- thomas.estpcf(redwood, c(kappa=10, scale=0.1))
u
plot(u, legendpos="topright")
u2 <- thomas.estpcf(redwood, c(kappa=10, scale=0.1),
      pcfargs=list(stoyan=0.12))
```

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)
thresholdCI(gold, distfault, nsim=100)
```

`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 numerical value giving the selected threshold. The result also belongs to the class "bw.optim" which can be plotted (the plot shows 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 <r.turner@auckland.ac.nz>

See Also

im

Examples

```
Z <- bei.extra$elev
plot(transect.im(Z))
```

`triplet.family` *Triplet Interaction Family*

Description

An object describing the family of all Gibbs point processes with interaction order equal to 3.

Details

Advanced Use Only!

This structure would not normally be touched by the user. It describes the interaction structure of Gibbs point processes which have infinite order of interaction, such as the triplet interaction process *Triplets*.

Anyway, `triplet.family` is an object of class `"isf"` containing a function `triplet.family$eval` for evaluating the sufficient statistics of a Gibbs point process model taking an exponential family form.

Value

Object of class `"isf"`, see `isf.object`.

Author(s)

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

and Rolf Turner `<r.turner@auckland.ac.nz>`

References

Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.

See Also

`Triplets` to create the triplet interaction process structure.

Other families: `pairwise.family`, `pairsat.family`, `inforder.family`, `ord.family`.

`Triplets` *The Triplet Point Process Model*

Description

Creates an instance of Geyer's triplet interaction point process model which can then be fitted to point pattern data.

Usage

```
Triplets(r)
```

Arguments

`r`	The interaction radius of the Triplets process

Details

The (stationary) Geyer triplet process (Geyer, 1999) with interaction radius r and parameters β and γ is the point process in which each point contributes a factor β to the probability density of the point pattern, and each triplet of close points contributes a factor γ to the density. A triplet of close points is a group of 3 points, each pair of which is closer than r units apart.

Thus the probability density is

$$f(x_1, \ldots, x_n) = \alpha \beta^{n(x)} \gamma^{s(x)}$$

where $x_1, \ldots, x_n$ represent the points of the pattern, $n(x)$ is the number of points in the pattern, $s(x)$ is the number of unordered triples of points that are closer than r units apart, and α is the normalising constant.

The interaction parameter γ must be less than or equal to 1 so that this model describes an "ordered" or "inhibitive" pattern.

The nonstationary Triplets process is similar except that the contribution of each individual point x_i is a function $\beta(x_i)$ of location, rather than a constant beta.

The function `ppm()`, which fits point process models to point pattern data, requires an argument of class `"interact"` describing the interpoint interaction structure of the model to be fitted. The appropriate description of the Triplets process pairwise interaction is yielded by the function `Triplets()`. See the examples below.

Note the only argument is the interaction radius `r`. When `r` is fixed, the model becomes an exponential family. The canonical parameters $\log(\beta)$ and $\log(\gamma)$ are estimated by `ppm()`, not fixed in `Triplets()`.

Value

An object of class `"interact"` describing the interpoint interaction structure of the Triplets process with interaction radius r.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

References

Geyer, C.J. (1999) Likelihood Inference for Spatial Point Processes. Chapter 3 in O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. Van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*, Chapman and Hall / CRC, Monographs on Statistics and Applied Probability, number 80. Pages 79–140.

See Also

ppm, triplet.family, ppm.object

Examples

```
Triplets(r=0.1)
# prints a sensible description of itself

ppm(cells ~1, Triplets(r=0.2))
# fit the stationary Triplets process to `cells'

# ppm(cells ~polynom(x,y,3), Triplets(r=0.2))
# fit a nonstationary Triplets process with log-cubic polynomial trend
```

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))
```

unitname *Name for Unit of Length*

Description

Inspect or change the name of the unit of length in a spatial dataset.

Usage

```
## S3 method for class 'dppm'
unitname(x)
## S3 method for class 'kppm'
unitname(x)
## S3 method for class 'minconfit'
unitname(x)
## S3 method for class 'ppm'
unitname(x)
## S3 method for class 'slrm'
unitname(x)
## S3 replacement method for class 'dppm'
unitname(x) <- value
## S3 replacement method for class 'kppm'
unitname(x) <- value
## S3 replacement method for class 'minconfit'
unitname(x) <- value
## S3 replacement method for class 'ppm'
unitname(x) <- value
## S3 replacement method for class 'slrm'
unitname(x) <- value
```

Arguments

x	A spatial dataset. Either a point pattern (object of class "ppp"), a line segment pattern (object of class "psp"), a window (object of class "owin"), a pixel image (object of class "im"), a tessellation (object of class "tess"), a quadrature scheme (object of class "quad"), or a fitted point process model (object of class "ppm" or "kppm" or "slrm" or "dppm" or "minconfit").
value	Name of the unit of length. See Details.

Details

Spatial datasets in the **spatstat** package may include the name of the unit of length. This name is used when printing or plotting the dataset, and in some other applications.

`unitname(x)` extracts this name, and `unitname(x) <- value` sets the name to `value`.

A valid name is either

- a single character string

- a vector of two character strings giving the singular and plural forms of the unit name
- a list of length 3, containing two character strings giving the singular and plural forms of the basic unit, and a number specifying the multiple of this unit.

Note that re-setting the name of the unit of length *does not* affect the numerical values in `x`. It changes only the string containing the name of the unit of length. To rescale the numerical values, use `rescale`.

Value

The return value of `unitname` is an object of class `"unitname"` containing the name of the unit of length in `x`. There are methods for `print`, `summary`, `as.character`, `rescale` and `compatible`.

Author(s)

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

See Also

`rescale`, `owin`, `ppp`

Examples

```
X <- runifrect(20)

# if the unit of length is 1 metre:
unitname(X) <- c("metre", "metres")

# if the unit of length is 6 inches:
unitname(X) <- list("inch", "inches", 6)
```

---

unstack.msr *Separate a Vector Measure into its Scalar Components*

Description

Converts a vector-valued measure into a list of scalar-valued measures.

Usage

