

Package ‘spatstat’

June 29, 2012

Version 1.28-1

Date 2012-06-29

Title Spatial Point Pattern analysis, model-fitting, simulation, tests

Author Adrian Baddeley <Adrian.Baddeley@csiro.au>

and Rolf Turner <r.turner@auckland.ac.nz> with substantial contributions of code by Kasper Klitgaard Berthelsen; Abdollah Jalilian; Marie-Colette van Lieshout; Ege Rubak; Dominic Schuhmacher; and Rasmus Waagepetersen. Additional contributions by Q.W. Ang; S. Azaele; C. Beale; R. Bernhardt; B. Biggerstaff; R. Bivand; F. Bonneau; J. Burgos; S. Byers; Y.M. Chang; J.B. Chen; I. Chernayavsky; Y.C. Chin; B. Christensen; J.-F. Coeurjolly; M. de la Cruz; P. Dalgard; P.J. Diggle; I. Dryden; S. Eglen; N. Funwi-Gabga; A. Gault; M. Genton; P. Grabarnik; C. Graf; J. Franklin; U. Hahn; A. Hardegen; M. Hering; M.B. Hansen; M. Hazelton; J. Heikkinen; K. Hornik; R. Ihaka; A. Jammalamadaka; R. John-Chandran; D. Johnson; M. Kuhn; J. Laake; T. Lawrence; R.A. Lamb; J. Lee; G.P. Leser; B. Madin; R. Mark; J. Mateu; P. McCullagh; U. Mehlig; S. Meyer; X.C. Mi; J. Moller; L.S. Nielsen; F. Nunes; J. Oehlschlaegel; T. Onkelinx; E. Parilov; J. Picka; S. Protsiv; A. Raftery; M. Reiter; T.O. Richardson; B.D. Ripley; B. Rowlingson; J. Rudge; F. Safavimanesh; A. Sarkka; K. Schladitz; B.T. Scott; G.C. Shen; V. Shcherbakov; I.-M. Sintorn; Y. Song; M. Spiess; M. Stevenson; K. Stucki; M. Sumner; P. Surovy; B. Taylor; B. Turlach; A. van Burgel; T. Verbeke; A. Villers; H. Wang; H. Wendrock; J. Wild; S. Wong and M.E. Zamboni.

Maintainer Adrian Baddeley <Adrian.Baddeley@csiro.au>

Depends R (>= 2.14.0), stats, graphics, utils, mgcv, deldir (>= 0.0-10)

Suggests gpclib, sm, maptools, gsl, locfit, spatial, rpanel, tkrplot, scatterplot3d, RandomFields (>= 2.0)

Description A package for analysing spatial data, mainly Spatial Point

Patterns, including multitype/marked points and spatial covariates, in any two-dimensional spatial region. Also supports three-dimensional point patterns, and space-time point patterns in any number of dimensions.

Contains over 1000 functions for plotting spatial data, exploratory data analysis, model-fitting, simulation, spatial sampling, model diagnostics, and formal inference.

Data types include point patterns, line segment patterns, spatial windows, pixel images and tessellations.

Exploratory methods include K-functions, nearest neighbour distance and empty space statistics, Fry plots, pair correlation function, kernel smoothed intensity, relative risk estimation with cross-validated bandwidth selection, mark correlation functions, segregation indices, mark dependence diagnostics etc.

Point process models can be fitted to point pattern data using functions ppm, kppm, slrm similar to glm. Models may include dependence on covariates, interpoint interaction, cluster formation and dependence on marks. Fitted models can be simulated automatically.
 Also provides facilities for formal inference (such as chi-squared tests) and model diagnostics (including simulation envelopes, residuals, residual plots and Q-Q plots).

License GPL (>= 2)

URL <http://www.spatstat.org>

LazyData true

LazyLoad true

ByteCompile true

R topics documented:

spatstat-package	14
adaptive.density	34
affine	35
affine.im	36
affine.owin	37
affine.ppp	38
affine.psp	39
allstats	40
alltypes	41
amacrine	44
anemones	45
angles.psp	46
anova.lppm	47
anova.ppm	48
anova.slrm	50
ants	51
append.psp	53
applynbd	54
area.owin	57
areaGain	58
AreaInter	59
areaLoss	62
as.box3	63
as.data.frame.hyperframe	64
as.data.frame.im	65
as.data.frame.ppp	66
as.data.frame.psp	67
as.function.fv	68
as.hyperframe	69
as.hyperframe.ppx	70
as.im	71
as.interact	74
as.mask	75
as.mask.psp	76
as.matrix.im	77

as.matrix.owin	78
as.owin	79
as.polygonal	82
as.ppp	83
as.psp	85
as.rectangle	88
as.tess	89
BadGey	90
bdist.pixels	92
bdist.points	93
bdist.tiles	94
bei	95
bermantest	96
betacells	98
bind.fv	99
blur	101
border	102
bounding.box	103
bounding.box.xy	104
box3	105
boxx	106
bramblecanes	107
bronzefilter	108
bw.diggle	109
bw.relrisk	110
bw.scott	112
bw.smoothppp	113
bw.stoyan	114
by.im	115
by.ppp	116
cauchy.estK	118
cauchy.estpcf	120
cbind.hyperframe	122
cells	123
centroid.owin	123
chicago	125
chop.tess	126
chorley	127
circumradius	128
clarkevans	129
clarkevans.test	131
clf.test	132
clickjoin	134
clickpoly	135
clickppp	136
clip.inflne	137
closing	138
coef.ppm	139
coef.slrm	141
collapse.fv	142
colourmap	143
colourtools	145

commonGrid	146
compareFit	147
compatible	148
compatible.fasp	149
compatible.fv	150
compatible.im	151
complement.owin	151
concatxy	153
connected	154
contour.im	155
contour.listof	156
convexhull	157
convexhull.xy	158
convolve.im	159
coords	160
copper	161
corners	163
crossdist	164
crossdist.default	165
crossdist.ppp3	166
crossdist.ppp	167
crossdist.ppx	168
crossdist.psp	169
crossing.psp	170
cut.im	171
cut.ppp	172
data.ppm	174
default.dummy	175
default.expand	176
default.rmhcontrol	177
delaunay	178
deltametric	179
demopat	180
density.ppp	181
density.psp	184
density.splitppp	185
dfbetas.ppm	186
diagnose.ppm	188
diameter	192
diameter.box3	193
diameter.boxx	194
diameter.owin	195
DiggleGatesStibbard	196
DiggleGratton	197
dilated.areas	199
dilation	200
dirichlet	201
dirichlet.weights	202
disc	203
discrepancyarea	204
discretise	205
distfun	206

distmap	207
distmap.owin	208
distmap.ppp	210
distmap.psp	211
dummify	212
dummy.ppm	213
duplicated.ppp	214
edges2triangles	215
eem	216
effectfun	217
Emark	219
endpoints.psp	221
envelope	222
envelope.envelope	229
envelope.lpp	231
envelope.pp3	233
eroded.areas	235
erosion	236
eval.fasp	237
eval.fv	239
eval.im	240
ewcdf	242
exactMPLEstrauss	243
expand.owin	244
Extract.fasp	245
Extract.fv	246
Extract.im	247
Extract.listof	249
Extract.lpp	250
Extract.msr	251
Extract.ppp	252
Extract.ppx	254
Extract.psp	255
Extract.quad	257
Extract.splitppp	258
Extract.tess	259
F3est	260
fasp.object	262
Fest	263
Fiksel	267
finpines	268
fitin.ppm	269
fitted.ppm	271
fitted.slrn	272
flipxy	273
flu	274
formula.ppm	276
fryplot	277
fv	279
fv.object	281
G3est	282
ganglia	283

Gcom	284
Gcross	287
Gdot	291
Gest	293
Geyer	296
Gfox	298
Gmulti	299
gorillas	302
gpc2owin	303
Gres	304
gridcentres	306
gridweights	307
hamster	308
Hardcore	309
harmonic	310
harmonise.im	312
heather	313
Hest	314
hist.im	316
humberside	317
hyperframe	318
identify.ppp	320
identify.psp	321
idw	322
Iest	323
im	325
im.object	327
imcov	328
incircle	329
inflne	330
influence.ppm	332
inforder.family	333
inside.owin	334
integral.im	335
intensity	336
intensity.ppm	337
intensity.ppp	338
interp.im	339
intersect.owin	340
intersect.tess	342
iplot	343
ippm	344
is.convex	346
is.empty	347
is.im	348
is.marked	348
is.marked.ppm	349
is.marked.ppp	350
is.multitype	351
is.multitype.ppm	352
is.multitype.ppp	354
is.owin	355

is.ppm	356
is.ppp	356
is.rectangle	357
is.stationary	358
is.subset.owin	359
istat	360
japanesepines	361
Jcross	362
Jdot	364
Jest	367
Jmulti	370
K3est	372
kaplan.meier	373
Kcom	374
Kcross	378
Kcross.inhom	380
Kdot	383
Kdot.inhom	386
Kest	389
Kest.fft	393
Kinhom	394
km.rs	398
Kmeasure	400
Kmodel	402
Kmulti	403
Kmulti.inhom	406
kppm	409
Kres	411
Kscaled	412
kstest.ppm	415
LambertW	418
lansing	419
latest.news	420
layered	421
Lcross	422
Lcross.inhom	423
Ldot	425
Ldot.inhom	426
lengths.psp	428
LennardJones	429
Lest	430
letterR	432
levelset	432
leverage.ppm	434
lgcp.estK	435
lgcp.estpcf	438
lineardisc	441
linearK	442
linearKinhom	443
linearpcf	444
linearpcfinhom	446
Linhom	447

linim	448
linnet	450
localK	451
localKinhom	453
localpcf	455
logLik.ppm	457
logLik.slrn	458
lohboot	459
longleaf	461
lpp	462
lppm	463
lurking	464
lut	467
markconnect	468
markcorr	470
markcorrint	474
marks	476
marks.psp	478
markstat	479
marktable	480
markvario	481
matchingdist	483
matclust.estK	485
matclust.estpcf	487
mean.im	489
methods.box3	490
methods.boxx	491
methods.distfun	492
methods.kppm	494
methods.linnet	495
methods.lpp	496
methods.lppm	498
methods.pp3	499
methods.ppx	500
methods.rho2hat	501
methods.rhohat	502
methods.slrn	503
methods.units	504
midpoints.psp	505
mincontrast	506
miplot	508
model.depends	509
model.frame.ppm	511
model.images	512
model.matrix.ppm	513
msr	515
MultiHard	517
multiplicity.ppp	518
MultiStrauss	519
MultiStraussHard	520
murchison	521
nbfires	523

