Package 'spatstat.model'

May 22, 2025

Version 3.3-6

Date 2025-04-22

Title Parametric Statistical Modelling and Inference for the 'spatstat' Family

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

Depends R (>= 3.5.0), spatstat.data (>= 3.1-4), spatstat.univar (>= 3.1-1), spatstat.geom (>= 3.3-4), spatstat.random (>= 3.3-3.005), spatstat.explore (>= 3.3-0), stats, graphics, grDevices, utils, methods, nlme, rpart

Imports spatstat.utils (>= 3.1-2), spatstat.sparse (>= 3.1-0), mgcv, Matrix, abind, tensor, goftest (>= 1.2-2)

Suggests sm, gsl, locfit, spatial, fftwtools (>= 0.9-8), nleqslv, glmnet, spatstat.linnet (>= 3.2-2), spatstat (>= 3.3)

Description Functionality for parametric statistical modelling and inference for 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'.)
Supports parametric modelling, formal statistical inference, and model validation.
Parametric models include Poisson point processes, Cox point processes, Neyman-Scott cluster processes, Gibbs point processes and determinantal point processes. Models can be fitted to data using maximum likelihood, maximum pseudolikelihood, maximum composite likelihood and the method of minimum contrast. Fitted models can be simulated and predicted. Formal inference includes hypothesis tests (quadrat counting tests, Cressie-Read tests, Clark-Evans test, Berman test, Diggle-Cressie-Loosmore-Ford test, scan test, studentised permutation test, segregation test, ANOVA tests of fitted models, adjusted composite likelihood ratio test, envelope tests, Dao-Genton test, balanced independent two-stage test), confidence intervals for parameters, and prediction intervals for point counts. Model validation techniques include leverage, influence, partial residuals, added variable plots, diagnostic plots, pseudoscore residual plots, model compensators and Q-Q plots.

License GPL (>= 2)

URL http://spatstat.org/

NeedsCompilation yes

ByteCompile true

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

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

Repository CRAN

Date/Publication 2025-05-22 06:30:01 UTC

Contents

`spatstat.model-package`

The spatstat.model Package

Description

The **spatstat.model** package belongs to the **spatstat** family of packages. It contains the core functionality for parametric statistical 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.model** contains all the main user-level functions that perform parametric statistical 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.univar** containing functions for estimating probability distributions of random variables
- **spatstat.geom** containing geometrical objects and geometrical operations
- **spatstat.explore** containing the functionality for exploratory analysis and nonparametric modelling of spatial data

- **spatstat.model** containing the main functionality for parametric modelling, analysis and inference for 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.model

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.model** package only.

To simulate a random point pattern:

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

Exploratory analysis

Exploratory graphics, smoothing, and exploratory analysis of spatial data are now provided in the **spatstat.explore** package.

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.kppm	Compute offspring density
clusterradius.kppm	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

Simulation-based inference including simulation envelopes and hypothesis tests is now supported by the package **spatstat.explore**.

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.ppm`	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 <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

`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 <rolfturner@posteo.net>, 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))
```

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 <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

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

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 <rolfturner@posteo.net> 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> and Rolf Turner <rolfturner@posteo.net>

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 <rolfturner@posteo.net>

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
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.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 <rolfturner@posteo.net> 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.fv.kppm *Convert Fitted Model To Class fv*

Description

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

Usage

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

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

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

Arguments

x	A fitted model which will be converted into a function table

Details

The generic command as.fv 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.

Objects of class "kppm" (and related classes) represent a model that has been fitted to a dataset by computing a summary function of the dataset and matching it to the corresponding summary function of the model. The methods for as.fv for classes "kppm", "dppm" and "minconfit" extract this information: the result is a function table containing the observed summary function and the best fit summary function.

Value

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

Author(s)

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

Examples

```
as.fv(kppm(redwood))
```

---

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 <rolfturner@posteo.net>

See Also

fitin, ppm.

Examples

```
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 <rolfturner@posteo.net> 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)
```

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 some other suitable class from another package. For full details, see vignette('shapefiles').

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

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

Value

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

Author(s)

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

See Also

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

owin.object, owin.

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

Examples

```
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)

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

Arguments

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

For the class "rppm" of models fitted by recursive partitioning (regression trees), the method as.ppm.rppm extracts the corresponding loglinear model that is fitted in the first stage of the procedure (whose purpose is merely to identify and evaluate the explanatory variables).

Value

An object of class "ppm".

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> 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.ppm *Area Under ROC Curve*

Description

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

Usage

```
## 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").
...	Arguments passed to as.mask controlling the pixel resolution for calculations.

Details

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

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.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 <rolfturner@posteo.net> 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)
```

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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net>.

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.ppm` *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

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

Arguments

`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`, `eps` and `rule.eps` passed to `as.mask`) or other undocumented features.

Details

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

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

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

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

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

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

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

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

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

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

Value

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

Warning

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

Author(s)

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

References

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

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

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

See Also

cdf.test, quadrat.test, ppm

Examples

```
# Berman's data
X <- copper$SouthPoints
L <- copper$SouthLines
D <- distmap(L, eps=1)
# test of fitted model
fit <- ppm(X ~ x+y)
berman.test(fit, D)
```

---

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. (2013) Cauchy cluster process. *Metrika* **76**, 697–706.

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. (2013) Cauchy cluster process. *Metrika* **76**, 697–706.

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.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 `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 covariate 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 <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

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

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)
```

`cdf.test.ppm` *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

```
## 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

`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; and arguments passed to `as.mask` to control the pixel resolution.
`interpolate`	Logical flag indicating whether to interpolate pixel images. If `interpolate=TRUE`, the value of the covariate at each point of `X` will be approximated by interpolating the nearby pixel values. If `interpolate=FALSE`, the nearest pixel value will be used.
`jitter`	Logical flag. If `jitter=TRUE`, values of the covariate will be slightly perturbed at random, to avoid tied values in the test.

`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 <rolfturner@posteo.net>

References

Baddeley, A., Turner, R., Møller, J. and Hazelton, M. (2005) Residual analysis for spatial point processes. *Journal of the Royal Statistical Society, Series B* **67**, 617–666.

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

See Also

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

Examples

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


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

options(op)
```

---

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 <rolfturner@posteo.net>

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.kppm *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 function clusterfield is generic, with a method for "kppm" (described here) and methods for "character" and "function".

The method clusterfield.kppm extracts the relevant information from the fitted model and calls clusterfield.function.

The 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.)

Value

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

Author(s)

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

See Also

kppm,

clusterfield.

Examples

```
fit <- kppm(redwood~1, "Thomas")
Z <- clusterfield(fit, eps = 0.01)
```

`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 <rolfturner@posteo.net> 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.kppm *Extract Cluster Offspring Kernel*

Description

Given a fitted 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 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.

The function clusterkernel is generic, with methods for class "kppm" (described here) and "character" (described in clusterkernel.character).

Value

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

Author(s)

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

See Also

clusterkernel.character, clusterfield, kppm

Examples

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

---

clusterradius.kppm *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 <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

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

Examples

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

---

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 <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

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

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 <rolfturner@posteo.net>

See Also

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

Examples

```
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> and Rolf Turner <rolfturner@posteo.net>

See Also

slrm

Examples

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

---

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
```

`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 <rolfturner@posteo.net> 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)
```

---

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 <rolfturner@posteo.net>.

See Also

ppm.object, ppp.object

Examples

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

---

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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net> 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))
```

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 <rolfturner@posteo.net>

References

Baddeley, A., Turner, R., Møller, J. and Hazelton, M. (2005) Residual analysis for spatial point processes. *Journal of the Royal Statistical Society, Series B* **67**, 617–666.

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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net>

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

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, ...)
```

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 <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

domain, domain.quadratcount, domain.quadrattest, domain.rmhmodel, domain.lpp. Window, Frame.

Examples

```
domain(ppm(redwood ~ 1))
```

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 <rolfturner@posteo.net>

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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net>

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 <rolfturner@posteo.net>

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 <rolfturner@posteo.net>

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 <rolfturner@posteo.net>

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 <rolfturner@posteo.net>

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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net>

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 <rolfturner@posteo.net>

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 <rolfturner@posteo.net>

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 <rolfturner@posteo.net>

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 <rolfturner@posteo.net>

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 <rolfturner@posteo.net>

See Also

ppm.object, ppp.object, ppm

Examples

```
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
```

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 <rolfturner@posteo.net>

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

```
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 <rolfturner@posteo.net>.

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))
```

---

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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net>

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 <rolfturner@posteo.net> 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.ppm *Simulation Envelopes of Summary Function*

Description

Computes simulation envelopes of a summary function.