```
## S3 method for class 'msr'
unstack(x, ...)
```

Arguments

`x`	A measure (object of class `"msr"`).
`...`	Ignored.

Details

This is a method for the generic unstack for the class "msr" of measures.

If x is a vector-valued measure, then y <- unstack(x) is a list of scalar-valued measures defined by the components of x. The jth entry of the list, y[[j]], is equivalent to the jth component of the vector measure x.

If x is a scalar-valued measure, then the result is a list consisting of one entry, which is x.

Value

A list of measures, of class "solist".

Author(s)

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

See Also

unstack

unstack.ppp

split.msr.

Examples

```
fit <- ppm(cells ~ x)
m <- residuals(fit, type="score")
m
unstack(m)
```

---

update.detpointprocfamily

Set Parameter Values in a Determinantal Point Process Model

Description

Set parameter values in a determinantal point process model object.

Usage

```
  ## S3 method for class 'detpointprocfamily'
update(object, ...)
```

Arguments

object	object of class "detpointprocfamily".
...	arguments of the form tag=value specifying the parameters values to set.

Value

Another object of class "detpointprocfamily".

Author(s)

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

---

update.interact *Update an Interpoint Interaction*

Description

This command updates the object using the arguments given.

Usage

```
## S3 method for class 'interact'
update(object, ...)
```

Arguments

object	Interpoint interaction (object of class "interact").
...	Additional or replacement values of parameters of object.

Details

This is a method for the generic function update for the class "interact" of interpoint interactions. It updates the object using the parameters given in the extra arguments ....

The extra arguments must be given in the form name=value and must be recognisable to the interaction object. They override any parameters of the same name in object.

Value

Another object of class "interact", equivalent to object except for changes in parameter values.

Author(s)

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

See Also

update.ppm

Examples

```
Str <- Strauss(r=1)
Str
update(Str, r=2)

M <- MultiStrauss(radii=matrix(1,2,2))
update(M, types=c("on", "off"))
```

---

update.kppm *Update a Fitted Cluster Point Process Model*

Description

update method for class "kppm".

Usage

```
 ## S3 method for class 'kppm'
update(object, ..., evaluate=TRUE,
                      envir=environment(terms(object)))
```

Arguments

object	Fitted cluster point process model. An object of class "kppm", obtained from kppm.
...	Arguments passed to kppm.
evaluate	Logical value indicating whether to return the updated fitted model (evaluate=TRUE, the default) or just the updated call to kppm (evaluate=FALSE).
envir	Environment in which to re-evaluate the call to ppm.

Details

object should be a fitted cluster point process model, obtained from the model-fitting function kppm. The model will be updated according to the new arguments provided.

If the argument trend is provided, it determines the intensity in the updated model. It should be an R formula (with or without a left hand side). It may include the symbols + or - to specify addition or deletion of terms in the current model formula, as shown in the Examples below. The symbol . refers to the current contents of the formula.

The intensity in the updated model is determined by the argument trend if it is provided, or otherwise by any unnamed argument that is a formula, or otherwise by the formula of the original model, formula(object).

The spatial point pattern data to which the new model is fitted is determined by the left hand side of the updated model formula, if this is present. Otherwise it is determined by the argument X if it is provided, or otherwise by any unnamed argument that is a point pattern or a quadrature scheme.

The model is refitted using kppm.

Value

Another fitted cluster point process model (object of class "kppm".

Author(s)

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

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

kppm, plot.kppm, predict.kppm, simulate.kppm, methods.kppm, vcov.kppm

Examples

```
fit <- kppm(redwood ~1, "Thomas")
fitx <- update(fit, ~ . + x)
fitM <- update(fit, clusters="MatClust")
fitC <- update(fit, cells)
fitCx <- update(fit, cells ~ x)
```

---

update.ppm *Update a Fitted Point Process Model*

Description

update method for class "ppm".

Usage

```
  ## S3 method for class 'ppm'
update(object, ..., fixdummy=TRUE, use.internal=NULL,
                                    envir=environment(terms(object)))
```

Arguments

object	An existing fitted point process model, typically produced by ppm.
...	Arguments to be updated in the new call to ppm.
fixdummy	Logical flag indicating whether the quadrature scheme for the call to ppm should use the same set of dummy points as that in the original call.
use.internal	Optional. Logical flag indicating whether the model should be refitted using the internally saved data (use.internal=TRUE) or by re-evaluating these data in the current frame (use.internal=FALSE).
envir	Environment in which to re-evaluate the call to ppm.

Details

This is a method for the generic function update for the class "ppm". An object of class "ppm" describes a fitted point process model. See ppm.object) for details of this class.

update.ppm will modify the point process model specified by object according to the new arguments given, then re-fit it. The actual re-fitting is performed by the model-fitting function ppm.

If you are comparing several model fits to the same data, or fits of the same model to different data, it is strongly advisable to use update.ppm rather than trying to fit them by hand. This is because update.ppm re-fits the model in a way which is comparable to the original fit.

The arguments ... are matched to the formal arguments of ppm as follows.

First, all the *named* arguments in ... are matched with the formal arguments of ppm. Use name=NULL to remove the argument name from the call.

Second, any *unnamed* arguments in ... are matched with formal arguments of ppm if the matching is obvious from the class of the object. Thus ... may contain

- exactly one argument of class "ppp" or "quad", which will be interpreted as the named argument Q;
- exactly one argument of class "formula", which will be interpreted as the named argument trend (or as specifying a change to the trend formula);
- exactly one argument of class "interact", which will be interpreted as the named argument interaction;
- exactly one argument of class "data.frame", which will be interpreted as the named argument covariates.

The trend argument can be a formula that specifies a *change* to the current trend formula. For example, the formula ~ . + Z specifies that the additional covariate Z will be added to the right hand side of the trend formula in the existing object.

The argument fixdummy=TRUE ensures comparability of the objects before and after updating. When fixdummy=FALSE, calling update.ppm is exactly the same as calling ppm with the updated arguments. However, the original and updated models are not strictly comparable (for example, their pseudolikelihoods are not strictly comparable) unless they used the same set of dummy points for the quadrature scheme. Setting fixdummy=TRUE ensures that the re-fitting will be performed using the same set of dummy points. This is highly recommended.

The value of use.internal determines where to find data to re-evaluate the model (data for the arguments mentioned in the original call to ppm that are not overwritten by arguments to update.ppm).

If use.internal=FALSE, then arguments to ppm are *re-evaluated* in the frame where you call update.ppm. This is like the behaviour of the other methods for update. This means that if you have changed any of the objects referred to in the call, these changes will be taken into account. Also if the original call to ppm included any calls to random number generators, these calls will be recomputed, so that you will get a different outcome of the random numbers.

If use.internal=TRUE, then arguments to ppm are extracted from internal data stored inside the current fitted model object. This is useful if you don't want to re-evaluate anything. It is also necessary if if object has been restored from a dump file using load or source. In such cases, we have lost the environment in which object was fitted, and data cannot be re-evaluated.

By default, if use.internal is missing, update.ppm will re-evaluate the arguments if this is possible, and use internal data if not.

Value

Another fitted point process model (object of class "ppm").

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

Examples