nearest.raster.point	526
nearestsegment	527
nnclean	528
nncorr	529
nncross	532
nndist	534
nndist.pp3	536
nndist.ppx	537
nndist.psp	539
nnwhich	540
nnwhich.pp3	542
nnwhich.ppx	543
npfun	545
npoints	546
nsegments	547
nztrees	548
opening	548
Ord	550
ord.family	551
OrdThresh	552
osteo	553
owin	555
owin.object	557
pairdist	558
pairdist.default	559
pairdist.Ipp	560
pairdist.pp3	561
pairdist.ppp	562
pairdist.ppx	563
pairdist.psp	564
PairPiece	565
pairs.im	567
pairsat.family	568
Pairwise	569
pairwise.family	571
pcf	571
pcf.fasp	573
pcf.fv	575
pcf.ppp	576
pcf3est	578
pcfcross	580
pcfcross.inhom	582
pcfdot	584
pcfdot.inhom	585
pcfinhom	587
perimeter	589
periodify	590
persp.im	591
pixellate	593
pixellate.owin	594
pixellate.ppp	595
pixellate.psp	596

pixelquad	597
plot.bermantest	599
plot.colourmap	600
plot.envelope	601
plot.fasp	602
plot.fv	604
plot.hyperframe	607
plot.im	608
plot.influence.ppm	611
plot.kppm	612
plot.kstest	613
plot.layered	614
plot.leverage.ppm	615
plot.linin	616
plot.linnet	617
plot.listof	618
plot.msr	620
plot.own	621
plot.plotppm	623
plot.pp3	625
plot.ppm	626
plot.ppp	628
plot.psp	631
plot.quad	632
plot.slrm	634
plot.splitppp	635
plot.tess	636
pointsOnLines	637
Poisson	638
ponderosa	639
pool	640
pool.envelope	641
pool.fasp	642
pool.rat	643
pp3	644
ppm	645
ppm.object	652
ppp	655
ppp.object	657
pppdist	659
pppmatching	662
pppmatching.object	663
ppx	664
predict.kppm	666
predict.lppm	667
predict.ppm	668
predict.slrm	672
print.im	673
print.own	674
print.ppm	675
print.ppp	676
print.psp	677

print.quad	677
profilepl	678
progressreport	680
project.ppm	681
project2segment	682
psp	684
psp.object	685
psst	686
psstA	688
psstG	691
qqplot.ppm	693
quad.object	697
quad.ppm	698
quadrat.test	700
quadrat.test.splitppp	702
quadratcount	703
quadratresample	705
quadrats	706
quadscheme	708
quantile.im	710
raster.x	711
rat	712
rCauchy	713
rcell	714
rDGS	715
rDiggleGratton	717
reach	718
reduced.sample	720
redwood	721
redwoodfull	722
reflect	723
relrisk	724
Replace.im	726
rescale	727
rescale.im	728
rescale.owin	729
rescale.ppp	730
rescale.psp	731
rescue.rectangle	732
residuals.ppm	733
residualspaper	736
rGaussPoisson	737
rgbim	738
rHardcore	739
rho2hat	740
rhohat	742
ripras	744
rjitter	746
rknn	747
rlabel	748
rLGCP	749
rlinegrid	751

rMatClust	752
rMaternI	753
rMaternII	754
rmh	755
rmh.default	757
rmh.ppm	765
rmhcontrol	769
rmhexpand	772
rmhmodel	774
rmhmodel.default	775
rmhmodel.list	781
rmhmodel.ppm	783
rmhstart	785
rMosaicField	787
rMosaicSet	788
rmpoint	789
rmpoispp	792
rNeymanScott	795
rotate	797
rotate.owin	798
rotate.ppp	799
rotate.psp	800
rpoint	801
rpoisline	802
rpoislinetess	803
rpoislpp	804
rpoispp	805
rpoispp3	806
rpoisppOnLines	807
rpoisppx	809
rPoissonCluster	810
rshift	811
rshift.ppp	812
rshift.psp	815
rshift.splitppp	816
rSSI	817
rstrat	818
rStrauss	819
rStraussHard	821
rsyst	823
rthin	824
rThomas	825
runifdisc	826
runiflpp	827
runifpoint	828
runifpoint3	829
runifpointOnLines	830
runifpointx	831
rVarGamma	832
SatPiece	833
Saturated	835
scalardilate	836

scaletointerval	837
scan.test	838
scanpp	840
selfcrossing.psp	841
setcov	842
shapley	843
sharpen	844
shift	845
shift.im	846
shift.owin	847
shift.ppp	848
shift.psp	849
simdat	850
simplenet	851
simplify.owin	851
simulate.kppm	852
simulate.ppm	853
simulate.slrn	855
slrn	856
smooth.fv	858
smooth.msr	859
smooth.ppp	860
Softcore	862
solutionset	864
spatstat.options	865
split.im	868
split.ppp	869
split.ppx	871
spokes	873
spruces	875
square	876
stieltjes	877
Strauss	878
StraussHard	879
suffstat	881
summary.im	883
summary.listof	884
summary.owin	885
summary.ppm	886
summary.ppp	887
summary.psp	888
summary.quad	889
summary.splitppp	890
sumouter	891
superimpose	892
swedishpines	894
tess	895
thomas.estK	897
thomas.estpcf	899
tile.areas	901
tiles	902
transect.im	903

trim.rectangle	904
triplet.family	905
Triplets	905
Tstat	907
union.quad	908
unique.ppp	909
unitname	910
unmark	912
unnormdensity	913
update.kppm	914
update.ppm	915
update.rmhcontrol	917
urkiola	918
valid.ppm	919
varblock	920
vargamma.estK	921
vargamma.estpcf	923
vcov.kppm	925
vcov.ppm	926
vertices	929
volume	930
which.max.im	931
whist	932
will.expand	933
with.fv	934
with.hyperframe	935
zapsmall.im	936

Index	938
--------------	------------

Description

This is a summary of the features of **spatstat**, a package in R for the statistical analysis of spatial point patterns.

Details

spatstat is a package for the statistical analysis of spatial data. Currently, it deals mainly with the analysis of patterns of points in the plane. The points may carry auxiliary data ('marks'), and the spatial region in which the points were recorded may have arbitrary shape.

The package supports

- creation, manipulation and plotting of point patterns
- exploratory data analysis
- simulation of point process models
- parametric model-fitting
- hypothesis tests and model diagnostics

Apart from two-dimensional point patterns and point processes, **spatstat** also supports patterns of line segments in two dimensions, point patterns in three dimensions, and multidimensional space-time point patterns. It also supports spatial tessellations and random sets.

The package can fit several types of point process models to a point pattern dataset:

- Poisson point process models (by Berman-Turner approximate maximum likelihood or by spatial logistic regression)
- Gibbs/Markov point process models (by Baddeley-Turner approximate maximum pseudolikelihood or Huang-Ogata approximate maximum likelihood)
- Cox/cluster process models (by Waagepetersen's two-step fitting procedure and minimum contrast)

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

Getting Started

For a quick introduction to **spatstat**, see the package vignette *Getting started with spatstat* installed with **spatstat**. (To see this document online, start R, type `help.start()` to open the help browser, and navigate to Packages > **spatstat** > Vignettes).

For a complete 2-day course on using **spatstat**, see the workshop notes by Baddeley (2010), available on the internet.

Type `demo(spatstat)` for a demonstration of the package's capabilities. Type `demo(data)` to see all the datasets available in the package.

For information about handling data in **shapefiles**, see the Vignette *Handling shapefiles in the spatstat package* installed with **spatstat**.

To learn about spatial point process methods, see the short book by Diggle (2003) and the handbook Gelfand et al (2010).

Updates

New versions of **spatstat** are produced about once a month. Users are advised to update their installation of **spatstat** regularly.

Type `latest.news()` to read the news documentation about changes to the current installed version of **spatstat**. Type `news(package="spatstat")` to read news documentation about all previous versions of the package.

FUNCTIONS AND DATASETS

Following is a summary of the main functions and datasets in the **spatstat** package. Alternatively an alphabetical list of all functions and datasets is available by typing `library(help=spatstat)`.

For further information on any of these, type `help(name)` where name is the name of the function or dataset.

CONTENTS:

- I. Creating and manipulating data
- II. Exploratory Data Analysis
- III. Model fitting (cluster models)
- IV. Model fitting (Poisson and Gibbs models)

- V. Model fitting (spatial logistic regression)
- VI. Simulation
- VII. Tests and diagnostics
- VIII. Documentation

I. CREATING AND MANIPULATING DATA

Types of spatial data:

The main types of spatial data supported by **spatstat** are:

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

To create a point pattern:

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

To simulate a random point pattern:

<code>runifpoint</code>	generate n independent uniform random points
<code>rpoint</code>	generate n independent random points
<code>rmpoint</code>	generate n independent multitype random points
<code>rpoispp</code>	simulate the (in)homogeneous Poisson point process
<code>rmipoispp</code>	simulate the (in)homogeneous multitype Poisson point process
<code>runifdisc</code>	generate n independent uniform random points in disc
<code>rstrat</code>	stratified random sample of points
<code>rsyst</code>	systematic random sample of points
<code>rjitter</code>	apply random displacements to points in a pattern
<code>rMaternI</code>	simulate the Mat'ern Model I inhibition process
<code>rMaternII</code>	simulate the Mat'ern Model II inhibition process
<code>rSSI</code>	simulate Simple Sequential Inhibition process
<code>rStrauss</code>	simulate Strauss process (perfect simulation)
<code>rHardcore</code>	simulate Hard Core process (perfect simulation)
<code>rDiggleGratton</code>	simulate Diggle-Gratton process (perfect simulation)
<code>rDGS</code>	simulate Diggle-Gates-Stibbard process (perfect simulation)
<code>rNeymanScott</code>	simulate a general Neyman-Scott process
<code>rPoissonCluster</code>	simulate a general Neyman-Scott process
<code>rNeymanScott</code>	simulate a general Neyman-Scott process

<code>rMatClust</code>	simulate the Mat'ern Cluster process
<code>rThomas</code>	simulate the Thomas process
<code>rGaussPoisson</code>	simulate the Gauss-Poisson cluster process
<code>rCauchy</code>	simulate Neyman-Scott Cauchy cluster process
<code>rVarGamma</code>	simulate Neyman-Scott Variance Gamma cluster process
<code>rthin</code>	random thinning
<code>rcell</code>	simulate the Baddeley-Silverman cell process
<code>rmh</code>	simulate Gibbs point process using Metropolis-Hastings
<code>simulate.ppm</code>	simulate Gibbs point process using Metropolis-Hastings
<code>runifpointOnLines</code>	generate n random points along specified line segments
<code>rpoisppOnLines</code>	generate Poisson random points along specified line segments

To randomly change an existing point pattern:

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

Standard point pattern datasets:

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

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

<code>amacrine</code>	Austin Hughes' rabbit amacrine cells
<code>anemones</code>	Upton-Fingleton sea anemones data
<code>ants</code>	Harkness-Isham ant nests data
<code>bei</code>	Tropical rainforest trees
<code>betacells</code>	Waessle et al. cat retinal ganglia data
<code>bramblecanes</code>	Bramble Canes data
<code>bronzefilter</code>	Bronze Filter Section data
<code>cells</code>	Crick-Ripley biological cells data
<code>chicago</code>	Chicago street crimes
<code>chorley</code>	Chorley-Ribble cancer data
<code>copper</code>	Berman-Huntington copper deposits data
<code>demopat</code>	Synthetic point pattern
<code>finpines</code>	Finnish Pines data
<code>flu</code>	Influenza virus proteins
<code>gorillas</code>	Gorilla nest sites
<code>hamster</code>	Aherne's hamster tumour data
<code>humberside</code>	North Humberside childhood leukaemia data
<code>japanesepines</code>	Japanese Pines data
<code>lansing</code>	Lansing Woods data
<code>longleaf</code>	Longleaf Pines data
<code>murchison</code>	Murchison gold deposits
<code>nbfires</code>	New Brunswick fires data
<code>nztrees</code>	Mark-Esler-Ripley trees data
<code>osteo</code>	Osteocyte lacunae (3D, replicated)
<code>ponderosa</code>	Getis-Franklin ponderosa pine trees data
<code>redwood</code>	Strauss-Ripley redwood saplings data

<code>redwoodfull</code>	Strauss redwood saplings data (full set)
<code>residualspaper</code>	Data from Baddeley et al (2005)
<code>shapley</code>	Galaxies in an astronomical survey
<code>simdat</code>	Simulated point pattern (inhomogeneous, with interaction)
<code>spruces</code>	Spruce trees in Saxonia
<code>swedishpines</code>	Strand-Ripley swedish pines data
<code>urkiola</code>	Urkiola Woods data

To manipulate a point pattern:

<code>plot.ppp</code>	plot a point pattern (e.g. <code>plot(X)</code>)
<code>iplot</code>	plot a point pattern interactively
<code>[.ppp</code>	extract or replace a subset of a point pattern <code>pp[subset]</code> or <code>pp[subwindow]</code>
<code>superimpose</code>	combine several point patterns
<code>by.ppp</code>	apply a function to sub-patterns of a point pattern
<code>cut.ppp</code>	classify the points in a point pattern
<code>unmark</code>	remove marks
<code>npoints</code>	count the number of points
<code>coords</code>	extract coordinates, change coordinates
<code>marks</code>	extract marks, change marks or attach marks
<code>split.ppp</code>	divide pattern into sub-patterns
<code>rotate</code>	rotate pattern
<code>shift</code>	translate pattern
<code>flipxy</code>	swap x and y coordinates
<code>reflect</code>	reflect in the origin
<code>periodify</code>	make several translated copies
<code>affine</code>	apply affine transformation
<code>scalardilate</code>	apply scalar dilation
<code>density.ppp</code>	kernel smoothing of point pattern
<code>smooth.ppp</code>	smooth the marks attached to points
<code>sharpen.ppp</code>	data sharpening
<code>identify.ppp</code>	interactively identify points
<code>unique.ppp</code>	remove duplicate points
<code>duplicated.ppp</code>	determine which points are duplicates
<code>dirichlet</code>	compute Dirichlet-Voronoi tessellation
<code>delaunay</code>	compute Delaunay triangulation
<code>convexhull</code>	compute convex hull
<code>discretise</code>	discretise coordinates
<code>pixelate.ppp</code>	approximate point pattern by pixel image
<code>as.im.ppp</code>	approximate point pattern by pixel image

See `spatstat.options` to control plotting behaviour.

To create a window:

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

<code>owin</code>	Create a window object
<code>owin(xlim, ylim)</code>	for rectangular window
<code>owin(poly)</code>	for polygonal window
<code>owin(mask)</code>	for binary image window

`as.owin` Convert other data to a window object

<code>square</code>	make a square window
<code>disc</code>	make a circular window
<code>ripras</code>	Ripley-Rasson estimator of window, given only the points
<code>convexhull</code>	compute convex hull of something
<code>letterR</code>	polygonal window in the shape of the R logo

To manipulate a window:

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

Digital approximations:

<code>as.mask</code>	Make a discrete pixel approximation of a given window
<code>as.im.owin</code>	convert window to pixel image
<code>pixellate.owin</code>	convert window to pixel image
<code>commonGrid</code>	find common pixel grid for windows
<code>nearest.raster.point</code>	map continuous coordinates to raster locations
<code>raster.x</code>	raster x coordinates
<code>raster.y</code>	raster y coordinates
<code>as.polygonal</code>	convert pixel mask to polygonal window

See `spatstat.options` to control the approximation

Geometrical computations with windows:

<code>intersect.owin</code>	intersection of two windows
<code>union.owin</code>	union of two windows
<code>setminus.owin</code>	set subtraction of two windows
<code>inside.owin</code>	determine whether a point is inside a window
<code>area.owin</code>	compute area
<code>perimeter</code>	compute perimeter length
<code>diameter.owin</code>	compute diameter
<code>incircle</code>	find largest circle inside a window
<code>connected</code>	find connected components of window
<code>eroded.areas</code>	compute areas of eroded windows
<code>dilated.areas</code>	compute areas of dilated windows
<code>bdist.points</code>	compute distances from data points to window boundary
<code>bdist.pixels</code>	compute distances from all pixels to window boundary

<code>bdist.tiles</code>	boundary distance for each tile in tessellation
<code>distmap.owin</code>	distance transform image
<code>distfun.owin</code>	distance transform
<code>centroid.owin</code>	compute centroid (centre of mass) of window
<code>is.subset.owin</code>	determine whether one window contains another
<code>is.convex</code>	determine whether a window is convex
<code>convexhull</code>	compute convex hull
<code>as.mask</code>	pixel approximation of window
<code>as.polygonal</code>	polygonal approximation of window
<code>is.rectangle</code>	test whether window is a rectangle
<code>is.polygonal</code>	test whether window is polygonal
<code>is.mask</code>	test whether window is a mask
<code>setcov</code>	spatial covariance function of window

Pixel images: An object of class "`im`" represents a pixel image. Such objects are returned by some of the functions in **spatstat** including `Kmeasure`, `setcov` and `density.ppp`.

<code>im</code>	create a pixel image
<code>as.im</code>	convert other data to a pixel image
<code>pixellate</code>	convert other data to a pixel image
<code>as.matrix.im</code>	convert pixel image to matrix
<code>as.data.frame.im</code>	convert pixel image to data frame
<code>plot.im</code>	plot a pixel image on screen as a digital image
<code>contour.im</code>	draw contours of a pixel image
<code>persp.im</code>	draw perspective plot of a pixel image
<code>rgbim</code>	create colour-valued pixel image
<code>hsvim</code>	create colour-valued pixel image
<code>[.im</code>	extract a subset of a pixel image
<code>[<.im</code>	replace a subset of a pixel image
<code>shift.im</code>	apply vector shift to pixel image
<code>X</code>	print very basic information about image <code>X</code>
<code>summary(X)</code>	summary of image <code>X</code>
<code>hist.im</code>	histogram of image
<code>mean.im</code>	mean pixel value of image
<code>integral.im</code>	integral of pixel values
<code>quantile.im</code>	quantiles of image
<code>cut.im</code>	convert numeric image to factor image
<code>is.im</code>	test whether an object is a pixel image
<code>interp.im</code>	interpolate a pixel image
<code>blur</code>	apply Gaussian blur to image
<code>connected</code>	find connected components
<code>compatible.im</code>	test whether two images have compatible dimensions
<code>harmonise.im</code>	make images compatible
<code>commonGrid</code>	find a common pixel grid for images
<code>eval.im</code>	evaluate any expression involving images
<code>scaletointerval</code>	rescale pixel values
<code>zapsmall.im</code>	set very small pixel values to zero
<code>levelset</code>	level set of an image
<code>solutionset</code>	region where an expression is true
<code>imcov</code>	spatial covariance function of image
<code>convolve.im</code>	spatial convolution of images
<code>transect.im</code>	line transect of image

Line segment patterns

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

<code>psp</code>	create a line segment pattern
<code>as.psp</code>	convert other data into a line segment pattern
<code>is.psp</code>	determine whether a dataset has class "psp"
<code>plot.psp</code>	plot a line segment pattern
<code>print.psp</code>	print basic information
<code>summary.psp</code>	print summary information
<code>[.psp</code>	extract a subset of a line segment pattern
<code>as.data.frame.psp</code>	convert line segment pattern to data frame
<code>marks.psp</code>	extract marks of line segments
<code>marks<- .psp</code>	assign new marks to line segments
<code>unmark.psp</code>	delete marks from line segments
<code>midpoints.psp</code>	compute the midpoints of line segments
<code>endpoints.psp</code>	extract the endpoints of line segments
<code>lengths.psp</code>	compute the lengths of line segments
<code>angles.psp</code>	compute the orientation angles of line segments
<code>superimpose</code>	combine several line segment patterns
<code>flipxy</code>	swap x and y coordinates
<code>rotate.psp</code>	rotate a line segment pattern
<code>shift.psp</code>	shift a line segment pattern
<code>periodify</code>	make several shifted copies
<code>affine.psp</code>	apply an affine transformation
<code>pixellate.psp</code>	approximate line segment pattern by pixel image
<code>as.mask.psp</code>	approximate line segment pattern by binary mask
<code>distmap.psp</code>	compute the distance map of a line segment pattern
<code>distfun.psp</code>	compute the distance map of a line segment pattern
<code>density.psp</code>	kernel smoothing of line segments
<code>selfcrossing.psp</code>	find crossing points between line segments
<code>crossing.psp</code>	find crossing points between two line segment patterns
<code>nncross</code>	find distance to nearest line segment from a given point
<code>nearestsegment</code>	find line segment closest to a given point
<code>project2segment</code>	find location along a line segment closest to a given point
<code>pointsOnLines</code>	generate points evenly spaced along line segment
<code>rpoisline</code>	generate a realisation of the Poisson line process inside a window
<code>rlinegrid</code>	generate a random array of parallel lines through a window

Tessellations

An object of class "tess" represents a tessellation.

<code>tess</code>	create a tessellation
<code>quadrats</code>	create a tessellation of rectangles
<code>as.tess</code>	convert other data to a tessellation
<code>plot.tess</code>	plot a tessellation
<code>tiles</code>	extract all the tiles of a tessellation
<code>[.tess</code>	extract some tiles of a tessellation
<code>[<-.tess</code>	change some tiles of a tessellation
<code>intersect.tess</code>	intersect two tessellations or restrict a tessellation to a window
<code>chop.tess</code>	subdivide a tessellation by a line
<code>dirichlet</code>	compute Dirichlet-Voronoi tessellation of points

<code>delaunay</code>	compute Delaunay triangulation of points
<code>rpoislinetess</code>	generate tessellation using Poisson line process
<code>tile.areas</code>	area of each tile in tessellation
<code>bdist.tiles</code>	boundary distance for each tile in tessellation

Three-dimensional point patterns

An object of class "pp3" represents a three-dimensional point pattern in a rectangular box. The box is represented by an object of class "box3".