Usage

```
  ## 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 <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

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

Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.

Diggle, P.J. *Statistical analysis of spatial point patterns*. Arnold, 2003.

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

Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.

Stoyan, D. and Stoyan, H. (1994) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

See Also

dclf.test, mad.test for envelope-based tests.

fv.object, plot.envelope, plot.fv, envelope.envelope, pool.envelope for handling envelopes. There are also methods for print and summary.

Kest, Gest, Fest, Jest, pcf, ppp, ppm, default.expand

Examples

```
X <- simdat
online <- interactive()
Nsim <- if(online) 19 else 3


# Envelope of K function 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 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)
}
```

`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 <rolfturner@posteo.net>

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.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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net> 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]
```

`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 <rolfturner@posteo.net>

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

X <- unmark(spruces)

fit <- ppm(X ~ 1, Fiksel(r=3.5, kappa=1))
plot(fitin(fit))
```

---

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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net>

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

References

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

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 <rolfturner@posteo.net> 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, or the fitted probabilities at each original data point.

Usage

```
  ## S3 method for class 'slrm'
fitted(object, ..., type="probabilities",
                 dataonly=FALSE, leaveoneout=FALSE)
```

Arguments

`object`	a fitted spatial logistic regression model. An object of class `"slrm"`.
`...`	Ignored.
`type`	Character string (partially) matching one of `"probabilities"`, `"intensity"` or `"link"` determining the quantity that should be predicted.
`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 the pixels used to fit the model.
`leaveoneout`	Logical value specifying whether to perform a leave-one-out calculation when `dataonly=TRUE`. If `leaveoneout=TRUE`, the fitted value at each data point `X[i]` is calculated by re-fitting the model to the data with `X[i]` removed.

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`).

By default, the algorithm computes the fitted probabilities of the presence of a random point in each pixel, and returns them as an image.

If `dataonly=TRUE`, the algorithm computes the fitted presence probabilities only at the locations of the original data points.

Value

A pixel image (object of class `"im"`) containing the fitted probability for each pixel, or a numeric vector containing the fitted probability at each data point.

Author(s)

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

See Also

slrm, fitted

Examples

```
X <- rpoispp(42)
fit <- slrm(X ~ x+y)
plot(fitted(fit))
fitted(fit, dataonly=TRUE)
```

`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 <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

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

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)
```

---

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

```
fit <- ppm(cells, ~x)
formula(fit)
terms(fit)
```

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

```
    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)
```

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 <rolfturner@posteo.net>

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'
```

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

```
    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 <rolfturner@posteo.net>

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())

  # equivalent:
  ppm(cells ~1, Hardcore)



  # fit a nonstationary hard core process
  # with log-cubic polynomial trend
  ppm(cells ~ polynom(x,y,3), Hardcore(0.05))
```

---

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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net> 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.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 <rolfturner@posteo.net> 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)
```

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 <rolfturner@posteo.net>

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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net>

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** $i \le j$. 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 <rolfturner@posteo.net>

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
```

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 <rolfturner@posteo.net>

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
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 <rolfturner@posteo.net>.

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
}
```

---

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 Plenge Waagepetersen <rw@math.auc.dk>. Adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> 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)
```

`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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net>

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, weight=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.
weight	Optional. A pixel image (object of class "im") or a function(x,y) giving a numerical weight to be applied to the integration.
...	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.

If weight is given, it should be a pixel image or a function of coordinates x and y returning numerical values. Then each increment of the measure will be multiplied by the corresponding value of weight. Effectively, weight becomes the integrand, and the result is the integral of weight with respect to the measure f.

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 <rolfturner@posteo.net> 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))

## weighted integral
integral(rr, weight=function(x,y){y})
```

`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 <rolfturner@posteo.net>

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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net> 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)

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

---

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 <rolfturner@posteo.net>

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)

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 <rolfturner@posteo.net>.

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 `<rolfturner@posteo.net>`

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.poissonclusterprocess

Recognise Poisson Cluster 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 Poisson cluster process.

Usage

```
is.poissonclusterprocess(model)
## S3 method for class 'kppm'
is.poissonclusterprocess(model)
## S3 method for class 'zclustermodel'
is.poissonclusterprocess(model)
## Default S3 method:
is.poissonclusterprocess(model)
```

Arguments

model	Any kind of object representing a spatial point process model, either a model fitted to data, or a specification of a point process model.

Details

The argument model represents a fitted spatial point process model (such as an object of class "ppm", "kppm" or similar) or a specification of a point process model (such as an object of class "zclustermodel").

This function returns TRUE if the model is a Poisson cluster process, and FALSE otherwise.

The function is.poissonclusterprocess is generic, with methods for classes kppm and zclustermodel, and a default method.

Value

A logical value.

Author(s)

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

See Also

kppm, zclustermodel.

Examples

```
fut <- kppm(redwood ~ 1, "Thomas")
is.poissonclusterprocess(fut)
fot <- slrm(cells ~ x, dimyx=16)
is.poissonclusterprocess(fot)
```

---

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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net> 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>`.

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)
    }
```

`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>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net>

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

```
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)
```

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"),
       penalised = FALSE,
       improve.type = c("none", "clik1", "wclik1", "quasi"),
       improve.args = list(),
       weightfun=NULL,
       control=list(),
       stabilize=TRUE,
       algorithm,
       trajectory=FALSE,
       statistic="K",
       statargs=list(),
       rmax = NULL,
       epsilon=0.01,
       covfunargs=NULL,
       use.gam=FALSE,
       nd=NULL, eps=NULL,
       ppm.improve.type=c("none", "ho", "enet"),
       ppm.improve.args=list())

## 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"),
penalised = FALSE,
improve.type = c("none", "clik1", "wclik1", "quasi"),
improve.args = list(),
weightfun=NULL,
control=list(),
stabilize=TRUE,
algorithm,
trajectory=FALSE,
statistic="K",
statargs=list(),
rmax = NULL,
epsilon=0.01,
covfunargs=NULL,
use.gam=FALSE,
nd=NULL, eps=NULL,
ppm.improve.type=c("none", "ho", "enet"),
ppm.improve.args=list())
```

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.
penalised	Logical value specifying whether the objective function (the composite likelihood or contrast) should be modified by adding a penalty against extreme values of cluster scale.

`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, or one of the strings `"threshold"` or `"taper"`. 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"`.

`trajectory` Logical value specifying whether to save the history of all function evaluations performed by the optimization algorithm.

`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.

`ppm.improve.type`, `ppm.improve.args`
Arguments controlling the initial fit of the trend. Passed to `ppm` as the arguments `improve.type` and `improve.args` respectively.

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`. If it is specified, the argument `weightfun` should be a `function` in the R language with one argument. Alternatively `weightfun` may be one of the strings `"threshold"` or `"taper"` representing the functions $w(d) = 1(d \le R)$ and $w(d) = min(1, R/d)$ respectively.

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

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

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.

If `penalised=TRUE`, the fitting procedure is modified by adding a penalty against extreme values of the cluster scale, as proposed by Baddeley et al (2022).

If `trajectory=TRUE`, the resulting object contains the history of all points in the cluster parameter space which were evaluated by the optimization algorithm. The trajectory can be extracted by `traj(fit)` or `traj(obsurf(fit))` where `fit` is the fitted model object.

Value

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

Cluster parameters for Neyman-Scott models

For Neyman-Scott models, the fitting procedure searches for the best-fitting values of the parameters that control the intensity of parents and the physical scale of the clusters. (Any parameters that control the shape of the clusters must be specified separately and are assumed to be fixed.)

The fitted object `fit` contains the fitted cluster parameters as the element `fit$par` in the format described below. Initial estimates for these cluster parameters can be specified using the argument `startpar` in the same format.

The cluster parameters will be stored in a *named* numeric vector `par` of length 2. The first value is always `kappa`, the intensity of parents (cluster centres). The format is as follows:

- for `clusters="Thomas"`, a vector `c(kappa, sigma2)` where `sigma2` is the square of the cluster standard deviation;
- for `clusters="MatClust"`, a vector `c(kappa, R)` where `R` is the radius of the cluster;
- for `clusters="Cauchy"`, a vector `c(kappa, eta2)` where `eta2` = `code{4 * scale^2}` where `scale` is the scale parameter for the model as used in `rCauchy`;
- for `clusters="VarGamma"`, a vector `c(kappa, eta)` where `eta` is equivalent to the scale parameter `omega` used in `rVarGamma`.

For `clusters="VarGamma"` it will be necessary to specify the shape parameter `nu` as described in the help for `rVarGamma`. This is specified separately as an argument `nu` in the call to `kppm`.

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, 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. Additional arguments specify the shape parameters of the covariance model. For example if `model="matern"` then the additional argument `nu` is required.

The available models are as follows:

`model="exponential"`: the exponential covariance function

$$C(r) = \sigma^2 \exp(-r/h)$$

where σ^2 is the (fitted) variance parameter, and h is the (fitted) scale parameter. No shape parameters are required.

`model="gauss"`: the Gaussian covariance function

$$C(r) = \sigma^2 \exp(-(r/h)^2)$$

where σ^2 is the (fitted) variance parameter, and h is the (fitted) scale parameter. No shape parameters are required.