```
data(nztrees)
data(cells)

# fit the stationary Poisson process
fit <- ppm(nztrees, ~ 1)

# fit a nonstationary Poisson process
fitP <- update(fit, trend=~x)
fitP <- update(fit, ~x)

# change the trend formula: add another term to the trend
fitPxy <- update(fitP, ~ . + y)
# change the trend formula: remove the x variable
fitPy <- update(fitPxy, ~ . - x)

# fit a stationary Strauss process
fitS <- update(fit, interaction=Strauss(13))
fitS <- update(fit, Strauss(13))

# refit using a different edge correction
fitS <- update(fitS, correction="isotropic")

# re-fit the model to a subset
# of the original point pattern
nzw <- owin(c(0,148),c(0,95))
nzsub <- nztrees[,nzw]
fut <- update(fitS, Q=nzsub)
fut <- update(fitS, nzsub)

# WARNING: the point pattern argument is called 'Q'

ranfit <- ppm(rpoispp(42), ~1, Poisson())
ranfit
# different random data!
update(ranfit)
# the original data
update(ranfit, use.internal=TRUE)
```

valid *Check Whether Point Process Model is Valid*

Description

Determines whether a point process model object corresponds to a valid point process.

Usage

```
valid(object, ...)
```

Arguments

object	Object of some class, describing a point process model.
...	Additional arguments passed to methods.

Details

The function valid is generic, with methods for the classes "ppm" and "dppmodel".

An object representing a point process is called valid if all its parameter values are known (for example, no parameter takes the value NA or NaN) and the parameter values correspond to a well-defined point process (for example, the parameter values satisfy all the constraints that are imposed by mathematical theory.)

See the methods for further details.

Value

A logical value, or NA.

Author(s)

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

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

valid.ppm, valid.detpointprocfamily

valid.detpointprocfamily *Check Validity of a Determinantal Point Process Model*

Description

Checks the validity of a determinantal point process model.

Usage

```
   ## S3 method for class 'detpointprocfamily'
valid(object, ...)
```

Arguments

object	Model of class "detpointprocfamily".
...	Ignored.

Value

Logical

Author(s)

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

Rolf Turner <r.turner@auckland.ac.nz>

and Ege Rubak <rubak@math.aau.dk>

See Also

valid

Examples

```
model1 <- dppMatern(lambda=100, alpha=.01, nu=1, d=2)
valid(model1)
model2 <- dppMatern(lambda=100, alpha=1, nu=1, d=2)
valid(model2)
```

valid.ppm *Check Whether Point Process Model is Valid*

Description

Determines whether a fitted point process model satisfies the integrability conditions for existence of the point process.

Usage

```
  ## S3 method for class 'ppm'
valid(object, warn=TRUE, ...)
```

Arguments

object	Fitted point process model (object of class "ppm").
warn	Logical value indicating whether to issue a warning if the validity of the model cannot be checked (due to unavailability of the required code).
...	Ignored.

Details

This is a method for the generic function valid for Poisson and Gibbs point process models (class "ppm").

The model-fitting function ppm fits Gibbs point process models to point pattern data. By default, ppm does not check whether the fitted model actually exists as a point process. This checking is done by valid.ppm.

Unlike a regression model, which is well-defined for any values of the fitted regression coefficients, a Gibbs point process model is only well-defined if the fitted interaction parameters satisfy some constraints. A famous example is the Strauss process (see Strauss) which exists only when the interaction parameter γ is less than or equal to 1. For values $\gamma > 1$, the probability density is not integrable and the process does not exist (and cannot be simulated).

By default, ppm does not enforce the constraint that a fitted Strauss process (for example) must satisfy $\gamma \leq 1$. This is because a fitted parameter value of $\gamma > 1$ could be useful information for data analysis, as it indicates that the Strauss model is not appropriate, and suggests a clustered model should be fitted.

The function valid.ppm checks whether the fitted model object specifies a well-defined point process. It returns TRUE if the model is well-defined.

Another possible reason for invalid models is that the data may not be adequate for estimation of the model parameters. In this case, some of the fitted coefficients could be NA or infinite values. If this happens then valid.ppm returns FALSE.

Use the function project.ppm to force the fitted model to be valid.

Value

A logical value, or NA.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz>

See Also

ppm, project.ppm

Examples

```
fit1 <- ppm(cells, ~1, Strauss(0.1))
valid(fit1)
fit2 <- ppm(redwood, ~1, Strauss(0.1))
valid(fit2)
```

---

valid.slrm *Check Whether Spatial Logistic Regression Model is Valid*

Description

Determines whether a fitted spatial logistic regression model is a well-defined model.

Usage

```
  ## S3 method for class 'slrm'
valid(object, warn=TRUE, ...)
```

Arguments

object	Fitted spatial logistic regression model (object of class "slrm").
warn	Logical value indicating whether to issue a warning if the validity of the model cannot be checked (due to unavailability of the required code).
...	Ignored.

Details

This is a method for the generic function valid for spatial logistic regression models (class "slrm").

In a model fitted by slrm, some of the fitted coefficients may be NA or infinite values. This can occur if the data are not adequate for estimation of the model parameters. The model is said to be *unidentifiable* or *confounded*.

The function valid.slrm checks whether the fitted coefficients of object specify a well-defined model. It returns TRUE if the model is well-defined, and FALSE otherwise.

Use the function emend.slrm to force the fitted model to be valid.

Value

A logical value, or NA.

Author(s)

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

See Also

slrm, emend.slrm

Examples

```
fit1 <- slrm(cells ~ x)
valid(fit1)
fit2 <- slrm(cells ~ x + I(x))
valid(fit2)
```

---

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 <r.turner@auckland.ac.nz>

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"))
```

---

varcount *Predicted Variance of the Number of Points*

Description

Given a fitted point process model, calculate the predicted variance of the number of points in a nominated set B.