<code>pp3</code>	create a 3-D point pattern
<code>plot.pp3</code>	plot a 3-D point pattern
<code>coords</code>	extract coordinates
<code>as.hyperframe</code>	extract coordinates
<code>unitname.pp3</code>	name of unit of length
<code>npoints</code>	count the number of points
<code>runifpoint3</code>	generate uniform random points in 3-D
<code>rpoispp3</code>	generate Poisson random points in 3-D
<code>envelope.pp3</code>	generate simulation envelopes for 3-D pattern
<code>box3</code>	create a 3-D rectangular box
<code>as.box3</code>	convert data to 3-D rectangular box
<code>unitname.box3</code>	name of unit of length
<code>diameter.box3</code>	diameter of box
<code>volume.box3</code>	volume of box
<code>shortside.box3</code>	shortest side of box
<code>eroded.volumes</code>	volumes of erosions of box

Multi-dimensional space-time point patterns

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

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

Point patterns on a linear network

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

<code>linnet</code>	create a linear network
<code>clickjoin</code>	interactively join vertices in network
<code>simplenet</code>	simple example of network
<code>lineardisc</code>	disc in a linear network
<code>methods.linnet</code>	methods for linnet objects

An object of class "lpp" represents a point pattern on a linear network (for example, road accidents on a road network).

<code>lpp</code>	create a point pattern on a linear network
<code>methods.lpp</code>	methods for lpp objects
<code>rpoislpp</code>	simulate Poisson points on linear network
<code>runiflpp</code>	simulate random points on a linear network
<code>chicago</code>	Chicago street crime data

Hyperframes

A hyperframe is like a data frame, except that the entries may be objects of any kind.

<code>hyperframe</code>	create a hyperframe
<code>as.hyperframe</code>	convert data to hyperframe
<code>plot.hyperframe</code>	plot hyperframe
<code>with.hyperframe</code>	evaluate expression using each row of hyperframe
<code>cbind.hyperframe</code>	combine hyperframes by columns
<code>rbind.hyperframe</code>	combine hyperframes by rows
<code>as.data.frame.hyperframe</code>	convert hyperframe to data frame

Layered objects

A layered object represents data that should be plotted in successive layers, for example, a background and a foreground.

<code>layered</code>	create layered object
<code>plot.layered</code>	plot layered object

II. EXPLORATORY DATA ANALYSIS

Inspection of data:

<code>summary(X)</code>	print useful summary of point pattern X
<code>X</code>	print basic description of point pattern X
<code>any(duplicated(X))</code>	check for duplicated points in pattern X
<code>istat(X)</code>	Interactive exploratory analysis

Classical exploratory tools:

<code>clarkevans</code>	Clark and Evans aggregation index
<code>fryplot</code>	Fry plot
<code>miplot</code>	Morishita Index plot

Modern exploratory tools:

<code>nnclean</code>	Byers-Raftery feature detection
<code>sharpen.ppp</code>	Choi-Hall data sharpening
<code>rhohat</code>	Smoothing estimate of covariate effect

Summary statistics for a point pattern:

<code>quadratcount</code>	Quadrat counts
<code>Fest</code>	empty space function F
<code>Gest</code>	nearest neighbour distribution function G
<code>Jest</code>	J -function $J = (1 - G)/(1 - F)$
<code>Kest</code>	Ripley's K -function
<code>Lest</code>	Besag L -function
<code>Tstat</code>	Third order T -function
<code>allstats</code>	all four functions F, G, J, K
<code>pcf</code>	pair correlation function
<code>Kinhom</code>	K for inhomogeneous point patterns
<code>Linhom</code>	L for inhomogeneous point patterns
<code>pcfinhom</code>	pair correlation for inhomogeneous patterns
<code>localL</code>	Getis-Franklin neighbourhood density function
<code>localK</code>	neighbourhood K -function
<code>localpcf</code>	local pair correlation function
<code>localKinhom</code>	local K for inhomogeneous point patterns
<code>localLinhom</code>	local L for inhomogeneous point patterns
<code>localpcfinhom</code>	local pair correlation for inhomogeneous patterns
<code>Kest.fft</code>	fast K -function using FFT for large datasets
<code>Kmeasure</code>	reduced second moment measure
<code>envelope</code>	simulation envelopes for a summary function
<code>varblock</code>	variances and confidence intervals
<code>lohboot</code>	for a summary function
	bootstrap for a summary function

Related facilities:

<code>plot.fv</code>	plot a summary function
<code>eval.fv</code>	evaluate any expression involving summary functions
<code>eval.fasp</code>	evaluate any expression involving an array of functions
<code>with.fv</code>	evaluate an expression for a summary function
<code>smooth.fv</code>	apply smoothing to a summary function
<code>nndist</code>	nearest neighbour distances
<code>nwhich</code>	find nearest neighbours
<code>pairdist</code>	distances between all pairs of points
<code>crossdist</code>	distances between points in two patterns
<code>ncross</code>	nearest neighbours between two point patterns
<code>exactdt</code>	distance from any location to nearest data point
<code>distmap</code>	distance map image
<code>distfun</code>	distance map function
<code>density.ppp</code>	kernel smoothed density
<code>smooth.ppp</code>	spatial interpolation of marks
<code>relrisk</code>	kernel estimate of relative risk
<code>sharpen.ppp</code>	data sharpening
<code>rknn</code>	theoretical distribution of nearest neighbour distance

Summary statistics for a multitype point pattern: A multitype point pattern is represented by an object X of class "ppp" such that `marks(X)` is a factor.

<code>relrisk</code>	kernel estimation of relative risk
<code>scan.test</code>	spatial scan test of elevated risk
<code>Gcross, Gdot, Gmulti</code>	multitype nearest neighbour distributions $G_{ij}, G_{i\bullet}$

<code>Kcross, Kdot, Kmulti</code>	multitype K -functions $K_{ij}, K_{i\bullet}$
<code>Lcross, Ldot</code>	multitype L -functions $L_{ij}, L_{i\bullet}$
<code>Jcross, Jdot, Jmulti</code>	multitype J -functions $J_{ij}, J_{i\bullet}$
<code>pcfcross</code>	multitype pair correlation function g_{ij}
<code>pcfdot</code>	multitype pair correlation function $g_{i\bullet}$
<code>markconnect</code>	marked connection function p_{ij}
<code>alltypes</code>	estimates of the above for all i, j pairs
<code>Iest</code>	multitype I -function
<code>Kcross.inhom, Kdot.inhom</code>	inhomogeneous counterparts of <code>Kcross, Kdot</code>
<code>Lcross.inhom, Ldot.inhom</code>	inhomogeneous counterparts of <code>Lcross, Ldot</code>
<code>pcfcross.inhom, pcfdot.inhom</code>	inhomogeneous counterparts of <code>pcfcross, pcfdot</code>

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

<code>markmean</code>	smoothed local average of marks
<code>markvar</code>	smoothed local variance of marks
<code>markcorr</code>	mark correlation function
<code>markvario</code>	mark variogram
<code>markcorrint</code>	mark correlation integral
<code>Emark</code>	mark independence diagnostic $E(r)$
<code>Vmark</code>	mark independence diagnostic $V(r)$
<code>nnmean</code>	nearest neighbour mean index
<code>nnvario</code>	nearest neighbour mark variance index

For marks of any type, there are the following:

<code>Gmulti</code>	multitype nearest neighbour distribution
<code>Kmulti</code>	multitype K -function
<code>Jmulti</code>	multitype J -function

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

Programming tools:

<code>applynbd</code>	apply function to every neighbourhood in a point pattern
<code>markstat</code>	apply function to the marks of neighbours in a point pattern
<code>marktable</code>	tabulate the marks of neighbours in a point pattern
<code>pppdist</code>	find the optimal match between two point patterns

Summary statistics for a point pattern on a linear network:

These are for point patterns on a linear network (class `lpp`).

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

Related facilities:

<code>pairdist.lpp</code>	shortest path distances
<code>envelope.lpp</code>	simulation envelopes
<code>rpoislpp</code>	simulate Poisson points on linear network
<code>runiflpp</code>	simulate random points on a linear network

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

Summary statistics for a three-dimensional point pattern:

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

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

Related facilities:

<code>envelope.pp3</code>	simulation envelopes
<code>pairdist.pp3</code>	distances between all pairs of points
<code>crossdist.pp3</code>	distances between points in two patterns
<code>nndist.pp3</code>	nearest neighbour distances
<code>nnwhich.pp3</code>	find nearest neighbours

Computations for multi-dimensional point pattern:

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

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

Summary statistics for random sets:

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

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

III. MODEL FITTING (CLUSTER MODELS)

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

<code>kppm</code>	Fit model
<code>plot.kppm</code>	Plot the fitted model
<code>predict.kppm</code>	Compute fitted intensity
<code>update.kppm</code>	Update the model
<code>simulate.kppm</code>	Generate simulated realisations
<code>vcov.kppm</code>	Variance-covariance matrix of coefficients
<code>Kmodel.kppm</code>	K function of fitted model

`pcfmodel.kppm` Pair correlation of fitted model

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

Lower-level fitting functions include:

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

IV. MODEL FITTING (POISSON AND GIBBS MODELS)

Types of models

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

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

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

To fit a Poisson or Gibbs point process model:

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

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

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

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

Manipulating the fitted model:

<code>plot.ppm</code>	Plot the fitted model
<code>predict.ppm</code>	Compute the spatial trend and conditional intensity

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

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

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

To specify a point process model:

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

<code>~1</code>	No trend (stationary)
<code>~x</code>	Loglinear trend $\lambda(x, y) = \exp(\alpha + \beta x)$ where x, y are Cartesian coordinates
<code>~polynom(x,y,3)</code>	Log-cubic polynomial trend
<code>~harmonic(x,y,2)</code>	Log-harmonic polynomial trend

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

<code>Poisson()</code>	the Poisson point process
<code>AreaInter()</code>	Area-interaction process
<code>BadGey()</code>	multiscale Geyer process
<code>DiggleGratton()</code>	Diggle-Gratton potential
<code>DiggleGatesStibbard()</code>	Diggle-Gates-Stibbard potential
<code>Fiksel()</code>	Fiksel pairwise interaction process
<code>Geyer()</code>	Geyer’s saturation process
<code>Hardcore()</code>	Hard core process
<code>LennardJones()</code>	Lennard-Jones potential
<code>MultiHard()</code>	multitype hard core process
<code>MultiStrauss()</code>	multitype Strauss process
<code>MultiStraussHard()</code>	multitype Strauss/hard core process
<code>OrdThresh()</code>	Ord process, threshold potential

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

Finer control over model fitting:

A quadrature scheme is represented by an object of class "quad". To create a quadrature scheme, typically use `quadscheme`.

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

To inspect a quadrature scheme:

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

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

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

To compute weights:

<code>gridweights</code>	quadrature weights by the grid-counting rule
<code>dirichlet.weights</code>	quadrature weights are Dirichlet tile areas

Simulation and goodness-of-fit for fitted models:

<code>rmh.ppm</code>	simulate realisations of a fitted model
<code>simulate.ppm</code>	simulate realisations of a fitted model
<code>envelope</code>	compute simulation envelopes for a fitted model

Point process models on a linear network:

An object of class "lpp" represents a pattern of points on a linear network. Point process models can also be fitted to these objects. Currently only Poisson models can be fitted.