`model="stable"`: the stable covariance function

$$C(r) = \sigma^2 \exp(-(r/h)^\alpha)$$

where σ^2 is the (fitted) variance parameter, h is the (fitted) scale parameter, and α is the shape parameter `alpha`. The parameter `alpha` must be given, either as a stand-alone argument, or as an entry in the list `covmodel`.

`model="gencauchy"`: the generalised Cauchy covariance function

$$C(r) = \sigma^2 (1 + (x/h)^\alpha)^{-\beta/\alpha}$$

where σ^2 is the (fitted) variance parameter, h is the (fitted) scale parameter, and α and β are the shape parameters `alpha` and `beta`. The parameters `alpha` and `beta` must be given, either as stand-alone arguments, or as entries in the list `covmodel`.

`model="matern"`: the Whittle-Matérn covariance function

$$C(r) = \sigma^2 \frac{1}{2^{\nu-1}\Gamma(\nu)} (\sqrt{2\nu}\, r/h)^\nu K_\nu(\sqrt{2\nu}\, r/h)$$

where σ^2 is the (fitted) variance parameter, h is the (fitted) scale parameter, and ν is the shape parameter `nu`. The parameter `nu` must be given, either as a stand-alone argument, or as an entry in the list `covmodel`.

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 <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>, with contributions from Abdollah Jalilian <jalilian@razi.ac.ir> and Rasmus Plenge Waagepetersen <rw@math.auc.dk>. Adaptive composite likelihood method contributed by Chiara Fend and modified by Adrian Baddeley. Penalised optimization developed by Adrian Baddeley, Tilman Davies <Tilman.Davies@otago.ac.nz> and Martin Hazelton <Martin.Hazelton@otago.ac.nz>.

References

Baddeley, A., Davies, T.M., Hazelton, M.L., Rakshit, S. and Turner, R. (2022) Fundamental problems in fitting spatial cluster process models. *Spatial Statistics* **52**, 100709. DOI: 10.1016/j.spasta.2022.100709

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")
kppm(redwood ~ x, "LGCP", statistic="pcf",
                          model="matern", nu=0.3,
                          control=list(maxit=10))

# Different fitting techniques
fitc <- kppm(redwood ~ 1, "Thomas", method="c")
fitp <- kppm(redwood ~ 1, "Thomas", method="p")
# penalised fit
fitmp <- kppm(redwood ~ 1, "Thomas", penalised=TRUE)
# quasi-likelihood improvement
fitq <- 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

```
    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)
```

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 <rolfturner@posteo.net>

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))
```

`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 <rolfturner@posteo.net>

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
```

`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 <rolfturner@posteo.net> 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

$$C(r) = \sigma^2 e^{-r/\alpha}.$$

For a list of available models see kppm.

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 Plenge 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

kppm and lgcp.estpcf for alternative methods 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.

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

$$C(r) = \sigma^2 e^{-r/\alpha}.$$

For a list of available models see `kppm`.

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 Plenge 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

kppm and lgcp.estK for alternative methods 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.

pcf for the pair correlation function.

Examples

```
u <- lgcp.estpcf(redwood, c(var=1, scale=0.1))
u
plot(u)
lgcp.estpcf(redwood, covmodel=list(model="matern", nu=0.3))
```

---

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)

$$AIC = -2\log(PL) + k \times \mathrm{edf}$$

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 <rolfturner@posteo.net>

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

$$AIC = -2\log(CL) + k \times \text{edf}$$

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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

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

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 <rolfturner@posteo.net> 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

```
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> and Rolf Turner <rolfturner@posteo.net>.

See Also

slrm

Examples

```
X <- rpoispp(42)
fit <- slrm(X ~ x+y)
logLik(fit)
logLik(fit, adjust=FALSE)
```

---

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 <rolfturner@posteo.net>.

References

Baddeley, A., Turner, R., Møller, J. and Hazelton, M. (2005) Residual analysis for spatial point processes. *Journal of the Royal Statistical Society, Series B* **67**, 617–666.

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)
```

---

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

$$h(z) = 2 + \frac{1}{\pi}[(8z^2 - 4)\arccos(z) - 2\arcsin(z) + 4z\sqrt{(1-z^2)^3} - 6z\sqrt{1-z^2}]$$

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 Plenge 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

```
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

```
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)
```

measureWeighted *Weighted Version of a Measure*

Description

Given a measure m (object of class "msr") and a spatially-varying weight function, construct the weighted version of m.

Usage

```
measureWeighted(m, w)
```

Arguments

m	A measure (object of class "msr").
w	A pixel image (object of class "im") or a function(x,y) giving the numeric weight at each spatial location.

Details

For any region of space B, the weighted measure wm has the value

$$wm(B) = \int_B w(x) dm(x)$$

In any small region of space, the increment of the weighted measure wm is equal to the increment of the original measure m multiplied by the weight w at that location.

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

```
fit <- ppm(cells ~ x)
res <- residuals(fit)
measureWeighted(res, function(x,y){x})
```

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 <rolfturner@posteo.net> 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

```
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 <rolfturner@posteo.net> 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.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, fmla, ..., 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.
fmla	Optional. A formula, to replace the formula of the model.
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.traj *Methods for Trajectories of Function Evaluations*

Description

Methods for objects of class "traj".

Usage

```
## S3 method for class 'traj'
print(x, ...)
## S3 method for class 'traj'
plot(x, ..., show.ends=TRUE, add=FALSE, xlab=NULL, ylab=NULL)
## S3 method for class 'traj'
lines(x, ..., directed=FALSE)
```

Arguments

x	Object of class "traj".
...	Additional arguments passed to other methods.
directed	Logical value specifying whether to draw arrows instead of undirected lines.
show.ends	Logical value specifying whether to indicate the start and finish of the trajectory. The start is a blue circle; the finish is a red cross.
add	Logical value specifying whether to draw the trajectory on the existing plot (add=TRUE) or to start a new plot (add=FALSE, the default).
xlab, ylab	Optional labels for the horizontal and vertical axes.

Details

An object of class "traj" represents the history of evaluations of the objective function performed when a cluster process model was fitted. It is a data frame containing the input parameter values for the objective function, and the corresponding value of the objective function, that were considered by the optimisation algorithm.

These functions are methods for the generic print, plot and lines.

Value

Null.

Author(s)

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

See Also

traj

Examples

```
fit <- kppm(redwood, pspace=list(save=TRUE))
h <- traj(fit)
h
plot(h)
lines(h)
```

`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)

 ## S3 method for class 'zclustermodel'
simulate(object, nsim=1, ..., win=unit.square())
```

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.

win	Window (object of class "owin") in which the simulated pattern should be generated.
nsim	Number of simulated patterns to be generated.

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))
X <- simulate(m)
```

---

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 Plenge 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,

---

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 ith canonical covariate depends on the jth 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 ith offset term depends on the jth 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 <rolfturner@posteo.net>

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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net> 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 `<rolfturner@posteo.net>`

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 `i`th experimental unit (corresponding to the `i`th 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 <rolfturner@posteo.net> 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*. 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, measureWeighted, 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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net>

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'
```

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,
                              enclose=FALSE,
                              ratio = 1.5, verbose = TRUE)

## S3 method for class 'kppm'
objsurf(x, ..., ngrid = 32, xlim=NULL, ylim=NULL,
                              enclose=FALSE,
                              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.
`enclose`	Logical value specifying whether the default values of `xlim` and `ylim` should enclose the history of all function evaluations. See Details.
`ratio`	Number greater than 1 determining the default ranges of parameter values. See Details.
`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.

The range of parameter values to be considered is determined by xlim and ylim. The default values of xlim and ylim are chosen as follows.

- if enclose=FALSE (the default), the default values of xlim and ylim are the ranges from opt/ratio to opt * ratio where opt is the optimal parameter value on the surface.
- If enclose=TRUE, and if x contains a trajectory (history of function evaluations), then xlim and ylim will be the ranges of parameter values examined in the trajectory.

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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net>

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 `<rolfturner@posteo.net>`

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 <rolfturner@posteo.net>

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

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 <rolfturner@posteo.net>

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

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
```

---

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 `<rolfturner@posteo.net>`

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 <rolfturner@posteo.net>

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"))
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"))
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 <rolfturner@posteo.net>

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.

---

palmdiagnose *Diagnostic based on Palm Intensity*

Description

Given a fitted cluster process or Cox process model, calculate a diagnostic which compares nonparametric and parametric estimates of the Palm intensity.

Usage

```
palmdiagnose(object, ..., breaks = 30, trim = 30, rmax=Inf)
```

Arguments

object	Fitted model (object of class "kppm") or a list of fitted models.
...	Optional. Additional arguments which are fitted models of class "kppm".
breaks	Optional argument passed to cut.default determining the breakpoints of distance values for the nonparametric estimate. Either an integer specifying the number of breakpoints, or a numeric vector of distance values to be used as the breakpoints.
trim	Optional. Maximum value of the translation edge correction weight.
rmax	Optional. Maximum interpoint distance r that should be considered. See Details.