Usage

```
varcount(model, B=Window(model), ..., dimyx = NULL)
```

Arguments

model	A fitted point process model (object of class "ppm", "kppm" or "dppm").
B	A window (object of class "owin" specifying the region in which the points are counted. Alternatively a pixel image (object of class "im") or a function of spatial coordinates specifying a numerical weight for each random point. The default is the window of the original point pattern data to which the model was fitted.
...	Additional arguments passed to B when it is a function.
dimyx	Spatial resolution for the calculations. Argument passed to as.mask.

Details

This command calculates the variance of the number of points falling in a specified window B according to the model. It can also calculate the variance of a sum of weights attached to each random point.

The model should be a fitted point process model (object of class "ppm", "kppm" or "dppm").

- If B is a window, this command calculates the variance of the number of points falling in B, according to the fitted model.

  If the model depends on spatial covariates other than the Cartesian coordinates, then B should be a subset of the domain in which these covariates are defined.

- If B is a pixel image, this command calculates the variance of $T = \sum_i B(x_i)$, the sum of the values of B over all random points falling in the domain of the image.

  If the model depends on spatial covariates other than the Cartesian coordinates, then the domain of the pixel image, as.owin(B), should be a subset of the domain in which these covariates are defined.
- If B is a function(x,y) or function(x,y,...) this command calculates the variance of $T = \sum_i B(x_i)$, the sum of the values of B over all random points falling inside the window W=as.owin(model), the window in which the original data were observed.

The variance calculation involves the intensity and the pair correlation function of the model. The calculation is exact (up to discretisation error) for models of class "kppm" and "dppm", and for Poisson point process models of class "ppm". For Gibbs point process models of class "ppm" the calculation depends on the Poisson-saddlepoint approximations to the intensity and pair correlation function, which are rough approximations. The approximation is not yet implemented for some Gibbs models.

Value

A single number.

Author(s)

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

See Also

predict.ppm, predict.kppm, predict.dppm

Examples

```
fitT <- kppm(redwood ~ 1, "Thomas")
B <- owin(c(0, 0.5), c(-0.5, 0))
varcount(fitT, B)

fitS <- ppm(swedishpines ~ 1, Strauss(9))
BS <- square(50)
varcount(fitS, BS)
```

---

vargamma.estK *Fit the Neyman-Scott Cluster Point Process with Variance Gamma kernel*

Description

Fits the Neyman-Scott cluster point process, with Variance Gamma kernel, to a point pattern dataset by the Method of Minimum Contrast.

Usage

```
vargamma.estK(X, startpar=c(kappa=1,scale=1), nu = -1/4, lambda=NULL,
            q = 1/4, p = 2, rmin = NULL, rmax = NULL, ...)
```

Arguments

X	Data to which the model will be fitted. Either a point pattern or a summary statistic. See Details.
startpar	Vector of starting values for the parameters of the model.
nu	Numerical value controlling the shape of the tail of the clusters. A number greater than -1/2.
lambda	Optional. An estimate of the intensity of the point process.
q,p	Optional. Exponents for the contrast criterion.
rmin, rmax	Optional. The interval of r values for the contrast criterion.
...	Optional arguments passed to optim to control the optimisation algorithm. See Details.

Details

This algorithm fits the Neyman-Scott Cluster point process model with Variance Gamma kernel (Jalilian et al, 2013) to a point pattern dataset by the Method of Minimum Contrast, using the K function.

The argument X can be either

a point pattern: An object of class "ppp" representing a point pattern dataset. The K function of the point pattern will be computed using Kest, and the method of minimum contrast will be applied to this.

a summary statistic: An object of class "fv" containing the values of a summary statistic, computed for a point pattern dataset. The summary statistic should be the K function, and this object should have been obtained by a call to Kest or one of its relatives.

The algorithm fits the Neyman-Scott Cluster point process with Variance Gamma kernel to X, by finding the parameters of the model which give the closest match between the theoretical K function of the model and the observed K function. For a more detailed explanation of the Method of Minimum Contrast, see mincontrast.

The Neyman-Scott cluster point process with Variance Gamma kernel is described in Jalilian et al (2013). It is a cluster process formed by taking a pattern of parent points, generated according to a Poisson process with intensity κ, and around each parent point, generating a random number of offspring points, such that the number of offspring of each parent is a Poisson random variable with mean μ, and the locations of the offspring points of one parent have a common distribution described in Jalilian et al (2013).

The shape of the kernel is determined by the dimensionless index nu. This is the parameter $\nu' = \alpha/2 - 1$ appearing in equation (12) on page 126 of Jalilian et al (2013). In previous versions of spatstat instead of specifying nu (called nu.ker at that time) the user could specify nu.pcf which is the parameter $\nu = \alpha - 1$ appearing in equation (13), page 127 of Jalilian et al (2013). These are related by nu.pcf = 2 * nu.ker + 1 and nu.ker = (nu.pcf - 1)/2. This syntax is still supported

but not recommended for consistency across the package. In that case exactly one of `nu.ker` or `nu.pcf` must be specified.

If the argument `lambda` is provided, then this is used as the value of the point process intensity λ. Otherwise, if `X` is a point pattern, then λ will be estimated from `X`. If `X` is a summary statistic and `lambda` is missing, then the intensity λ cannot be estimated, and the parameter μ will be returned as `NA`.

The remaining arguments `rmin,rmax,q,p` control the method of minimum contrast; see `mincontrast`.

The corresponding model can be simulated using `rVarGamma`.

The parameter `eta` appearing in `startpar` is equivalent to the scale parameter `omega` used in `rVarGamma`.

Homogeneous or inhomogeneous Neyman-Scott/VarGamma models can also be fitted using the function `kppm` and the fitted models can be simulated using `simulate.kppm`.

The optimisation algorithm can be controlled through the additional arguments `"..."` which are passed to the optimisation function `optim`. For example, to constrain the parameter values to a certain range, use the argument `method="L-BFGS-B"` to select an optimisation algorithm that respects box constraints, and use the arguments `lower` and `upper` to specify (vectors of) minimum and maximum values for each parameter.