<code>lppm</code>	point process model on linear network
-------------------	---------------------------------------

<code>anova.lppm</code>	analysis of deviance for point process model on linear network
<code>envelope.lppm</code>	simulation envelopes for point process model on linear network
<code>predict.lppm</code>	model prediction on linear network
<code>linim</code>	pixel image on linear network
<code>plot.linim</code>	plot a pixel image on linear network

V. MODEL FITTING (SPATIAL LOGISTIC REGRESSION)

Logistic regression

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

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

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

Fitting a spatial logistic regression

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

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

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

Manipulating a fitted spatial logistic regression

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

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

VI. SIMULATION

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

Random point patterns:

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

Resampling a point pattern:

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

See also `varblock` for estimating the variance of a summary statistic by block resampling, and `loohboot` for another bootstrap technique.

Fitted point process models:

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

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

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

Other random patterns:

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

Simulation-based inference

<code>envelope</code>	critical envelope for Monte Carlo test of goodness-of-fit
<code>qqplot.ppm</code>	diagnostic plot for interpoint interaction
<code>scan.test</code>	spatial scan statistic/test

VII. TESTS AND DIAGNOSTICS

Classical hypothesis tests:

<code>quadrat.test</code>	χ^2 goodness-of-fit test on quadrat counts
<code>clarkevans.test</code>	Clark and Evans test
<code>kstest</code>	Kolmogorov-Smirnov goodness-of-fit test
<code>bermantest</code>	Berman's goodness-of-fit tests
<code>envelope</code>	critical envelope for Monte Carlo test of goodness-of-fit
<code>clf.test</code>	Cressie (1991)/Loosmore and Ford (2006) test
<code>mad.test</code>	Mean Absolute Deviation test
<code>scan.test</code>	spatial scan statistic/test
<code>anova.ppm</code>	Analysis of Deviance for point process models

Sensitivity diagnostics:

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

<code>leverage.ppm</code>	Leverage for point process model
<code>influence.ppm</code>	Influence for point process model
<code>dfbetas.ppm</code>	Parameter influence

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 Moller (2011).

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

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

Resampling and randomisation procedures

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

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

<code>rthin</code>	random thinning
--------------------	-----------------

VIII. DOCUMENTATION

The online manual entries are quite detailed and should be consulted first for information about a particular function.

The paper by Baddeley and Turner (2005a) is a brief overview of the package. Baddeley and Turner (2005b) is a more detailed explanation of how to fit point process models to data. Baddeley (2010) is a complete set of notes from a 2-day workshop on the use of `spatstat`.

Type `citation("spatstat")` to get these references.

Licence

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

Acknowledgements

Kasper Klitgaard Berthelsen, Abdollah Jalilian, Marie-Colette van Lieshout, Ege Rubak, Dominic Schuhmacher and Rasmus Waagepetersen made substantial contributions of code. Additional contributions by Ang Qi Wei, Sandro Azaele, Colin Beale, Ricardo Bernhardt, Brad Biggerstaff, Roger Bivand, Florent Bonneau, Julian Burgos, Simon Byers, Ya-Mei Chang, Jianbao Chen, Igor Chernayavsky, Y.C. Chin, Bjarke Christensen, Jean-Francois Coeurjolly, Marcelino de la Cruz, Peter Dalgaard, Peter Diggle, Ian Dryden, Stephen Eglen, Neba Funwi-Gabga, Agnes Gault, Marc Genton, Pavel Grabarnik, C. Graf, Janet Franklin, Ute Hahn, Andrew Hardegen, Mandy Hering, Martin Bogsted Hansen, Martin Hazelton, Juha Heikkilä, Kurt Hornik, Ross Ihaka, Aruna Jammalamadaka, Robert John-Chandran, Devin Johnson, Mike Kuhn, Jeff Laake, Tom Lawrence, Robert Lamb, Jonathan Lee, George Leser, Ben Madin, Robert Mark, Jorge Mateu Mahiques, Monia Mahling, Peter McCullagh, Ulf Mehlig, Sebastian Wastl Meyer, Mi Xiangcheng, Jesper Moller, Linda Stougaard Nielsen, Felipe Nunes, Jens Oehlschlaegel, Thierry Onkelinx, Evgeni Parilov, Jeff Picka, Sergiy Protsiv, Adrian Raftery, Matt Reiter, Tom Richardson, Brian Ripley, Barry Rowlingson, John Rudge, Farzaneh Safavimanesh, Aila Sarkka, Katja Schladitz, Bryan Scott, Vadim Shcherbakov, Shen Guochun, Ida-Maria Sintorn, Yong Song, Malte Spiess, Mark Stevenson, Kasper Stucki, Michael Sumner, P. Surovy, Ben Taylor, Berwin Turlach, Andrew van Burgel, Tobias Verbeke, Alexendre Villers, Hao Wang, H. Wendrock, Jan Wild, Selene Wong and Mike Zamboni.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

References

- Baddeley, A. (2010) *Analysing spatial point patterns in R*. Workshop notes. Version 4.1. CSIRO online technical publication. URL: www.csiro.au/resources/pf16h.html
- Baddeley, A. and Turner, R. (2005a) Spatstat: an R package for analyzing spatial point patterns. *Journal of Statistical Software* **12**:6, 1–42. URL: www.jstatsoft.org, ISSN: 1548-7660.
- Baddeley, A. and Turner, R. (2005b) Modelling spatial point patterns in R. In: A. Baddeley, P. 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., Turner, R., Moller, 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 Moller, J. (2011) Score, pseudo-score and residual diagnostics for spatial point process models. *Statistical Science* **26**, 613–646.
- Diggle, P.J. (2003) *Statistical analysis of spatial point patterns*, Second edition. Arnold.
- Gelfand, A.E., Diggle, P.J., Fuentes, M. and Guttorp, P., editors (2010) *Handbook of Spatial Statistics*. CRC Press.
- Huang, F. and Ogata, Y. (1999) Improvements of the maximum pseudo-likelihood estimators in various spatial statistical models. *Journal of Computational and Graphical Statistics* **8**, 510–530.
- Waagepetersen, R. An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63** (2007) 252–258.

adaptive.density*Intensity Estimate of Point Pattern Using Tessellation***Description**

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

Usage

```
adaptive.density(X, f = 0.1, ..., nrep = 1)
```

Arguments

X	Point pattern dataset (object of class "ppp").
f	Fraction (between 0 and 1) of the data points that will be removed from the data and used to determine a tessellation for the intensity estimate.
...	Arguments passed to as.im determining the pixel resolution of the result.
nrep	Number of independent repetitions of the randomised procedure.

Details

This function is an alternative to [density.ppp](#). It computes an estimate of the intensity function of a point pattern dataset.

The dataset X is randomly split into two patterns A and B containing a fraction f and 1-f, respectively, of the original data. The subpattern A is used to construct a Dirichlet tessellation (see [dirichlet](#)). The subpattern B is retained for counting. For each tile of the Dirichlet tessellation, we count the number of points of B falling in the tile, and divide by the area of the same tile, to obtain an estimate of the intensity of the pattern B in the tile. This estimate is divided by 1-f to obtain an estimate of the intensity of X in the tile. The result is a pixel image of intensity estimates which are constant on each tile of the tessellation.

If nrep is greater than 1, this randomised procedure is repeated nrep times, and the results are averaged.

This technique has been used by Ogata et al. (2003), Ogata (2004) and Baddeley (2007).

Value

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

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Baddeley, A. (2007) Validation of statistical models for spatial point patterns. In J.G. Babu and E.D. Feigelson (eds.) *SCMA IV: Statistical Challenges in Modern Astronomy IV*, volume 317 of Astronomical Society of the Pacific Conference Series, San Francisco, California USA, 2007. Pages 22–38.
- Ogata, Y. (2004) Space-time model for regional seismicity and detection of crustal stress changes. *Journal of Geophysical Research*, **109**, 2004.
- Ogata, Y., Katsura, K. and Tanemura, M. (2003). Modelling heterogeneous space-time occurrences of earthquake and its residual analysis. *Applied Statistics* **52** 499–509.

See Also

[density.ppp](#), [dirichlet](#), [im.object](#).

Examples

```
## Not run:
data(nztrees)
plot(adaptive.density(nztrees))

## End(Not run)
```

affine

Apply Affine Transformation

Description

Applies any affine transformation of the plane (linear transformation plus vector shift) to a plane geometrical object, such as a point pattern or a window.

Usage

`affine(X, ...)`

Arguments

- | | |
|-----|---|
| X | Any suitable dataset representing a two-dimensional object, such as a point pattern (object of class "ppp"), a line segment pattern (object of class "psp"), a window (object of class "owin") or a pixel image (object of class "im"). |
| ... | Arguments determining the affine transformation. |

Details

This is generic. Methods are provided for point patterns ([affine.ppp](#)) and windows ([affine.owin](#)).

Value

Another object of the same type, representing the result of applying the affine transformation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[affine.ppp](#), [affine.psp](#), [affine.owin](#), [affine.im](#), [flipxy](#), [reflect](#), [rotate](#), [shift](#)

[affine.im](#)

Apply Affine Transformation To Pixel Image

Description

Applies any affine transformation of the plane (linear transformation plus vector shift) to a pixel image.

Usage

```
## S3 method for class 'im'
affine(X, mat=diag(c(1,1)), vec=c(0,0), ...)
```

Arguments

- | | |
|-----|---|
| X | Pixel image (object of class "im"). |
| mat | Matrix representing a linear transformation. |
| vec | Vector of length 2 representing a translation. |
| ... | Optional arguments passed to as.mask controlling the pixel resolution of the transformed image. |

Details

The image is subjected first to the linear transformation represented by `mat` (multiplying on the left by `mat`), and then the result is translated by the vector `vec`.

The argument `mat` must be a nonsingular 2×2 matrix.

This is a method for the generic function [affine](#).

Value

Another pixel image (of class "im") representing the result of applying the affine transformation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[affine](#), [affine.ppp](#), [affine.psp](#), [affine.owin](#), [rotate](#), [shift](#)

Examples

```
X <- setcov(owin())
stretch <- diag(c(2,3))
Y <- affine(X, mat=stretch)
shear <- matrix(c(1,0,0.6,1), ncol=2, nrow=2)
Z <- affine(X, mat=shear)
```

affine.owin

Apply Affine Transformation To Window

Description

Applies any affine transformation of the plane (linear transformation plus vector shift) to a window.

Usage

```
## S3 method for class 'owin'
affine(X, mat=diag(c(1,1)), vec=c(0,0), ..., rescue=TRUE)
```

Arguments

<code>X</code>	Window (object of class "owin").
<code>mat</code>	Matrix representing a linear transformation.
<code>vec</code>	Vector of length 2 representing a translation.
<code>rescue</code>	Logical. If TRUE, the transformed window will be processed by rescue.rectangle .
<code>...</code>	Optional arguments passed to as.mask controlling the pixel resolution of the transformed window, if <code>X</code> is a binary pixel mask.