Details

This function computes the diagnostic proposed by Tanaka, Ogata and Stoyan (2008, Section 2.3) for assessing goodness-of-fit of a Neyman-Scott cluster process model to a point pattern dataset.

The fitted model `object` should be an object of class `"kppm"` representing a Neyman-Scott cluster process model or a Cox process model. In the current implementation, the model must be stationary.

The code computes parametric and non-parametric estimates of the Palm intensity $\lambda_0(r)$, loosely speaking, the intensity of the point process given that there is a point at the origin. The parametric estimate is obtained from the fitted model by substituting the fitted parameter estimates into expressions for the pair correlation and the intensity.

The non-parametric estimate is obtained by considering all pairs of data points, dividing the range of interpoint distances into several equally-spaced bands (determined by the argument `breaks`), counting the number of pairs of points whose interpoint distances fall in each band, and numerically adjusting for edge effects. Tanaka, Ogata and Stoyan (2008) used the periodic (toroidal) edge correction; our code uses the translation edge correction so that the method can be applied to data in any window.

The result is a function value table (object of class `"fv"`) containing the nonparametric and parametric estimates of the Palm intensity. The result also belongs to the class `"palmdiag"` which has a method for `plot`. The default behaviour of `plot.palmdiag` is to plot the model fit as a curve, and to display the nonparametric estimates as dots; this is the plot style proposed by Tanaka, Ogata and Stoyan (2008). Alternative display styles are also supported by `plot.palmdiag`.

For computational efficiency, the argument `rmax` specifies the maximum value of interpoint distance r for which estimates of $\lambda_0(r)$ shall be computed. The default `rmax = Inf` implies there is no constraint on interpoint distance, and the resulting function object contains estimates of $\lambda_0(r)$ up to the maximum distance that would have been observable in the window containing the original point pattern data.

If there are additional arguments `...` which are fitted models of class `"kppm"`, or if `object` is a list of fitted models of class `"kppm"`, then the parametric estimates for each of the fitted models will be included in the resulting function object. If names are attached to these fitted models, the names will be used in the resulting function object.

Value

Function value table (object of class `"fv"`) containing the nonparametric and parametric estimates of the Palm intensity. Also belongs to the class `"palmdiag"` which has a `plot` method.

Author(s)

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

References

Tanaka, U., Ogata, Y. and Stoyan, D. (2008) Parameter estimation and model selection for Neyman-Scott Point Processes. *Biometrical Journal* **50**, 1, 43–57.

See Also

`plot.palmdiag`

Examples

```
fitK <- kppm(redwood)
R <- palmdiagnose(fitK)
plot(R)

fitg <- kppm(redwood, statistic="pcf")
R2 <- palmdiagnose(A=fitK, B=fitg)
plot(R2)
```

`panel.contour` *Panel Plots using Colour Image or Contour Lines*

Description

These functions can be passed to `pairs` or `coplot` to determine what kind of plotting is done in each panel of a multi-panel graphical display.

Usage

```
panel.contour(x, y, ..., sigma = NULL)

panel.image(x, y, ..., sigma = NULL)

panel.histogram(x, ...)
```

Arguments

`x, y`	Coordinates of points in a scatterplot.
`...`	Extra graphics arguments, passed to `contour.im`, `plot.im` or `rect`, respectively, to control the appearance of the panel.
`sigma`	Bandwidth of kernel smoother, on a scale where x and y range between 0 and 1.

Details

These functions can serve as one of the arguments `panel`, `lower.panel`, `upper.panel`, `diag.panel` passed to graphics commands like `pairs` or `coplot`, to determine what kind of plotting is done in each panel of a multi-panel graphical display. In particular they work with `pairs.im`.

The functions `panel.contour` and `panel.contour` are suitable for the off-diagonal plots which involve two datasets `x` and `y`. They first rescale `x` and `y` to the unit square, then apply kernel smoothing with bandwidth `sigma` using `density.ppp`. Then `panel.contour` draws a contour plot while `panel.image` draws a colour image.

The function `panel.histogram` is suitable for the diagonal plots which involve a single dataset `x`. It displays a histogram of the data.

Value

Null.

Author(s)

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

See Also

pairs.im, pairs.default, panel.smooth

Examples

```
pairs(bei.extra,
      panel      = panel.contour,
      diag.panel = panel.histogram)
with(bei.extra,
      pairs(grad, elev,
            panel      = panel.image,
            diag.panel = panel.histogram))
pairs(marks(finpines), panel=panel.contour, diag.panel=panel.histogram)
```

---

panysib *Probability that a Point Has Any Siblings*

Description

Given a cluster process model, calculate the probability that a point of the process has any siblings.

Usage

```
panysib(object)
```

Arguments

object	Fitted cluster 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. This function computes the probability that a given random point has any siblings.

If object is a stationary point process, the result is a single number, which is the probability that a typical point of the process has any siblings. If this number is small, then the process is approximately a homogeneous Poisson process (complete spatial randomness). The converse is not true (Baddeley et al, 2022).

Otherwise, the result is a pixel image, in which the value at any location u is the conditional probability, given there is a point of the process at u, that this point has any siblings. If the pixel values are all small, then the process is approximately an inhomogeneous Poisson process.

This concept was proposed by Baddeley et al (2022).

Value

A single number (if `object` is a stationary point process) or a pixel image (otherwise).

Author(s)

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

References

Baddeley, A., Davies, T.M., Hazelton, M.L., Rakshit, S. and Turner, R. (2022) Fundamental problems in fitting spatial cluster process models. *Spatial Statistics* **52**, 100709. DOI: 10.1016/j.spasta.2022.100709

See Also

psib

Examples

```
fit <- kppm(redwood ~ polynom(x,y,2))
plot(panysib(fit))
```

`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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net>, 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))
```

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 <rolfturner@posteo.net> 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.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 <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

dppm, plot.ppm, plot.fv.

Examples

```
fit <- dppm(swedishpines ~ x + y, dppGauss, method="c")
plot(fit)
```

`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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

kppm, plot.ppm, plot.fv.

Examples

```
fit <- kppm(redwood~1, "Thomas")
plot(fit)
```

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 <rolfturner@posteo.net> 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"),
                main)
```

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.
main	Character string for the main title of the plot.

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 <rolfturner@posteo.net>

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

References

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

See Also

plot.ppm, mppm, plot.anylist

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 <rolfturner@posteo.net> 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.palmdiag *Plot the Palm Intensity Diagnostic*

Description

Plot the Palm intensity diagnostic for a fitted cluster process or Cox process model.

Usage

```
## S3 method for class 'palmdiag'
plot(x, ...,
          style = c("intervals", "dots", "bands"),
          args.dots = list(pch = 16), args.intervals = list(),
          xlim=NULL, main)
```

Arguments

x	Object of class "palmdiag" produced by palmdiagnose.
...	Additional arguments passed to plot.fv when the fitted curve is plotted.
style	Character string specifying the style of plot for the nonparametric estimates. See Details.
args.dots	Arguments passed to points when style="dots".
args.intervals	Arguments passed to segments when style="intervals".
xlim	Optional range of distances plotted along the horizontal axis. A numeric vector of length 2.
main	Optional main title for plot.

Details

This function plots the diagnostic proposed by Tanaka, Ogata and Stoyan (2008, Section 2.3) for assessing goodness-of-fit of a Neyman-Scott cluster process model to a point pattern dataset. The diagnostic is computed by the function palmdiagnose.

First the Palm intensity of the fitted model is plotted as a function of interpoint distance r using plot.fv. Then the nonparametric estimates of the Palm intensity are plotted on the same graph as follows:

- if style="dots", the nonparametric estimate for each band of distances is plotted as a dot, with horizontal coordinate at the middle of the band. This is the style proposed by Tanaka et al (2008).
- if style="intervals" (the default), each nonparametric estimate is plotted as a dot, and a 95% confidence interval is plotted as a vertical line segment, centred on the dot. The confidence interval is based on the Poisson approximation.
- if style="bands", the nonparametric estimates are drawn as a continuous curve which is flat on each band of distances. The 95% confidence intervals are drawn as grey shading.

Value

Null.

Author(s)

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

References

Tanaka, U., Ogata, Y. and Stoyan, D. (2008) Parameter estimation and model selection for Neyman-Scott Point Processes. *Biometrical Journal* **50**, 1, 43–57.