Value

An object of class `"minconfit"`. There are methods for printing and plotting this object. It contains the following main components:

`par`	Vector of fitted parameter values.
`fit`	Function value table (object of class `"fv"`) containing the observed values of the summary statistic (`observed`) and the theoretical values of the summary statistic computed from the fitted model parameters.

Author(s)

Abdollah Jalilian and Rasmus Waagepetersen. Adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu.a

References

Jalilian, A., Guan, Y. and Waagepetersen, R. (2013) Decomposition of variance for spatial Cox processes. *Scandinavian Journal of Statistics* **40**, 119-137.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

`kppm`, `vargamma.estpcf`, `lgcp.estK`, `thomas.estK`, `cauchy.estK`, `mincontrast`, `Kest`, `Kmodel`.

`rVarGamma` to simulate the model.

Examples

```
if(interactive()) {
   u <- vargamma.estK(redwood)
   print(u)
   plot(u)
}
```

`vargamma.estpcf` *Fit the Neyman-Scott Cluster Point Process with Variance Gamma kernel*

Description

Fits the Neyman-Scott cluster point process, with Variance Gamma kernel, to a point pattern dataset by the Method of Minimum Contrast, using the pair correlation function.

Usage

```
vargamma.estpcf(X, startpar=c(kappa=1,scale=1), nu = -1/4, lambda=NULL,
            q = 1/4, p = 2, rmin = NULL, rmax = NULL,
            ..., pcfargs = list())
```

Arguments

`X`	Data to which the model will be fitted. Either a point pattern or a summary statistic. See Details.
`startpar`	Vector of starting values for the parameters of the model.
`nu`	Numerical value controlling the shape of the tail of the clusters. A number greater than `-1/2`.
`lambda`	Optional. An estimate of the intensity of the point process.
`q,p`	Optional. Exponents for the contrast criterion.
`rmin, rmax`	Optional. The interval of r values for the contrast criterion.
`...`	Optional arguments passed to `optim` to control the optimisation algorithm. See Details.
`pcfargs`	Optional list containing arguments passed to `pcf.ppp` to control the smoothing in the estimation of the pair correlation function.

Details

This algorithm fits the Neyman-Scott Cluster point process model with Variance Gamma kernel (Jalilian et al, 2013) to a point pattern dataset by the Method of Minimum Contrast, using the pair correlation function.

The argument `X` can be either

a point pattern: An object of class "ppp" representing a point pattern dataset. The pair correlation function of the point pattern will be computed using pcf, and the method of minimum contrast will be applied to this.

a summary statistic: An object of class "fv" containing the values of a summary statistic, computed for a point pattern dataset. The summary statistic should be the pair correlation function, and this object should have been obtained by a call to pcf or one of its relatives.

The algorithm fits the Neyman-Scott Cluster point process with Variance Gamma kernel to X, by finding the parameters of the model which give the closest match between the theoretical pair correlation function of the model and the observed pair correlation function. For a more detailed explanation of the Method of Minimum Contrast, see mincontrast.

The Neyman-Scott cluster point process with Variance Gamma kernel is described in Jalilian et al (2013). It is a cluster process formed by taking a pattern of parent points, generated according to a Poisson process with intensity κ, and around each parent point, generating a random number of offspring points, such that the number of offspring of each parent is a Poisson random variable with mean μ, and the locations of the offspring points of one parent have a common distribution described in Jalilian et al (2013).

The shape of the kernel is determined by the dimensionless index nu. This is the parameter $\nu' = \alpha/2 - 1$ appearing in equation (12) on page 126 of Jalilian et al (2013). In previous versions of spatstat instead of specifying nu (called nu.ker at that time) the user could specify nu.pcf which is the parameter $\nu = \alpha - 1$ appearing in equation (13), page 127 of Jalilian et al (2013). These are related by nu.pcf = 2 * nu.ker + 1 and nu.ker = (nu.pcf - 1)/2. This syntax is still supported but not recommended for consistency across the package. In that case exactly one of nu.ker or nu.pcf must be specified.

If the argument lambda is provided, then this is used as the value of the point process intensity λ. Otherwise, if X is a point pattern, then λ will be estimated from X. If X is a summary statistic and lambda is missing, then the intensity λ cannot be estimated, and the parameter μ will be returned as NA.

The remaining arguments rmin,rmax,q,p control the method of minimum contrast; see mincontrast.

The corresponding model can be simulated using rVarGamma.

The parameter eta appearing in startpar is equivalent to the scale parameter omega used in rVarGamma.

Homogeneous or inhomogeneous Neyman-Scott/VarGamma models can also be fitted using the function kppm and the fitted models can be simulated using simulate.kppm.

The optimisation algorithm can be controlled through the additional arguments "..." which are passed to the optimisation function optim. For example, to constrain the parameter values to a certain range, use the argument method="L-BFGS-B" to select an optimisation algorithm that respects box constraints, and use the arguments lower and upper to specify (vectors of) minimum and maximum values for each parameter.

Value

An object of class "minconfit". There are methods for printing and plotting this object. It contains the following main components:

par	Vector of fitted parameter values.

`fit` Function value table (object of class `"fv"`) containing the observed values of the summary statistic (`observed`) and the theoretical values of the summary statistic computed from the fitted model parameters.

Author(s)

Abdollah Jalilian and Rasmus Waagepetersen. Adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu.a

References

Jalilian, A., Guan, Y. and Waagepetersen, R. (2013) Decomposition of variance for spatial Cox processes. *Scandinavian Journal of Statistics* **40**, 119-137.

Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

`kppm`, `vargamma.estK`, `lgcp.estpcf`, `thomas.estpcf`, `cauchy.estpcf`, `mincontrast`, `pcf`, `pcfmodel`.

`rVarGamma` to simulate the model.

Examples

```
u <- vargamma.estpcf(redwood)
u
plot(u, legendpos="topright")
```

---

vcov.kppm *Variance-Covariance Matrix for a Fitted Cluster Point Process Model*

Description

Returns the variance-covariance matrix of the estimates of the parameters of a fitted cluster point process model.