Details

The window is subjected first to the linear transformation represented by `mat` (multiplying on the left by `mat`), and then the result is translated by the vector `vec`.

The argument `mat` must be a nonsingular 2×2 matrix.

This is a method for the generic function [affine](#).

Value

Another window (of class "owin") representing the result of applying the affine transformation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[affine](#), [affine.ppp](#), [affine.psp](#), [affine.im](#), [rotate](#), [shift](#)

Examples

```
# shear transformation
shear <- matrix(c(1,0,0.6,1),ncol=2)
X <- affine(owin(), shear)
## Not run:
plot(X)

## End(Not run)
data(letterR)
affine(letterR, shear, c(0, 0.5))
affine(as.mask(letterR), shear, c(0, 0.5))
```

affine.ddd

Apply Affine Transformation To Point Pattern

Description

Applies any affine transformation of the plane (linear transformation plus vector shift) to a point pattern.

Usage

```
## S3 method for class 'ddd'
affine(X, mat=diag(c(1,1)), vec=c(0,0), ...)
```

Arguments

- X Point pattern (object of class "ddd").
- mat Matrix representing a linear transformation.
- vec Vector of length 2 representing a translation.
- ... Arguments passed to `affine.owin` affecting the handling of the observation window, if it is a binary pixel mask.

Details

The point pattern, and its window, are subjected first to the linear transformation represented by `mat` (multiplying on the left by `mat`), and are then translated by the vector `vec`.

The argument `mat` must be a nonsingular 2×2 matrix.

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

Value

Another point pattern (of class "ddd") representing the result of applying the affine transformation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[affine](#), [affine.owin](#), [affine.psp](#), [affine.im](#), [flipxy](#), [rotate](#), [shift](#)

Examples

```
data(cells)
# shear transformation
X <- affine(cells, matrix(c(1,0,0.6,1),ncol=2))
## Not run:
plot(X)
# rescale y coordinates by factor 1.3
plot(affine(cells, diag(c(1,1.3))))
```

End(Not run)

affine.psp

*Apply Affine Transformation To Line Segment Pattern***Description**

Applies any affine transformation of the plane (linear transformation plus vector shift) to a line segment pattern.

Usage

```
## S3 method for class 'psp'
affine(X, mat=diag(c(1,1)), vec=c(0,0), ...)
```

Arguments

X	Line Segment pattern (object of class "psp").
mat	Matrix representing a linear transformation.
vec	Vector of length 2 representing a translation.
...	Arguments passed to affine.owin affecting the handling of the observation window, if it is a binary pixel mask.

Details

The line segment pattern, and its window, are subjected first to the linear transformation represented by `mat` (multiplying on the left by `mat`), and are then translated by the vector `vec`.

The argument `mat` must be a nonsingular 2×2 matrix.

This is a method for the generic function [affine](#).

Value

Another line segment pattern (of class "psp") representing the result of applying the affine transformation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[affine](#), [affine.owin](#), [affine.ppp](#), [affine.im](#), [flipxy](#), [rotate](#), [shift](#)

Examples

```
oldpar <- par(mfrow=c(2,1))
X <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
plot(X, main="original")
# shear transformation
Y <- affine(X, matrix(c(1,0,0.6,1),ncol=2))
plot(Y, main="transformed")
par(oldpar)
#
# rescale y coordinates by factor 0.2
affine(X, diag(c(1,0.2)))
```

allstats

Calculate four standard summary functions of a point pattern.

Description

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

Usage

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

Arguments

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

Details

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

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

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

Value

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

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

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

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

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`plot.listof`, `plot.fv`, `fv.object`, `Fest`, `Gest`, `Jest`, `Kest`

Examples

```
data(swedishpines)
a <- allstats(swedishpines,dataname="Swedish Pines")
## Not run:
plot(a)
plot(a, subset=list("r<=15", "r<=15", "r<=15", "r<=50"))

## End(Not run)
```

alltypes

Calculate Summary Statistic for All Types in a Multitype Point Pattern

Description

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

Usage

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

Arguments

- | | |
|------------------|---|
| <code>X</code> | The observed point pattern, for which summary function estimates are required. An object of class "ppp". |
| <code>fun</code> | The summary function. Either an R function, or a character string indicating the summary function required. Options for strings are "F", "G", "J", "K", "L", "pcf", "Gcross", "Jcross", "Kcross", "Lcross", "Gdot", "Jdot", "Kdot", "Ldot". |
| <code>...</code> | Arguments passed to the summary function (and to the function <code>envelope</code> if appropriate) |

dataname	Character string giving an optional (alternative) name to the point pattern, different from what is given in the call. This name, if supplied, may be used by plot.fasp() in forming the title of the plot. If not supplied it defaults to the parsing of the argument supplied as X in the call.
verb	Logical value. If verb is true then terse “progress reports” (just the values of the mark indices) are printed out when the calculations for that combination of marks are completed.
envelope	Logical value. If envelope is true, then simulation envelopes of the summary function will also be computed. See Details.

Details

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

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

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

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

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

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

If fun is "Gdot", "Jdot", "Kdot" or "Ldot", the routine calculates, for each type i , an estimate of the “i-to-any” dot-type function $G_{i\bullet}(r)$, $J_{i\bullet}(r)$ or $K_{i\bullet}(r)$ or $L_{i\bullet}(r)$ respectively describing the dependence between X_i and X. See [Gdot](#), [Jdot](#), [Kdot](#) or [Ldot](#) respectively for explanation of these functions. The estimate is computed by applying the relevant function ([Gdot](#) etc) to X using each possible value of the argument i, together with the optional arguments

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

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

Value

A function array (an object of class "fasp", see [fasp.object](#)). This can be plotted using [plot.fasp](#). If the pattern is not marked, the resulting “array” has dimensions 1×1 . Otherwise the following is true:

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

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

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

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

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

Note

Sizeable amounts of memory may be needed during the calculation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

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

Examples

```
# bramblecanes (3 marks).
data(bramblecanes)

bF <- alltypes(bramblecanes, "F", verb=TRUE)
plot(bF)
plot(alltypes(bramblecanes, "G"))
plot(alltypes(bramblecanes, "Gdot"))

# Swedishpines (unmarked).
data(swedishpines)

plot(alltypes(swedishpines, "K"))

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

# A setting where you might REALLY want to use dataname:
## Not run:
xxx <- alltypes(ppp(Melvin$x, Melvin$y,
                      window=as.owin(c(5,20,15,50)), marks=clyde),
```

```

fun="F",verb=TRUE,dataname="Melvin")

## End(Not run)

# envelopes
bKE <- alltypes(bramblecanes,"K",envelope=TRUE,nsim=19)
## Not run:
bFE <- alltypes(bramblecanes,"F",envelope=TRUE,nsim=19,global=TRUE)

## End(Not run)

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

```

amacrine

Hughes' Amacrine Cell Data

Description

Austin Hughes' data: a point pattern of displaced amacrine cells in the retina of a rabbit. A marked point pattern.

Usage

```
data(amacrine)
```

Format

An object of class "ppp" representing the point pattern of cell locations. Entries include

x	Cartesian x -coordinate of cell
y	Cartesian y -coordinate of cell
marks	factor with levels off and on indicating "off" and "on" cells

See [ppp.object](#) for details of the format.

Notes

Austin Hughes' data: a point pattern of displaced amacrine cells in the retina of a rabbit. 152 "on" cells and 142 "off" cells in a rectangular sampling frame.

The true dimensions of the rectangle are 1060 by 662 microns. The coordinates here are scaled to a rectangle of height 1 and width $1060/662 = 1.601$ so the unit of measurement is approximately 662 microns.

The data were analysed by Diggle (1986).

Source

Peter Diggle, personal communication

References

Diggle, P. J. (1986). Displaced amacrine cells in the retina of a rabbit: analysis of a bivariate spatial point pattern. *J. Neurosci. Meth.* **18**, 115–125.

anemones

Beadlet Anemones Data

Description

These data give the spatial locations and diameters of sea anemones (beadlet anemone *Actinia equina*) in a sample plot on the north face of a boulder, well above low tide level, at Quiberon (Bretagne, France) in May 1976.

The data were originally described and discussed by Kooijman (1979a). Kooijman (1979b) shows a hand-drawn plot of the original data. The data are discussed by Upton and Fingleton (1985) as Example 1.8 on pages 64–67.

The anemones dataset is taken directly from Table 1.11 of Upton and Fingleton (1985). The coordinates and diameters are integer multiples of an idiosyncratic unit of length. The boundary is a rectangle 280 by 180 units.

Usage

```
data(anemones)
```

Format

anemones is an object of class "ppp" representing the point pattern of anemone locations. It is a marked point pattern with numeric marks representing anemone diameter. See [ppp.object](#) for details of the format.

Units

There is some confusion about the correct physical scale for these data. According to Upton and Fingleton (1985), one unit in the dataset is approximately 0.475 cm. According to Kooijman (1979a, 1979b) and also quoted by Upton and Fingleton (1985), the physical size of the sample plot was 14.5 by 9.75 cm. However if the data are plotted at this scale, they are too small for a rectangle of this size, and the appearance of the plot does not match the original hand-drawn plot in Kooijman (1979b). To avoid confusion, we have not assigned a unit scale to this dataset.

Source

Table 1.11 on pages 62–63 of Upton and Fingleton (1985), who acknowledge Kooijman (1979a) as the source.

References

- Kooijman, S.A.L.M. (1979a) The description of point patterns. In *Spatial and temporal analysis in ecology* (ed. R.M. Cormack and J.K. Ord), International Cooperative Publishing House, Fairland, Maryland, USA. Pages 305–332.
- Kooijman, S.A.L.M. (1979b) Inference about dispersal patterns. *Acta Biotheoretica* **28**, 149–189.
- Upton, G.J.G. and Fingleton, B. (1985) *Spatial data analysis by example*. Volume 1: Point pattern and quantitative data. John Wiley and Sons, Chichester.

Examples

```
data(anemones)
# plot diameters on same scale as x, y
plot(anemones, markscale=0.5)
```

angles.psp

Orientation Angles of Line Segments

Description

Computes the orientation angle of each line segment in a line segment pattern.

Usage

```
angles.psp(x, directed=FALSE)
```

Arguments

- | | |
|----------|---|
| x | A line segment pattern (object of class "psp"). |
| directed | Logical flag. See details. |

Details

For each line segment, the angle of inclination to the x -axis (in radians) is computed, and the angles are returned as a numeric vector.

If `directed=TRUE`, the directed angle of orientation is computed. The angle respects the sense of direction from (x_0, y_0) to (x_1, y_1) . The values returned are angles in the full range from $-\pi$ to π . The angle is computed as $\text{atan2}(y_1-y_0, x_1-x_0)$. See `atan2`.

If `directed=FALSE`, the undirected angle of orientation is computed. Angles differing by π are regarded as equivalent. The values returned are angles in the range from 0 to π . These angles are computed by first computing the directed angle, then adding π to any negative angles.