See Also

palmdiagnose

Examples

```
fit <- kppm(redwood)
R <- palmdiagnose(fit)
plot(R, style="d")
plot(R)
plot(R, style="b")
```

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 <rolfturner@posteo.net>

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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net> 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.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 <rolfturner@posteo.net> 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.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> and Rolf Turner <rolfturner@posteo.net>

See Also

`slrm`, `predict.slrm`, `plot.im`

Examples

```
X <- copper$SouthPoints
Y <- copper$SouthLines
Z <- distmap(Y)
fit <- slrm(X ~ Z)
plot(fit)
plot(fit, type="link")
```

`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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net> 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)
```

---

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 <rolfturner@posteo.net> 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="VBlogi")

 ppm(swedishpines ~ 1, Strauss(9), improve.type="ho",
     nsim=if(!online) 8 else 99)

 # Elastic net fit:
 if(require(glmnet)) {
   ppm(swedishpines ~ x+y, Strauss(9), improve.type="enet")
 }

# 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 <rolfturner@posteo.net>.

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=c("mpl", "logi", "VBlogi"),
       forcefit=FALSE,
       improve.type = c("none", "ho", "enet"),
       improve.args=list(),
       emend=project,
       project=FALSE,
       prior.mean = NULL,
       prior.var = NULL,
       nd = NULL,
       eps = NULL,
       quad.args=list(),
       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=c("mpl", "logi", "VBlogi"),
        forcefit=FALSE,
        improve.type = c("none", "ho", "enet"),
        improve.args=list(),
        emend=project,
        project=FALSE,
        prior.mean = NULL,
        prior.var = NULL,
        nd = NULL,
        eps = NULL,
        quad.args=list(),
        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	String (partially matched) specifying the method used to fit the model. Options are "mpl" for the method of Maximum PseudoLikelihood (the default), "logi" for the Logistic Likelihood method and "VBlogi" for the Variational Bayes Logistic 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.
improve.type	String (partially matched) specifying a method for improving the initial fit. If improve.type = "none" (the default), no improvement is performed. If improve.type="ho", the Huang-Ogata approximate maximum likelihood method is used. If improve.type="enet", the model coefficients are re-estimated using a regularized version of the composite likelihood.
improve.args	Arguments used to control the algorithm for improving the initial fit. See Details.
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.
quad.args	Arguments controlling the construction of the quadrature scheme, when Q is a point pattern. A list of arguments that will be passed to quadscheme or (if method="logi") to quadscheme.logi.
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 improve.type="ho")
nrmh	Number of Metropolis-Hastings iterations for each simulated realisation (for improve.type="ho")
start, control	Arguments passed to rmh controlling the behaviour of the Metropolis-Hastings algorithm (for improve.type="ho")
verb	Logical flag indicating whether to print progress reports (for improve.type="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.

The argument quad.args can be used to control the construction of the quadrature scheme. For example quad.args=list(quasi=TRUE, method="dirichlet", eps=0.1) would create dummy points according to a quasirandom pattern, with a typical spacing of 0.1 units between dummy points, and compute quadrature weights based on the Dirichlet tessellation.

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 ith 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.

Note that method='logi' and method='VBlogi' involve randomisation, so that the results are subject to random variation.

After this initial fit, there are several ways to improve the fit:

improve.type="none": No further improvement is performed.

improve.type="ho": the model will be re-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. This method involves randomisation, so the results are subject to random variation.

improve.type="enet": The model will be re-fitted using a regularized version of the composite likelihood. See below.

Huang-Ogata method: If `improve.type="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.

Regularization: This requires the package **glmnet**. **Details to be written.**

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.

"Model is invalid" or "Model is not valid" The fitted model coefficients do not define a valid point process. This can occur because some of the fitted coefficients are `NA` (perhaps because the model formula included redundant covariates so that the coefficients cannot be estimated), or because the fitted interaction coefficients do not define a valid point process (e.g. because a point process model which always has inhibition between points was fitted to a clustered point pattern). See `valid.ppm` for detailed information.

"System is computationally singular" or "Fisher information matrix is singular" The software cannot calculate standard errors or confidence intervals for the coefficients of the fitted model. This requires the (asymptotic) variance-covariance matrix, which is the inverse matrix of the Fisher information matrix of the fitted model. The error message states that the determinant of the Fisher information matrix is zero, or close to zero, so that the matrix cannot be inverted. 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 <rolfturner@posteo.net> 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)
```

---

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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net>

See Also

`kppm`, `plot.kppm`, `vcov.kppm`, `fitted.ppm`, `predict.ppm`

Examples

```
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 <rolfturner@posteo.net> 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*. 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,
   rule.eps = c("adjust.eps", "grow.frame", "shrink.frame"),
   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.

rule.eps
: Argument passed to as.mask controlling the discretisation. See as.mask.

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 <rolfturner@posteo.net>

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 <rolfturner@posteo.net> 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> and Rolf Turner <rolfturner@posteo.net>

See Also

slrm

Examples

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

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 <rolfturner@posteo.net>

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 <rolfturner@posteo.net> 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

```
human <- interactive()
# one irregular parameter
if(human) {
   rr <- data.frame(r=seq(0.05,0.15, by=0.01))
} else {
  rr <- data.frame(r=c(0.05,0.1,0.15))
}
ps <- profilepl(rr, Strauss, cells)
ps
plot(ps)

# two irregular parameters
if(human) {
   rs <- expand.grid(r=seq(0.05,0.15, by=0.01),sat=1:3)
} else {
  rs <- expand.grid(r=c(0.07,0.12),sat=1:2)
}
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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net> 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). It was shown in Baddeley et al (2022) that the sibling probability is directly related to the strength of clustering.

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.

Baddeley, A., Davies, T.M., Hazelton, M.L., Rakshit, S. and Turner, R. (2022) Fundamental problems in fitting spatial cluster process models. *Spatial Statistics* **52**, 100709. DOI: 10.1016/j.spasta.2022.100709

See Also

kppm, panysib

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 <rolfturner@posteo.net>

References

Baddeley, A., Turner, R., Møller, J. and Hazelton, M. (2005) Residual analysis for spatial point processes. *Journal of the Royal Statistical Society, Series B* **67**, 617–666.

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

```
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, 4, type="pearson")
  ## or use the idiom .Last.value if you forgot to assign them
  qqplot.ppm(fit, 4, type="pearson")
  mypreciousdata <- .Last.value
  plot(mypreciousdata)


########################################################
# 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 <rolfturner@posteo.net>

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.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 `fit2 <- 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 <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

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

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.ppm *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

```
## 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)
```

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.
df.est	Optional. Advanced use only. The number of fitted parameters, or the number of degrees of freedom lost by estimation of parameters.
...	Ignored.
xbreaks	Optional. Numeric vector giving the x coordinates of the boundaries of the quadrats. Incompatible with nx.
ybreaks	Optional. Numeric vector giving the y coordinates of the boundaries of the quadrats. Incompatible with ny.
tess	Tessellation (object of class "tess" or something acceptable to as.tess) determining the quadrats. Incompatible with nx, ny, xbreaks, ybreaks.
nsim	The number of simulated samples to generate when method="MonteCarlo".

Details

These functions perform χ^2 tests or Monte Carlo tests of goodness-of-fit for a point process model, based on quadrat counts.

The function `quadrat.test` is generic, with methods for point patterns (class `"ppp"`), split point patterns (class `"splitppp"`), point process models (class `"ppm"` or `"slrm"`) and quadrat count tables (class `"quadratcount"`).

- if `X` is a point pattern, we test the null hypothesis that the data pattern is a realisation of Complete Spatial Randomness (the uniform Poisson point process). Marks in the point pattern are ignored. (If `lambda` is given then the null hypothesis is the Poisson process with intensity `lambda`.)
- if `X` is a split point pattern, then for each of the component point patterns (taken separately) we test the null hypotheses of Complete Spatial Randomness. See `quadrat.test.splitppp` for documentation.
- If `X` is a fitted point process model, then it should be a Poisson point process model. The data to which this model was fitted are extracted from the model object, and are treated as the data point pattern for the test. We test the null hypothesis that the data pattern is a realisation of the (inhomogeneous) Poisson point process specified by `X`.

In all cases, the window of observation is divided into tiles, and the number of data points in each tile is counted, as described in `quadratcount`. The quadrats are rectangular by default, or may be regions of arbitrary shape specified by the argument `tess`. The expected number of points in each quadrat is also calculated, as determined by CSR (in the first case) or by the fitted model (in the second case). Then the Pearson X^2 statistic

$$X^2 = sum((observed - expected)^2/expected)$$

is computed.

If `method="Chisq"` then a χ^2 test of goodness-of-fit is performed by comparing the test statistic to the χ^2 distribution with $m - k$ degrees of freedom, where `m` is the number of quadrats and k is the number of fitted parameters (equal to 1 for `quadrat.test.ppp`). The default is to compute the *two-sided* p-value, so that the test will be declared significant if X^2 is either very large or very small. One-sided p-values can be obtained by specifying the `alternative`. An important requirement of the χ^2 test is that the expected counts in each quadrat be greater than 5.

If `method="MonteCarlo"` then a Monte Carlo test is performed, obviating the need for all expected counts to be at least 5. In the Monte Carlo test, `nsim` random point patterns are generated from the null hypothesis (either CSR or the fitted point process model). The Pearson X^2 statistic is computed as above. The p-value is determined by comparing the X^2 statistic for the observed point pattern, with the values obtained from the simulations. Again the default is to compute the *two-sided* p-value.

If `conditional` is `TRUE` then the simulated samples are generated from the multinomial distribution with the number of "trials" equal to the number of observed points and the vector of probabilities equal to the expected counts divided by the sum of the expected counts. Otherwise the simulated samples are independent Poisson counts, with means equal to the expected counts.

If the argument CR is given, then instead of the Pearson X^2 statistic, the Cressie-Read (1984) power divergence test statistic

$$2nI = \frac{2}{CR(CR+1)} \sum_i \left[\left(\frac{X_i}{E_i} \right)^C R - 1 \right]$$

is computed, where X_i is the ith observed count and E_i is the corresponding expected count. The value CR=1 gives the Pearson X^2 statistic; CR=0 gives the likelihood ratio test statistic G^2; CR=-1/2 gives the Freeman-Tukey statistic T^2; CR=-1 gives the modified likelihood ratio test statistic GM^2; and CR=-2 gives Neyman's modified statistic NM^2. In all cases the asymptotic distribution of this test statistic is the same χ^2 distribution as above.

The return value is an object of class "htest". Printing the object gives comprehensible output about the outcome of the test.

The return value also belongs to the special class "quadrat.test". Plotting the object will display the quadrats, annotated by their observed and expected counts and the Pearson residuals. See the examples.

Value

An object of class "htest". See chisq.test for explanation.

The return value is also an object of the special class "quadrattest", and there is a plot method for this class. See the examples.

Author(s)

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

References

Cressie, N. and Read, T.R.C. (1984) Multinomial goodness-of-fit tests. *Journal of the Royal Statistical Society, Series B* **46**, 440–464.

See Also

quadrat.test.splitppp, quadratcount, quadrats, quadratresample, chisq.test, cdf.test.

To test a Poisson point process model against a specific alternative, use anova.ppm.

Examples