Usage

```
   ## S3 method for class 'kppm'
vcov(object, ...,
          what=c("vcov", "corr", "fisher", "internals"),
          fast = NULL, rmax = NULL, eps.rmax = 0.01,
          verbose = TRUE)
```

Arguments

object	A fitted cluster point process model (an object of class "kppm".)
...	Ignored.
what	Character string (partially-matched) that specifies what matrix is returned. Options are "vcov" for the variance-covariance matrix, "corr" for the correlation matrix, and "fisher" for the Fisher information matrix.
fast	Logical specifying whether tapering (using sparse matrices from **Matrix**) should be used to speed up calculations. Warning: This is expected to underestimate the true asymptotic variances/covariances.
rmax	Optional. The dependence range. Not usually specified by the user. Only used when fast=TRUE.
eps.rmax	Numeric. A small positive number which is used to determine rmax from the tail behaviour of the pair correlation function when fast option (fast=TRUE) is used. Namely rmax is the smallest value of r at which $(g(r) - 1)/(g(0) - 1)$ falls below eps.rmax. Only used when fast=TRUE. Ignored if rmax is provided.
verbose	Logical value indicating whether to print progress reports during very long calculations.

Details

This function computes the asymptotic variance-covariance matrix of the estimates of the canonical (regression) parameters in the cluster point process model object. It is a method for the generic function vcov.

The result is an n * n matrix where n = length(coef(model)).

To calculate a confidence interval for a regression parameter, use confint as shown in the examples.

Value

A square matrix.

Author(s)

Abdollah Jalilian and Rasmus Waagepetersen. Ported to **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Ege Rubak <rubak@math.aau.dk>.

References

Waagepetersen, R. (2007) Estimating functions for inhomogeneous spatial point processes with incomplete covariate data. *Biometrika* **95**, 351–363.

See Also

kppm, vcov, vcov.ppm

Examples

```
fit <- kppm(redwood ~ x + y)
vcov(fit)
vcov(fit, what="corr")

# confidence interval
confint(fit)
# cross-check the confidence interval by hand:
sd <- sqrt(diag(vcov(fit)))
t(coef(fit) + 1.96 * outer(sd, c(lower=-1, upper=1)))
```

`vcov.mppm` *Calculate Variance-Covariance Matrix for Fitted Multiple Point Process Model*

Description

Given a fitted multiple point process model, calculate the variance-covariance matrix of the parameter estimates.

Usage

```
   ## S3 method for class 'mppm'
vcov(object, ..., what="vcov", err="fatal")
```

Arguments

`object`	A multiple point process model (object of class `"mppm"`).
`...`	Arguments recognised by `vcov.ppm`.
`what`	Character string indicating which quantity should be calculated. Options include `"vcov"` for the variance-covariance matrix, `"corr"` for the correlation matrix, and `"fisher"` for the Fisher information matrix.
`err`	Character string indicating what action to take if an error occurs. Either `"fatal"`, `"warn"` or `"null"`.

Details

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

The argument `object` should be a fitted multiple point process model (object of class `"mppm"`) generated by `mppm`.

The variance-covariance matrix of the parameter estimates is computed using asymptotic theory for maximum likelihood (for Poisson processes) or estimating equations (for other Gibbs models).

If `what="vcov"` (the default), the variance-covariance matrix is returned. If `what="corr"`, the variance-covariance matrix is normalised to yield a correlation matrix, and this is returned. If `what="fisher"`, the Fisher information matrix is returned instead.

In all three cases, the rows and columns of the matrix correspond to the parameters (coefficients) in the same order as in `coef{model}`.

If errors or numerical problems occur, the argument `err` determines what will happen. If `err="fatal"` an error will occur. If `err="warn"` a warning will be issued and `NA` will be returned. If `err="null"`, no warning is issued, but `NULL` is returned.

Value

A numeric matrix (or `NA` or `NULL`).

Error messages

An error message that reports *system is computationally singular* indicates that the determinant of the Fisher information matrix of one of the models was either too large or too small for reliable numerical calculation. See `vcov.ppm` for suggestions on how to handle this.

Author(s)

Adrian Baddeley, Ida-Maria Sintorn and Leanne Bischoff. Implemented by Adrian Baddeley <Adrian.Baddeley@curtin.ed
Rolf Turner <r.turner@auckland.ac.nz> and Ege Rubak <rubak@math.aau.dk>.

References

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

See Also

`vcov`, `vcov.ppm`, `mppm`

Examples

```
fit <- mppm(Wat ~x, data=hyperframe(Wat=waterstriders))
vcov(fit)
```

`vcov.ppm` *Variance-Covariance Matrix for a Fitted Point Process Model*

Description

Returns the variance-covariance matrix of the estimates of the parameters of a fitted point process model.

Usage

```
  ## S3 method for class 'ppm'
vcov(object, ..., what = "vcov", verbose = TRUE,
                  fine=FALSE,
                  gam.action=c("warn", "fatal", "silent"),
                  matrix.action=c("warn", "fatal", "silent"),
                  logi.action=c("warn", "fatal", "silent"),
                  nacoef.action=c("warn", "fatal", "silent"),
                  hessian=FALSE)
```

Arguments

object	A fitted point process model (an object of class "ppm".)
...	Ignored.
what	Character string (partially-matched) that specifies what matrix is returned. Options are "vcov" for the variance-covariance matrix, "corr" for the correlation matrix, and "fisher" or "Fisher" for the Fisher information matrix.
fine	Logical value indicating whether to use a quick estimate (fine=FALSE, the default) or a slower, more accurate estimate (fine=TRUE).
verbose	Logical. If TRUE, a message will be printed if various minor problems are encountered.
gam.action	String indicating what to do if object was fitted by gam.
matrix.action	String indicating what to do if the matrix is ill-conditioned (so that its inverse cannot be calculated).
logi.action	String indicating what to do if object was fitted via the logistic regression approximation using a non-standard dummy point process.
nacoef.action	String indicating what to do if some of the fitted coefficients are NA (so that variance cannot be calculated).
hessian	Logical. Use the negative Hessian matrix of the log pseudolikelihood instead of the Fisher information.

Details

This function computes the asymptotic variance-covariance matrix of the estimates of the canonical parameters in the point process model object. It is a method for the generic function vcov.

object should be an object of class "ppm", typically produced by ppm.

The canonical parameters of the fitted model object are the quantities returned by coef.ppm(object). The function vcov calculates the variance-covariance matrix for these parameters.

The argument what provides three options:

what="vcov" return the variance-covariance matrix of the parameter estimates

what="corr" return the correlation matrix of the parameter estimates

what="fisher" return the observed Fisher information matrix.