Value

Numeric vector.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[summary.psp](#), [midpoints.psp](#), [lengths.psp](#)

Examples

```
a <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
b <- angles.psp(a)
```

anova.lppmANOVA for Fitted Point Process Models on Linear Network

Description

Performs analysis of deviance for two or more fitted point process models on a linear network.

Usage

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

Arguments

object	A fitted point process model on a linear network (object of class "lppm").
...	One or more fitted point process models on the same linear network.
test	Character string, partially matching one of "Chisq", "F" or "Cp".
override	Logical flag indicating whether to proceed even when there is no statistical theory to support the calculation.

Details

This is a method for [anova](#) for fitted point process models on a linear network (objects of class "lppm", usually generated by the model-fitting function [lppm](#)).

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, then there is no statistical theory available to support a similar analysis. The function issues a warning, and (by default) returns a `NULL` value.

However if `override=TRUE`, then a kind of analysis of deviance table will be printed. The 'deviance' differences in this table are equal to 2 times the differences in the maximised values of the log pseudolikelihood (see [ppm](#)). At the time of writing, there is no statistical theory to support inferential interpretation of log pseudolikelihood ratios. The `override` option is provided for research purposes only!

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 generally occurs when the point process models are fitted on different linear networks.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

References

- Ang, Q.W. (2010) *Statistical methodology for events on a network*. Master’s thesis, School of Mathematics and Statistics, University of Western Australia.
- Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. To appear in *Scandinavian Journal of Statistics*.
- McSwiggan, G., Nair, M.G. and Baddeley, A. (2012) Fitting Poisson point process models to events on a linear network. Manuscript in preparation.

See Also

[lppm](#)

Examples

```
example(lpp)
mod0 <- lppm(X, ~1)
modx <- lppm(X, ~x)
anova(mod0, modx, test="Chi")
```

anova.ppm

ANOVA for Fitted Point Process Models

Description

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

Usage

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

Arguments

- | | |
|----------|---|
| object | A fitted point process model (object of class “ <code>ppm</code> ”). |
| ... | One or more fitted point process models. |
| test | Character string, partially matching one of “ <code>Chisq</code> ”, “ <code>F</code> ” or “ <code>Cp</code> ”. |
| override | Logical flag indicating whether to proceed even when there is no statistical theory to support the calculation. |

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 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, then there is no statistical theory available to support a similar analysis. The function issues a warning, and (by default) returns a NULL value.

However if `override=TRUE`, then a kind of analysis of deviance table will be printed. The 'deviance' differences in this table are equal to 2 times the differences in the maximised values of the log pseudolikelihood (see [ppm](#)). At the time of writing, there is no statistical theory to support inferential interpretation of log pseudolikelihood ratios. The `override` option is provided for research purposes only!

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.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppm](#)

Examples

```
data(swedishpines)
mod0 <- ppm(swedishpines, ~1, Poisson())
modx <- ppm(swedishpines, ~x, Poisson())
anova(mod0, modx, 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 <i>p</i> -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@maths.uwa.edu.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also[slrm](#)**Examples**

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

ants

*Harkness-Isham ants' nests data***Description**

These data give the spatial locations of nests of two species of ants, *Messor wasmanni* and *Cataglyphis bicolor*, recorded by Professor R.D.\ Harkness at a site in northern Greece, and described in Harkness \& Isham (1983). The full dataset (supplied here) has an irregular polygonal boundary, while most analyses have been confined to two rectangular subsets of the pattern (also supplied here).

The harvester ant *M. wasmanni* collects seeds for food and builds a nest composed mainly of seed husks. *C. bicolor* is a heat-tolerant desert foraging ant which eats dead insects and other arthropods. Interest focuses on whether there is evidence in the data for intra-species competition between *Messor* nests (i.e. competition for resources) and for preferential placement of *Cataglyphis* nests in the vicinity of *Messor* nests.

The full dataset is displayed in Figure 1 of Harkness \& Isham (1983). See **Usage** below to produce a comparable plot. It comprises 97 nests (68 *Messor* and 29 *Cataglyphis*) inside an irregular convex polygonal boundary, together with annotations showing a foot track through the region, the boundary between field and scrub areas inside the region, and indicating the two rectangular subregions A and B used in their analysis.

Rectangular subsets of the data were analysed by Harkness \& Isham (1983), Isham (1984), Takacs \& Fiksel (1986), S\"{a}rkk\"{a} (1993, section 5.3), H\"{o}gmander and S\"{a}rkk\"{a} (1999) and Baddeley \& Turner (2000). The full dataset (inside its irregular boundary) was first analysed by Baddeley \& Turner (2005b).

The dataset `ants` is the full point pattern enclosed by the irregular polygonal boundary. The *x* and *y* coordinates are eastings (E-W) and northings (N-S) scaled so that 1 unit equals 0.5 feet. This is a multitype point pattern object, each point carrying a mark indicating the ant species (with levels *Cataglyphis* and *Messor*).

The dataset `ants.extra` is a list of auxiliary information:

A and B The subsets of the pattern within the rectangles A and B demarcated in Figure 1 of Harkness \& Isham (1983). These are multitype point pattern objects.

trackNE and trackSW coordinates of two straight lines bounding the foot track.

fieldscrub The endpoints of a straight line separating the regions of ‘field’ and ‘scrub’: scrub to the North and field to the South.

side A function(*x*,*y*) that determines whether the location (*x*,*y*) is in the scrub or the field. The function can be applied to numeric vectors *x* and *y*, and returns a factor with levels “scrub” and “field”. This function is useful as a spatial covariate.

plot A function which produces a plot of the full dataset.

Usage

```
data(ants)
```

Format

`ants` is an object of class "ppp" representing the full point pattern of ants' nests. See [ppp.object](#) for details of the format. The coordinates are scaled so that 1 unit equals 0.5 feet. The points are marked by species (with levels *Cataglyphis* and *Messor*).

`ants.extra` is a list with entries

A point pattern of class "ppp"

B point pattern of class "ppp"

trackNE data in format `list(x=numeric(2),y=numeric(2))` giving the two endpoints of line markings

trackSW data in format `list(x=numeric(2),y=numeric(2))` giving the two endpoints of line markings

fieldscrub data in format `list(x=numeric(2),y=numeric(2))` giving the two endpoints of line markings

side Function with arguments `x,y`

plot Function

Source

Harkness and Isham (1983). Nest coordinates kindly provided by Prof Valerie Isham. Polygon coordinates digitised by Adrian Baddeley from a reprint of Harkness & Isham (1983).

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. (2005a) Spatstat: an R package for analyzing spatial point patterns. *Journal of Statistical Software* **12**:6, 1–42. URL: www.jstatsoft.org, ISSN: 1548-7660.
- Baddeley, A. and Turner, R. (2005b) Modelling spatial point patterns in R. In: A. Baddeley, P. 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.
- Harkness, R.D. and Isham, V. (1983) A bivariate spatial point pattern of ants' nests. *Applied Statistics* **32**, 293–303.
- Hagman, H. and Arkkila, A. (1999) Multitype spatial point patterns with hierarchical interactions. *Biometrics* **55**, 1051–1058.
- Isham, V.S. (1984) Multitype Markov point processes: some approximations. *Proceedings of the Royal Society of London, Series A*, **391**, 39–53.
- Takacs, R. and Fiksel, T. (1986) Interaction pair-potentials for a system of ants' nests. *Biometrical Journal* **28**, 1007–1013.
- Arkkila, A. (1993) *Pseudo-likelihood approach for pair potential estimation of Gibbs processes*. Number 22 in Jyväskylä Studies in Computer Science, Economics and Statistics. University of Jyväskylä, Finland.

Examples

```
# Equivalent to Figure 1 of Harkness and Isham (1983)

data(ants)
ants.extra$plot()

# Data in subrectangle A, rotated
# Approximate data used by Sarkka (1993)

angle <- atan(diff(ants.extra$fieldscrub$y)/diff(ants.extra$fieldscrub$x))
plot(rotate(ants.extra$A, -angle))

# Approximate window used by Takacs and Fiksel (1986)

tfwindow <- bounding.box(ants>window)
antsTF <- ppp(ants$x, ants$y, window=tfwindow)
plot(antsTF)
```

append.psp

Combine Two Line Segment Patterns

Description

Combine two line segment patterns into a single pattern.

Usage

```
append.psp(A, B)
```

Arguments

A,B	Line segment patterns (objects of class "psp").
-----	---

Details

This function is used to superimpose two line segment patterns A and B.

The two patterns must have **identical** windows. If one pattern has marks, then the other must also have marks of the same type. If the marks are data frames then the number of columns of these data frames, and the names of the columns must be identical.

(To combine two point patterns, see `superimpose`).

Value

Another line segment pattern (object of class "psp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[psp](#), [as.psp](#), [superimpose](#),

Examples

```
X <- psp(runif(20), runif(20), runif(20), runif(20), window=owin())
Y <- psp(runif(5), runif(5), runif(5), runif(5), window=owin())
append.psp(X,Y)
```

applynbd

Apply Function to Every Neighbourhood in a Point Pattern

Description

Visit each point in a point pattern, find the neighbouring points, and apply a given function to them.

Usage

```
applynbd(X, FUN, N, R, criterion, exclude=FALSE, ...)
```

Arguments

X	Point pattern. An object of class "ppp", or data which can be converted into this format by as.ppp .
FUN	Function to be applied to each neighbourhood. The arguments of FUN are described under Details .
N	Integer. If this argument is present, the neighbourhood of a point of X is defined to consist of the N points of X which are closest to it. This argument is incompatible with R and criterion.
R	Nonnegative numeric value. If this argument is present, the neighbourhood of a point of X is defined to consist of all points of X which lie within a distance R of it. This argument is incompatible with N and criterion.
criterion	Function. If this argument is present, the neighbourhood of a point of X is determined by evaluating this function. See under Details . This argument is incompatible with N and R.
exclude	Logical. If TRUE then the point currently being visited is excluded from its own neighbourhood.
...	extra arguments passed to the function FUN. They must be given in the form name=value.

Details

This is an analogue of [apply](#) for point patterns. It visits each point in the point pattern X, determines which points of X are “neighbours” of the current point, applies the function FUN to this neighbourhood, and collects the values returned by FUN.

The definition of “neighbours” depends on the arguments N, R and criterion, exactly one of which must be given. Also the argument exclude determines whether the current point is excluded from its own neighbourhood.

- If N is given, then the neighbours of the current point are the N points of X which are closest to the current point (including the current point itself unless $\text{exclude}=\text{TRUE}$).
- If R is given, then the neighbourhood of the current point consists of all points of X which lie closer than a distance R from the current point.
- If criterion is given, then it must be a function with two arguments dist and drank which will be vectors of equal length. The interpretation is that $\text{dist}[i]$ will be the distance of a point from the current point, and $\text{drank}[i]$ will be the rank of that distance (the three points closest to the current point will have rank 1, 2 and 3). This function must return a logical vector of the same length as dist and drank whose i -th entry is TRUE if the corresponding point should be included in the neighbourhood. See the examples below.

When `applynbd` is executed, each point of X is visited, and the following happens for each point:

- the neighbourhood of the current point is determined according to the chosen rule, and stored as a point pattern Y ;
- the function FUN is called as:
 $\text{FUN}(Y=Y, \text{current}=\text{current}, \text{dists}=\text{dists}, \text{dranks}=\text{dranks}, \dots)$
 where current is the location of the current point (in a format explained below), dists is a vector of distances from the current point to each of the points in Y , dranks is a vector of the ranks of these distances with respect to the full point pattern X , and \dots are the arguments passed from the call to `applynbd`;
- The result of the call to FUN is stored.

The results of each call to FUN are collected and returned according to the usual rules for [apply](#) and its relatives. See the **Value** section of this help file.

The format of the argument current is as follows. If X is an unmarked point pattern, then current is a vector of length 2 containing the coordinates of the current point. If X is marked, then current is a point pattern containing exactly one point, so that $\text{current}\$x$ is its x -coordinate and $\text{current}\$marks$ is its mark value. In either case, the coordinates of the current point can be referred to as $\text{current}\$x$ and $\text{current}\$y$.

Note that FUN will be called exactly as described above, with each argument named explicitly. Care is required when writing the function FUN to ensure that the arguments will match up. See the Examples.

See [markstat](#) for a common use of this function.

To simply tabulate the marks in every R -neighbourhood, use [marktable](#).

Value

Similar to the result of [apply](#). If each call to FUN returns a single numeric value, the result is a vector of dimension $X\$n$, the number of points in X . If each call to FUN returns a vector of the same length m , then the result is a matrix of dimensions $c(m, n)$; note the transposition of the indices, as usual for the family of `apply` functions. If the calls to FUN return vectors of different lengths, the result is a list of length $X\$n$.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppp.object](#), [apply](#), [markstat](#), [marktable](#)

Examples