```
# 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)))
```

`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 <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

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

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)
```

---

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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net>

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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

Examples

```
fit <- kppm(redwood ~ 1)
reach(fit)
```

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 <rolfturner@posteo.net> 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")
```

---

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)
```

`residualMeasure` *Residual Measure for an Observed Point Pattern and a Fitted Intensity*

Description

Given a point pattern and an estimate of its intensity function obtained in any fashion, compute the residual measure.

Usage

```
residualMeasure(Q, lambda,
                type = c("raw", "inverse", "Pearson", "pearson"),
                ...)
```

Arguments

Q	A point pattern (object of class "ppp") or quadrature scheme (object of class "quad").
lambda	Predicted intensity. An image (object of class "im") or a list of images.
type	Character string (partially matched) specifying the type of residuals.
...	Arguments passed to quadscheme if Q is a point pattern.

Details

This command constructs the residual measure for the model in which Q is the observed point pattern or quadrature scheme, and lambda is the estimated intensity function obtained in some fashion.

Value

A measure (object of class "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.

See Also

residuals.ppm

Examples

```
## nonparametric regression estimate of intensity
## as a function of terrain elevation
f <- rhohat(bei, bei.extra$elev)
## predicted intensity as a function of location
lam <- predict(f)
## residuals
res <- residualMeasure(bei, lam)
res
plot(res)
```

`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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net>

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 <rolfturner@posteo.net> 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*. 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 <rolfturner@posteo.net>

References

Baddeley, A., Turner, R., Møller, J. and Hazelton, M. (2005) Residual analysis for spatial point processes. *Journal of the Royal Statistical Society, Series B* **67**, 617–666.

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.rppm *Residuals for Recursively Partitioned Point Process Model*

Description

Given a point process model that was fitted to a point pattern by recursive partitioning (regression tree) methods, compute the residual measure.

Usage

```
  ## S3 method for class 'rppm'
residuals(object,
                          type=c("raw", "inverse", "Pearson"),
                          ...)
```

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 the residual measure for a point process model that was fitted to point pattern data by recursive partitioning of the covariates.

The argument `object` must be a fitted model object of class `"rppm"`). Such objects are created by the fitting algorithm `rppm`.

The type of residual is chosen by the argument `type`.

Value

A measure (object of class `"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.

See Also

`residuals.ppm`

Examples

```
fit <- rppm(bei ~ elev + grad, data=bei.extra)
res <- residuals(fit)
plot(res)
```

`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. For a point process model, the observed point pattern is extracted.

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 'rppm'
response(object)

## S3 method for class 'mppm'
response(object)
```

Arguments

object	A fitted model (object of class "lm", "glm", "ppm", "kppm", "dppm", "slrm", "rppm", 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", "lppm" or "rppm", 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, response.slrm and response.rppm, 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 <rolfturner@posteo.net>.

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)
```