In all three cases, the result is a square matrix. The rows and columns of the matrix correspond to the canonical parameters given by `coef.ppm(object)`. The row and column names of the matrix are also identical to the names in `coef.ppm(object)`.

For models fitted by the Berman-Turner approximation (Berman and Turner, 1992; Baddeley and Turner, 2000) to the maximum pseudolikelihood (using the default `method="mpl"` in the call to `ppm`), the implementation works as follows.

- If the fitted model `object` is a Poisson process, the calculations are based on standard asymptotic theory for the maximum likelihood estimator (Kutoyants, 1998). The observed Fisher information matrix of the fitted model `object` is first computed, by summing over the Berman-Turner quadrature points in the fitted model. The asymptotic variance-covariance matrix is calculated as the inverse of the observed Fisher information. The correlation matrix is then obtained by normalising.
- If the fitted model is not a Poisson process (i.e. it is some other Gibbs point process) then the calculations are based on Coeurjolly and Rubak (2012). A consistent estimator of the variance-covariance matrix is computed by summing terms over all pairs of data points. If required, the Fisher information is calculated as the inverse of the variance-covariance matrix.

For models fitted by the Huang-Ogata method (`method="ho"` in the call to `ppm`), the implementation uses the Monte Carlo estimate of the Fisher information matrix that was computed when the original model was fitted.

For models fitted by the logistic regression approximation to the maximum pseudolikelihood (`method="logi"` in the call to `ppm`), calculations are based on (Baddeley et al., 2013). A consistent estimator of the variance-covariance matrix is computed by summing terms over all pairs of data points. If required, the Fisher information is calculated as the inverse of the variance-covariance matrix. In this case the calculations depend on the type of dummy pattern used, and currently only the types `"stratrand"`, `"binomial"` and `"poisson"` as generated by `quadscheme.logi` are implemented. For other types the behavior depends on the argument `logi.action`. If `logi.action="fatal"` an error is produced. Otherwise, for types `"grid"` and `"transgrid"` the formulas for `"stratrand"` are used which in many cases should be conservative. For an arbitrary user specified dummy pattern (type `"given"`) the formulas for `"poisson"` are used which in many cases should be conservative. If `logi.action="warn"` a warning is issued otherwise the calculation proceeds without a warning.

The argument `verbose` makes it possible to suppress some diagnostic messages.

The asymptotic theory is not correct if the model was fitted using `gam` (by calling `ppm` with `use.gam=TRUE`). The argument `gam.action` determines what to do in this case. If `gam.action="fatal"`, an error is generated. If `gam.action="warn"`, a warning is issued and the calculation proceeds using the incorrect theory for the parametric case, which is probably a reasonable approximation in many applications. If `gam.action="silent"`, the calculation proceeds without a warning.

If `hessian=TRUE` then the negative Hessian (second derivative) matrix of the log pseudolikelihood, and its inverse, will be computed. For non-Poisson models, this is not a valid estimate of variance, but is useful for other calculations.

Note that standard errors and 95% confidence intervals for the coefficients can also be obtained using `confint(object)` or `coef(summary(object))`.

Value

A square matrix.

Error messages

An error message that reports *system is computationally singular* indicates that the determinant of the Fisher information matrix was either too large or too small for reliable numerical calculation.

If this message occurs, try repeating the calculation using `fine=TRUE`.

Singularity can occur because of numerical overflow or collinearity in the covariates. To check this, rescale the coordinates of the data points and refit the model. See the Examples.

In a Gibbs model, a singular matrix may also occur if the fitted model is a hard core process: this is a feature of the variance estimator.

Author(s)

Original code for Poisson point process was written by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

and Rolf Turner <r.turner@auckland.ac.nz> . New code for stationary Gibbs point processes was generously contributed by Ege Rubak <rubak@math.aau.dk> and Jean-Francois Coeurjolly. New code for generic Gibbs process written by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>. New code for logistic method contributed by Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A., Coeurjolly, J.-F., Rubak, E. and Waagepetersen, R. (2014) Logistic regression for spatial Gibbs point processes. *Biometrika* **101** (2) 377–392.

Coeurjolly, J.-F. and Rubak, E. (2013) Fast covariance estimation for innovations computed from a spatial Gibbs point process. Scandinavian Journal of Statistics **40** 669–684.

Kutoyants, Y.A. (1998) **Statistical Inference for Spatial Poisson Processes**, Lecture Notes in Statistics 134. New York: Springer 1998.

See Also

`vcov` for the generic,

`ppm` for information about fitted models,

`confint` for confidence intervals.

Examples

```
  X <- rpoispp(42)
  fit <- ppm(X, ~ x + y)
  vcov(fit)
  vcov(fit, what="Fish")

  # example of singular system
  m <- ppm(demopat ~polynom(x,y,2))
  ## Not run:
    try(v <- vcov(m))

## End(Not run)
  # rescale x, y coordinates to range [0,1] x [0,1] approximately
  demopatScale <- rescale(demopat, 10000)
```

```
m <- ppm(demopatScale ~ polynom(x,y,2))
v <- vcov(m)

# Gibbs example
fitS <- ppm(swedishpines ~1, Strauss(9))
coef(fitS)
sqrt(diag(vcov(fitS)))
```

`vcov.slrm` *Variance-Covariance Matrix for a Fitted Spatial Logistic Regression*

Description

Returns the variance-covariance matrix of the estimates of the parameters of a point process model that was fitted by spatial logistic regression.

Usage

```
  ## S3 method for class 'slrm'
vcov(object, ...,
         what=c("vcov", "corr", "fisher", "Fisher"))
```

Arguments

`object`	A fitted point process model of class `"slrm"`.
`...`	Ignored.
`what`	Character string (partially-matched) that specifies what matrix is returned. Options are `"vcov"` for the variance-covariance matrix, `"corr"` for the correlation matrix, and `"fisher"` or `"Fisher"` for the Fisher information matrix.

Details

This function computes the asymptotic variance-covariance matrix of the estimates of the canonical parameters in the point process model `object`. It is a method for the generic function `vcov`.

`object` should be an object of class `"slrm"`, typically produced by `slrm`. It represents a Poisson point process model fitted by spatial logistic regression.