---

rhohat.ppm *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

```
## S3 method for class 'ppm'
rhohat(object, covariate, ...,
       weights=NULL,
       method=c("ratio", "reweight", "transform"),
       horvitz=FALSE,
       smoother=c("kernel", "local", "decreasing", "increasing",
                  "mountain", "valley", "piecewise"),
       subset=NULL,
       do.CI=TRUE,
       jitter=TRUE, jitterfactor=1, interpolate=TRUE,
       dimyx=NULL, eps=NULL,
       rule.eps = c("adjust.eps", "grow.frame", "shrink.frame"),
       n = 512, bw = "nrd0", adjust=1, from = NULL, to = NULL,
       bwref=bw,
       covname, confidence=0.95, positiveCI, breaks=NULL)

## S3 method for class 'slrm'
rhohat(object, covariate, ...,
       weights=NULL,
       method=c("ratio", "reweight", "transform"),
       horvitz=FALSE,
       smoother=c("kernel", "local", "decreasing", "increasing",
                  "mountain", "valley", "piecewise"),
       subset=NULL,
```

```
do.CI=TRUE,
jitter=TRUE, jitterfactor=1, interpolate=TRUE,
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.
method	Character string determining the estimation method. See Details.
horvitz	Logical value indicating whether to use Horvitz-Thompson weights. See Details.
smoother	Character string determining the smoothing algorithm and the type of curve that will be estimated. See Details.
subset	Optional. A spatial window (object of class "owin") specifying a subset of the data, from which the estimate should be calculated.
do.CI	Logical value specifying whether to calculate standard errors and confidence bands.
jitter	Logical value. If jitter=TRUE (the default), the values of the covariate at the data points will be jittered (randomly perturbed by adding a small amount of noise) using the function jitter. If jitter=FALSE, the covariate values at the data points will not be altered. See the section on *Randomisation and discretisation*.
jitterfactor	Numeric value controlling the scale of noise added to the covariate values at the data points when jitter=TRUE. Passed to the function jitter as the argument factor.
interpolate	Logical value specifying whether to use spatial interpolation to obtain the values of the covariate at the data points, when the covariate is a pixel image (object of class "im"). If interpolate=FALSE, the covariate value for each data point is simply the value of the covariate image at the pixel centre that is nearest to the data point. If interpolate=TRUE, the covariate value for each data point is obtained by interpolating the nearest pixel values using interp.im.
dimyx, eps, rule.eps	Arguments controlling the pixel resolution at which the covariate will be evaluated. See Details.
bw	Smoothing bandwidth or bandwidth rule (passed to density.default).

adjust	Smoothing bandwidth adjustment factor (passed to density.default).
n, from, to	Arguments passed to density.default to control the number and range of values at which the function will be estimated.
bwref	Optional. An alternative value of bw to use when smoothing the reference density (the density of the covariate values observed at all locations in the window).
...	Additional arguments passed to density.default or locfit.
covname	Optional. Character string to use as the name of the covariate.
confidence	Confidence level for confidence intervals. A number between 0 and 1.
positiveCI	Logical value. If TRUE, confidence limits are always positive numbers; if FALSE, the lower limit of the confidence interval may sometimes be negative. Default is FALSE if smoother="kernel" and TRUE if smoother="local". See Details.
breaks	Breakpoints for the piecewise-constant function computed when smoother='piecewise'. Either a vector of numeric values specifying the breakpoints, or a single integer specifying the number of equally-spaced breakpoints. There is a sensible default.

Details

This command estimates the relationship between point process intensity and a given spatial covariate. Such a relationship is sometimes called a *resource selection function* (if the points are organisms and the covariate is a descriptor of habitat) or a *prospectivity index* (if the points are mineral deposits and the covariate is a geological variable). This command uses nonparametric methods which do not assume a particular form for the relationship.

If object is a point pattern, and baseline is missing or null, this command assumes that object is a realisation of a point process with intensity function $\lambda(u)$ of the form

$$\lambda(u) = \rho(Z(u))$$

where Z is the spatial covariate function given by covariate, and $\rho(z)$ is the resource selection function or prospectivity index. A nonparametric estimator of the function $\rho(z)$ is computed.

If object is a point pattern, and baseline is given, then the intensity function is assumed to be

$$\lambda(u) = \rho(Z(u))B(u)$$

where $B(u)$ is the baseline intensity at location u. A nonparametric estimator of the relative intensity $\rho(z)$ is computed.

If object is a fitted point process model, suppose X is the original data point pattern to which the model was fitted. Then this command assumes X is a realisation of a Poisson point process with intensity function of the form

$$\lambda(u) = \rho(Z(u))\kappa(u)$$

where $\kappa(u)$ is the intensity of the fitted model object. A nonparametric estimator of the relative intensity $\rho(z)$ is computed.

The nonparametric estimation procedure is controlled by the arguments smoother, method and horvitz.

The argument smoother selects the type of estimation technique.

- If `smoother="kernel"` (the default), the nonparametric estimator is a *kernel smoothing estimator* of $\rho(z)$ (Guan, 2008; Baddeley et al, 2012). The estimated function $\rho(z)$ will be a smooth function of z which takes nonnegative values. If `do.CI=TRUE` (the default), confidence bands are also computed, assuming a Poisson point process. See the section on *Smooth estimates*.
- If `smoother="local"`, the nonparametric estimator is a *local regression estimator* of $\rho(z)$ (Baddeley et al, 2012) obtained using local likelihood. The estimated function $\rho(z)$ will be a smooth function of z. If `do.CI=TRUE` (the default), confidence bands are also computed, assuming a Poisson point process. See the section on *Smooth estimates*.
- If `smoother="increasing"`, we assume that $\rho(z)$ is an increasing function of z, and use the *nonparametric maximum likelihood estimator* of $\rho(z)$ described by Sager (1982). The estimated function will be a step function, that is increasing as a function of z. Confidence bands are not computed. See the section on *Monotone estimates*.
- If `smoother="decreasing"`, we assume that $\rho(z)$ is a decreasing function of z, and use the *nonparametric maximum likelihood estimator* of $\rho(z)$ described by Sager (1982). The estimated function will be a step function, that is decreasing as a function of z. Confidence bands are not computed. See the section on *Monotone estimates*.
- If `smoother="mountain"`, we assume that $\rho(z)$ is a function with an inverted U shape, with a single peak at a value z_0, so that $\rho(z)$ is an increasing function of z for $z < z_0$ and a decreasing function of z for $z > z_0$. We compute the *nonparametric maximum likelihood estimator*. The estimated function will be a step function, which is increasing and then decreasing as a function of z. Confidence bands are not computed. See the section on *Unimodal estimates*.
- If `smoother="valley"`, we assume that $\rho(z)$ is a function with a U shape, with a single minimum at a value z_0, so that $\rho(z)$ is a decreasing function of z for $z < z_0$ and an increasing function of z for $z > z_0$. We compute the *nonparametric maximum likelihood estimator*. The estimated function will be a step function, which is decreasing and then increasing as a function of z. Confidence bands are not computed. See the section on *Unimodal estimates*.
- If `smoother="piecewise"`, the estimate of $\rho(z)$ is piecewise constant. The range of covariate values is divided into several intervals (ranges or bands). The endpoints of these intervals are the breakpoints, which may be specified by the argument `breaks`; there is a sensible default. The estimate of $\rho(z)$ takes a constant value on each interval. The estimate of $\rho(z)$ in each interval of covariate values is simply the average intensity (number of points per unit area) in the relevant sub-region. If `do.CI=TRUE` (the default), confidence bands are computed assuming a Poisson process.

See Baddeley (2018) for a comparison of these estimation techniques (except for `"mountain"` and `"valley"`).

If the argument `weights` is present, then the contribution from each data point `X[i]` to the estimate of ρ is multiplied by `weights[i]`.

If the argument `subset` is present, then the calculations are performed using only the data inside this spatial region.

This technique assumes that `covariate` has continuous values. It is not applicable to covariates with categorical (factor) values or discrete values such as small integers. For a categorical covariate, use `intensity.quadratcount` applied to the result of `quadratcount(X, tess=covariate)`.

The argument `covariate` should be a pixel image, or a function, or one of the strings `"x"` or `"y"` signifying the cartesian coordinates. It will be evaluated on a fine grid of locations, with spatial resolution controlled by the arguments `dimyx,eps,rule.eps` which are passed to `as.mask`.

Value

A function value table (object of class "fv") containing the estimated values of ρ (and confidence limits) for a sequence of values of Z. Also belongs to the class "rhohat" which has special methods for print, plot and predict.

Smooth estimates

Smooth estimators of $\rho(z)$ were proposed by Baddeley and Turner (2005) and Baddeley et al (2012). Similar estimators were proposed by Guan (2008) and in the literature on relative distributions (Handcock and Morris, 1999).

The estimated function $\rho(z)$ will be a smooth function of z.

The smooth estimation procedure involves computing several density estimates and combining them. The algorithm used to compute density estimates is determined by smoother:

- If smoother="kernel", the smoothing procedure is based on fixed-bandwidth kernel density estimation, performed by density.default.
- If smoother="local", the smoothing procedure is based on local likelihood density estimation, performed by locfit.

The argument method determines how the density estimates will be combined to obtain an estimate of $\rho(z)$:

- If method="ratio", then $\rho(z)$ is estimated by the ratio of two density estimates, The numerator is a (rescaled) density estimate obtained by smoothing the values $Z(y_i)$ of the covariate Z observed at the data points y_i. The denominator is a density estimate of the reference distribution of Z. See Baddeley et al (2012), equation (8). This is similar but not identical to an estimator proposed by Guan (2008).
- If method="reweight", then $\rho(z)$ is estimated by applying density estimation to the values $Z(y_i)$ of the covariate Z observed at the data points y_i, with weights inversely proportional to the reference density of Z. See Baddeley et al (2012), equation (9).
- If method="transform", the smoothing method is variable-bandwidth kernel smoothing, implemented by applying the Probability Integral Transform to the covariate values, yielding values in the range 0 to 1, then applying edge-corrected density estimation on the interval $[0, 1]$, and back-transforming. See Baddeley et al (2012), equation (10).

If horvitz=TRUE, then the calculations described above are modified by using Horvitz-Thompson weighting. The contribution to the numerator from each data point is weighted by the reciprocal of the baseline value or fitted intensity value at that data point; and a corresponding adjustment is made to the denominator.

Pointwise confidence intervals for the true value of $\rho(z)$ are also calculated for each z, and will be plotted as grey shading. The confidence intervals are derived using the central limit theorem, based on variance calculations which assume a Poisson point process. If positiveCI=FALSE, the lower limit of the confidence interval may sometimes be negative, because the confidence intervals are based on a normal approximation to the estimate of $\rho(z)$. If positiveCI=TRUE, the confidence limits are always positive, because the confidence interval is based on a normal approximation to the estimate of $\log(\rho(z))$. For consistency with earlier versions, the default is positiveCI=FALSE for smoother="kernel" and positiveCI=TRUE for smoother="local".

Monotone estimates

The nonparametric maximum likelihood estimator of a monotone function $\rho(z)$ was described by Sager (1982). This method assumes that $\rho(z)$ is either an increasing function of z, or a decreasing function of z. The estimated function will be a step function, increasing or decreasing as a function of z.

This estimator is chosen by specifying `smoother="increasing"` or `smoother="decreasing"`. The argument `method` is ignored this case.

To compute the estimate of $\rho(z)$, the algorithm first computes several primitive step-function estimates, and then takes the maximum of these primitive functions.

If `smoother="decreasing"`, each primitive step function takes the form $\rho(z) = \lambda$ when $z \leq t$, and $\rho(z) = 0$ when $z > t$, where and λ is a primitive estimate of intensity based on the data for $Z \leq t$. The jump location t will be the value of the covariate Z at one of the data points. The primitive estimate λ is the average intensity (number of points divided by area) for the region of space where the covariate value is less than or equal to t.

If `horvitz=TRUE`, then the calculations described above are modified by using Horvitz-Thompson weighting. The contribution to the numerator from each data point is weighted by the reciprocal of the baseline value or fitted intensity value at that data point; and a corresponding adjustment is made to the denominator.

Confidence intervals are not available for the monotone estimators.

Unimodal estimators

If `smoother="valley"` then we estimate a U-shaped function. A function $\rho(z)$ is U-shaped if it is decreasing when $z < z_0$ and increasing when $z > z_0$, where z_0 is called the critical value. The nonparametric maximum likelihood estimate of such a function can be computed by profiling over z_0. The algorithm considers all possible candidate values of the critical value z_0, and estimates the function $\rho(z)$ separately on the left and right of z_0 using the monotone estimators described above. These function estimates are combined into a single function, and the Poisson point process likelihood is computed. The optimal value of z_0 is the one which maximises the Poisson point process likelihood.

If `smoother="mountain"` then we estimate a function which has an inverted U shape. A function $\rho(z)$ is inverted-U-shaped if it is increasing when $z < z_0$ and decreasing when $z > z_0$. The nonparametric maximum likelihood estimate of such a function can be computed by profiling over z_0 using the same technique *mutatis mutandis*.

Confidence intervals are not available for the unimodal estimators.

Randomisation

By default, `rhohat` adds a small amount of random noise to the data. This is designed to suppress the effects of discretisation in pixel images.

This strategy means that `rhohat` does not produce exactly the same result when the computation is repeated. If you need the results to be exactly reproducible, set `jitter=FALSE`.

By default, the values of the covariate at the data points will be randomly perturbed by adding a small amount of noise using the function `jitter`. To reduce this effect, set `jitterfactor` to a number smaller than 1. To suppress this effect entirely, set `jitter=FALSE`.

Author(s)

Smoothing algorithm by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Ya-Mei Chang, Yong Song, and Rolf Turner <rolfturner@posteo.net>.

Nonparametric maximum likelihood algorithm by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

References

Baddeley, A., Chang, Y.-M., Song, Y. and Turner, R. (2012) Nonparametric estimation of the dependence of a point process on spatial covariates. *Statistics and Its Interface* **5** (2), 221–236.

Baddeley, A. and Turner, R. (2005) Modelling spatial point patterns in R. In: A. Baddeley, P. Gregori, J. Mateu, R. Stoica, and D. Stoyan, editors, *Case Studies in Spatial Point Pattern Modelling*, Lecture Notes in Statistics number 185. Pages 23–74. Springer-Verlag, New York, 2006. ISBN: 0-387-28311-0.

Baddeley, A. (2018) A statistical commentary on mineral prospectivity analysis. Chapter 2, pages 25–65 in *Handbook of Mathematical Geosciences: Fifty Years of IAMG*, edited by B.S. Daya Sagar, Q. Cheng and F.P. Agterberg. Springer, Berlin.

Guan, Y. (2008) On consistent nonparametric intensity estimation for inhomogeneous spatial point processes. *Journal of the American Statistical Association* **103**, 1238–1247.

Handcock, M.S. and Morris, M. (1999) *Relative Distribution Methods in the Social Sciences*. Springer, New York.

Sager, T.W. (1982) Nonparametric maximum likelihood estimation of spatial patterns. *Annals of Statistics* **10**, 1125–1136.

See Also

rho2hat, methods.rhohat, parres.

See ppm for a parametric method for the same problem.

Examples

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

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 <rolfturner@posteo.net>

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 <rolfturner@posteo.net>

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.ppm *Receiver Operating Characteristic*

Description

Computes the Receiver Operating Characteristic curve for a point pattern or a fitted point process model.

Usage