The canonical parameters of the fitted model `object` are the quantities returned by `coef.slrm(object)`. The function `vcov` calculates the variance-covariance matrix for these parameters.

The argument `what` provides three options:

`what="vcov"` return the variance-covariance matrix of the parameter estimates

`what="corr"` return the correlation matrix of the parameter estimates

`what="fisher"` return the observed Fisher information matrix.

In all three cases, the result is a square matrix. The rows and columns of the matrix correspond to the canonical parameters given by `coef.slrm(object)`. The row and column names of the matrix are also identical to the names in `coef.slrm(object)`.

Note that standard errors and 95% confidence intervals for the coefficients can also be obtained using `confint(object)` or `coef(summary(object))`.

Standard errors for the fitted intensity can be obtained using `predict.slrm`.

Value

A square matrix.

Error messages

An error message that reports *system is computationally singular* indicates that the determinant of the Fisher information matrix was either too large or too small for reliable numerical calculation. This can occur because of numerical overflow or collinearity in the covariates.

Author(s)

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

and Rolf Turner <r.turner@auckland.ac.nz> .

References

Baddeley, A., Berman, M., Fisher, N.I., Hardegen, A., Milne, R.K., Schuhmacher, D., Shah, R. and Turner, R. (2010) Spatial logistic regression and change-of-support for spatial Poisson point processes. *Electronic Journal of Statistics* **4**, 1151–1201. DOI: 10.1214/10-EJS581

See Also

`vcov` for the generic,

`slrm` for information about fitted models,

`predict.slrm` for other kinds of calculation about the model,

`confint` for confidence intervals.

Examples

```
X <- rpoispp(42)
fit <- slrm(X ~ x + y)
vcov(fit)
vcov(fit, what="corr")
vcov(fit, what="f")
```

Window.ppm *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 'ppm'
Window(X, ..., from=c("points", "covariates"))

 ## S3 method for class 'kppm'
Window(X, ..., from=c("points", "covariates"))

 ## S3 method for class 'dppm'
Window(X, ..., from=c("points", "covariates"))

 ## S3 method for class 'slrm'
Window(X, ..., from=c("points", "covariates"))

 ## S3 method for class 'msr'
Window(X, ...)

 ## S3 method for class 'quadrattest'
Window(X, ...)
```

Arguments

X	A spatial object.
...	Ignored.
from	Character string. See Details.

Details

These are methods for the generic function Window which extract the spatial window in which the object X is defined. The argument from applies when X is a fitted two-dimensional point process model (object of class "ppm", "kppm", "slrm" or "dppm"). If from="data" (the default), Window extracts the window of the original point pattern data to which the model was fitted. If from="covariates" then Window returns the window in which the spatial covariates of the model were provided.

Value

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

Author(s)

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

See Also

Window, Window.ppp, Window.psp.

owin.object

Examples

```
A <- ppm(cells ~ 1)
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 <r.turner@auckland.ac.nz>

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.msr` *Evaluate Expression Involving Components of a Measure*

Description

An expression involving the names of components of a measure is evaluated.

Usage

```
## S3 method for class 'msr'
with(data, expr, ...)
```

Arguments

`data`	A measure (object of class `"msr"`).
`expr`	An expression to be evaluated.
`...`	Ignored.

Details

This is a method for the generic function `with` for the class `"msr"`. The argument `data` should be an object of class `"msr"` representing a measure (a function which assigns a value to each subset of two-dimensional space).

This function can be used to extract the components of the measure, or to perform more complicated manipulations of the components.

The argument `expr` should be an un-evaluated expression in the R language. The expression may involve any of the variable names listed below with their corresponding meanings.

`qlocations`	(point pattern) all quadrature locations

qweights	(numeric) all quadrature weights
density	(numeric) density value at each quadrature point
discrete	(numeric) discrete mass at each quadrature point
continuous	(numeric) increment of continuous component
increment	(numeric) increment of measure
is.atom	(logical) whether quadrature point is an atom
atoms	(point pattern) locations of atoms
atommass	(numeric) massess of atoms

The measure is the sum of discrete and continuous components. The discrete component assigns non-zero mass to several points called atoms. The continuous component has a density which should be integrated over a region to determine the value for that region.

An object of class "msr" approximates the continuous component by a sum over quadrature points. The quadrature points are chosen so that they include the atoms of the measure. In the list above, we have increment = continuous + discrete, continuous = density * qweights, is.atom = (discrete > 0), atoms = qlocations[is.atom] and atommass = discrete[is.atom].

Value

The result of evaluating the expression could be an object of any kind.

Author(s)

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

See Also

msr, split.msr, measureContinuous, measurePositive

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X, ~x+y)
rp <- residuals(fit, type="pearson")

with(rp, atoms)
with(rp, qlocations %mark% continuous)
```

---

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)
```

`zclustermodel` *Cluster Point Process Model*

Description

Experimental code. Creates an object representing a cluster point process model. Typically used for theoretical calculations about such a model.

Usage

```
zclustermodel(name = "Thomas", ..., mu, kappa, scale)
```

Arguments

`name`	Name of the cluster process. One of `"Thomas"`, `"MatClust"`, `"VarGamma"` or `"Cauchy"`.
`...`	Other arguments needed for the model.
`mu`	Mean cluster size. A single number, or a pixel image.
`kappa`	Parent intensity. A single number.
`scale`	Cluster scale parameter of the model.

Details

Experimental.

Value

Object of the experimental class `"zclustermodel"`.

Author(s)

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

See Also

`methods.zclustermodel`

Examples

```
m <- zclustermodel("Thomas", kappa=10, mu=5, scale=0.1)
```

zgibbsmodel *Gibbs Model*

Description

Experimental code. Creates an object representing a Gibbs point process model. Typically used for theoretical calculations about such a model.

Usage

```
zgibbsmodel(beta = 1, interaction = NULL, icoef = NULL)
```

Arguments

beta	First order trend term. A numeric value, numeric vector, pixel image, function, or a list of such objects.
interaction	Object of class "interact" specifying the interpoint interaction structure, or NULL representing the Poisson process.
icoef	Numeric vector of coefficients for the interpoint interaction.

Details

Experimental.

Value

Object belonging to the experimental class zgibbsmodel.

Author(s)

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

See Also

methods.zgibbsmodel

Examples

```
m <- zgibbsmodel(10, Strauss(0.1), -0.5)
```

`[.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