```
## 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").
...	Arguments passed to as.mask controlling the pixel resolution for calculations.

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 <rolfturner@posteo.net> 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

```
fit <- ppm(swedishpines ~ x+y)
plot(roc(fit))
```

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 <rolfturner@posteo.net> 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`, `update.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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net>

See Also

ppm, pairsat.family, Geyer, SatPiece, ppm.object

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 <rolfturner@posteo.net> 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.

Warning: new implementation for LGCP

The simulation algorithm for log-Gaussian Cox processes has been completely re-written in **spatstat.random** version 3.2-0 to avoid depending on the package **RandomFields** which is now defunct (and is sadly missed).

It is no longer possible to replicate results of simulate.kppm for log-Gaussian Cox processes that were obtained using previous versions of **spatstat.random**.

The current code for simulating log-Gaussian Cox processes is a new implementation and should be considered vulnerable to new bugs.

Author(s)

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

References

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

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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net>

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 <rolfturner@posteo.net>

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 <rolfturner@posteo.net>.

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 <- solapply(murchison, rescale, s=1000, unitname="km")
      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.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))
```

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 <rolfturner@posteo.net> 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))
```

---

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 <rolfturner@posteo.net> 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
```

`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 <rolfturner@posteo.net>.

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 <rolfturner@posteo.net>

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

# 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
```

---

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 <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

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

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)
```

---

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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>

References

Baddeley, A., Davies, T.M., Hazelton, M.L., Rakshit, S. and Turner, R. (2022) Fundamental problems in fitting spatial cluster process models. *Spatial Statistics* **52**, 100709. DOI: 10.1016/j.spasta.2022.100709

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 <rolfturner@posteo.net>

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
  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 Plenge 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

```
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

```
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))
```

`traj` *Extract trajectory of function evaluations*

Description

Extract the history of evaluations of the objective function performed when a cluster process model was fitted.

Usage

```
traj(object)
```

Arguments

`object`	Fitted cluster point process model (object of class `"kppm"`) or objective function surface (object of class `"objsurf"`).

Details

Under appropriate circumstances, the fitted model `object` contains the history of evaluations of the objective function that were performed by the optimisation algorithm. This history is extracted by `traj`.

The result is a data frame containing the input parameter values for the objective function, and the corresponding value of the objective function, that were considered by the optimisation algorithm. This data frame also belongs to the class `"traj"` which has methods for `plot`, `print` and other purposes.

Value

Either a data frame (belonging to class `"traj"`) or `NULL`.

Author(s)

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

See Also

methods.traj

Examples

```
fit <- kppm(redwood, trajectory=TRUE)
h <- traj(fit)
```

---

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 <rolfturner@posteo.net>

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 <rolfturner@posteo.net>

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
```

---

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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net> 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 <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

`update.dppm` *Update a Fitted Determinantal Point Process Model*

Description

`update` method for class `"dppm"`.

Usage

```
 ## S3 method for class 'dppm'
update(object, ..., evaluate=TRUE,
                      envir=environment(terms(object)))
```

Arguments

object	Fitted determinantal point process model. An object of class "dppm", obtained from dppm.
...	Arguments passed to dppm.
evaluate	Logical value indicating whether to return the updated fitted model (evaluate=TRUE, the default) or just the updated call to dppm (evaluate=FALSE).
envir	Environment in which to re-evaluate the call to dppm.

Details

object should be a fitted determinantal point process model, obtained from the model-fitting function dppm. 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 dppm.

Value

Another fitted cluster point process model (object of class "dppm".

Author(s)

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

See Also

dppm, plot.dppm, predict.dppm, simulate.dppm, methods.dppm.

Examples

```
fit <- dppm(swedishpines ~ x + y, dppGauss, method="c")
fitx <- update(fit, ~x)
fit2 <- update(fit, flipxy(swedishpines))
```

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 <rolfturner@posteo.net> 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 kppm.

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 <rolfturner@posteo.net>

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 <rolfturner@posteo.net>

Examples

```
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)
```

update.rppm *Update a Recursively Partitioned Point Process Model*

Description

update method for class "rppm".

Usage

```
 ## S3 method for class 'rppm'
update(object, ..., envir=environment(terms(object)))
```

Arguments

object	Fitted recursively partitioned point process model. An object of class "rppm", obtained from rppm.
...	Arguments passed to rppm.
envir	Environment in which to re-evaluate the call to rppm.

Details

object should be a fitted recursively partitioned point process model, obtained from the model-fitting function rppm.

The model will be updated according to the new arguments provided.

Value

Another fitted recursively partitioned point process model (object of class "rppm".

Author(s)

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

See Also

rppm.

Examples

```
fit <- rppm(nztrees ~ x)
newfit <- update(fit, . ~ x + y)
```

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 <rolfturner@posteo.net>

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 <rolfturner@posteo.net>

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 <rolfturner@posteo.net>

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 <rolfturner@posteo.net> 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)
```

---

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, relative=FALSE)
```

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.
relative	Logical value specifying whether to divide the variance by the mean value.

Details

The function varcount 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.

If relative=FALSE (the default), the result is the variance. If relative=TRUE, the result is the variance divided by the mean, which is the overdispersion index (equal to 1 if the number of points has a Poisson distribution).

The model should be a fitted point process model (object of class "ppm", "kppm" or "dppm").

- If B is a window, varcount 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, varcount 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,...) then varcount 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 <rolfturner@posteo.net> 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"),
          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 <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

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

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 = c("vcov", "corr", "fisher"),
                    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.
- The calculations depend on the type of dummy pattern used when the model was fitted:
  - currently only the dummy types "stratrand" (the default), "binomial" and "poisson" as generated by quadscheme.logi are supported.
  - For other dummy types the behavior depends on the argument logi.action. If logi.action="fatal" an error is produced. Otherwise, for dummy 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 result of the calculation is **random** (i.e. not deterministic) when dummy type is "stratrand" (the default) because some of the variance terms are estimated by random sampling. This can be avoided by specifying dummytype='poisson' or dummytype='binomial' in the call to ppm when the model is fitted.

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 <rolfturner@posteo.net>. New code for stationary Gibbs point processes was generously contributed by Ege Rubak <rubak@math.aau.dk> and Jean-François Coeurjolly. New code for generic Gibbs process written by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>. New code for logistic method written 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))

  try(v <- vcov(m))

# 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 `<rolfturner@posteo.net>` .

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, ...)
```

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 <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

Window, Window.ppp, Window.psp.

owin.object

Examples

```
A <- ppm(cells ~ 1)
Window(A)
```

---

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 <rolfturner@posteo.net> 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)
```

`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)
```

Index