```

data(redwood)

# count the number of points within radius 0.2 of each point of X
nneighbours <- applynbd(redwood, R=0.2, function(Y, ...){Y$n-1})
# equivalent to:
nneighbours <- applynbd(redwood, R=0.2, function(Y, ...){Y$n}, exclude=TRUE)

# compute the distance to the second nearest neighbour of each point
secondnndist <- applynbd(redwood, N = 2,
                           function(dists, ...){max(dists)},
                           exclude=TRUE)

# marked point pattern
data(longleaf)

# compute the median of the marks of all neighbours of a point
# (see also 'markstat')
dbh.med <- applynbd(longleaf, R=90, exclude=TRUE,
                      function(Y, ...){median(Y$marks)})

# ANIMATION explaining the definition of the K function
# (arguments 'fullpicture' and 'rad' are passed to FUN)

## Not run:
showoffK <- function(Y, current, dists, dranks, fullpicture, rad) {
  plot(fullpicture, main="")
  points(Y, cex=2)
  u <- current
  points(u$x, u$y, pch="+", cex=3)
  theta <- seq(0, 2*pi, length=100)
  polygon(u$x + rad * cos(theta), u$y + rad * sin(theta))
  text(u$x + rad/3, u$y + rad/2, Y$n, cex=3)
  Sys.sleep(if(runif(1) < 0.1) 1.5 else 0.3)
  return(Y$n - 1)
}
applynbd(redwood, R=0.2, showoffK, fullpicture=redwood, rad=0.2, exclude=TRUE)

# animation explaining the definition of the G function

showoffG <- function(Y, current, dists, dranks, fullpicture) {
  plot(fullpicture, main="")
  points(Y, cex=2)
  u <- current
  points(u[1], u[2], pch="+", cex=3)
  v <- c(Y$x[1], Y$y[1])
  segments(u[1], u[2], v[1], v[2], lwd=2)
  w <- (u + v)/2
  nnd <- dists[1]
  text(w[1], w[2], round(nnd, 3), cex=2)
  Sys.sleep(if(runif(1) < 0.1) 1.5 else 0.3)
  return(nnd)
}

data(cells)

```

```
applynbd(cells, N=1, showoffG, exclude=TRUE, fullpicture=cells)
## End(Not run)
```

area.owin

Area of a Window

Description

Computes the area of a window

Usage

```
area.owin(w)
## S3 method for class 'owin'
volume(x)
```

Arguments

- w A window, whose area will be computed. This should be an object of class `owin`, or can be given in any format acceptable to `as.owin()`.
- x Object of class `owin`

Details

If the window `w` is of type "rectangle" or "polygonal", the area of this rectangular window is computed by analytic geometry. If `w` is of type "mask" the area of the discrete raster approximation of the window is computed by summing the binary image values and adjusting for pixel size.

The function `volume.owin` is identical to `area.owin` except for the argument name. It is a method for the generic function `volume`.

Value

A numerical value giving the area of the window.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`perimeter`, `diameter.owin`, `owin.object`, `as.owin`

Examples

```
w <- unit.square()
area.owin(w)
# returns 1.00000

k <- 6
theta <- 2 * pi * (0:(k-1))/k
co <- cos(theta)
si <- sin(theta)
mas <- owin(c(-1,1), c(-1,1), poly=list(x=co, y=si))
area.owin(mas)
# returns approx area of k-gon

mas <- as.mask(square(2), eps=0.01)
X <- raster.x(mas)
Y <- raster.y(mas)
mas$m <- ((X - 1)^2 + (Y - 1)^2 <= 1)
area.owin(mas)
# returns 3.14 approx
```

areaGain

Difference of Disc Areas

Description

Computes the area of that part of a disc that is not covered by other discs.

Usage

```
areaGain(u, X, r, ..., W=as.owin(X), exact=FALSE,
        ngrid=spatstat.options("ngrid.disc"))
```

Arguments

- u** Coordinates of the centre of the disc of interest. A vector of length 2. Alternatively, a point pattern (object of class "ppp").
- X** Locations of the centres of other discs. A point pattern (object of class "ppp").
- r** Disc radius, or vector of disc radii.
- ...** Ignored.
- W** Window (object of class "owin") in which the area should be computed.
- exact** Choice of algorithm. If **exact=TRUE**, areas are computed exactly using analytic geometry. If **exact=FALSE** then a faster algorithm is used to compute a discrete approximation to the areas.
- ngrid** Integer. Number of points in the square grid used to compute the discrete approximation, when **exact=FALSE**.

Details

This function computes the area of that part of the disc of radius r centred at the location u that is *not* covered by any of the discs of radius r centred at the points of the pattern X . This area is important in some calculations related to the area-interaction model [AreaInter](#).

If u is a point pattern and r is a vector, the result is a matrix, with one row for each point in u and one column for each entry of r . The $[i, j]$ entry in the matrix is the area of that part of the disc of radius $r[j]$ centred at the location $u[i]$ that is *not* covered by any of the discs of radius $r[j]$ centred at the points of the pattern X .

If W is not NULL, then the areas are computed only inside the window W .

Value

A matrix with one row for each point in u and one column for each value in r .

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[AreaInter](#), [areaLoss](#)

Examples

```
data(cells)
u <- c(0.5,0.5)
areaGain(u, cells, 0.1)
```

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

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

for a point pattern x , where x_1, \dots, 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, \dots, 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

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

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Baddeley, A.J. and Van Lieshout, M.N.M. (1995). Area-interaction point processes. *Annals of the Institute of Statistical Mathematics* **47** (1995) 601–619.
- Widom, B. and Rowlinson, J.S. (1970). New model for the study of liquid-vapor phase transitions. *The Journal of Chemical Physics* **52** (1970) 1670–1684.

See Also

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

Examples

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

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

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

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

# Fit a nonstationary area interaction with log-cubic polynomial trend
## Not run:
ppm(swedishpines, ~polynom(x/10,y/10,3), AreaInter(r=7))

## End(Not run)
```

areaLoss*Difference of Disc Areas***Description**

Computes the area of that part of a disc that is not covered by other discs.

Usage

```
areaLoss(X, r, ..., W=as.owin(X), subset=NULL,
         exact=FALSE,
         ngrid=spatstat.options("ngrid.disc"))
```

Arguments

X	Locations of the centres of discs. A point pattern (object of class "ppp").
r	Disc radius, or vector of disc radii.
...	Ignored.
W	Optional. Window (object of class "owin") inside which the area should be calculated.
subset	Optional. Index identifying a subset of the points of X for which the area difference should be computed.
exact	Choice of algorithm. If exact=TRUE, areas are computed exactly using analytic geometry. If exact=FALSE then a faster algorithm is used to compute a discrete approximation to the areas.
ngrid	Integer. Number of points in the square grid used to compute the discrete approximation, when exact=FALSE.

Details

This function computes, for each point $X[i]$ in X and for each radius r , the area of that part of the disc of radius r centred at the location $X[i]$ that is *not* covered by any of the other discs of radius r centred at the points $X[j]$ for j not equal to i . This area is important in some calculations related to the area-interaction model [AreaInter](#).

The result is a matrix, with one row for each point in X and one column for each entry of r.

Value

A matrix with one row for each point in X (or $X[\text{subset}]$) and one column for each value in r .

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[AreaInter](#), [areaGain](#), [dilated.areas](#)

Examples

```
data(cells)
areaLoss(cells, 0.1)
```

as.box3*Convert Data to Three-Dimensional Box*

Description

Interprets data as the dimensions of a three-dimensional box.

Usage

```
as.box3(...)
```

Arguments

... Data that can be interpreted as giving the dimensions of a three-dimensional box. See Details.

Details

This function converts data in various formats to an object of class "box3" representing a three-dimensional box (see [box3](#)). The arguments ... may be

- an object of class "box3"
- arguments acceptable to [box3](#)
- a numeric vector of length 6, interpreted as `c(xrange[1], xrange[2], yrange[1], yrange[2], zrange[1], zrange[2])`
- an object of class "pp3" representing a three-dimensional point pattern contained in a box.

Value

Object of class "box3".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[box3](#), [pp3](#)

Examples

```
X <- c(0,10,0,10,0,5)
as.box3(X)
X <- pp3(runif(42),runif(42),runif(42), box3(c(0,1)))
as.box3(X)
```

`as.data.frame.hyperframe`

Coerce Hyperframe to Data Frame

Description

Converts a hyperframe to a data frame.

Usage

```
## S3 method for class 'hyperframe'
as.data.frame(x, row.names = NULL,
              optional = FALSE, ...,
              discard=TRUE, warn=TRUE)
```

Arguments

<code>x</code>	Point pattern (object of class "hyperframe").
<code>row.names</code>	Optional character vector of row names.
<code>optional</code>	Argument passed to <code>as.data.frame</code> controlling what happens to row names.
<code>...</code>	Ignored.
<code>discard</code>	Logical. Whether to discard columns of the hyperframe that do not contain atomic data. See Details.
<code>warn</code>	Logical. Whether to issue a warning when columns are discarded.

Details

This is a method for the generic function `as.data.frame` for the class of hyperframes (see [hyperframe](#)). If `discard=TRUE`, any columns of the hyperframe that do not contain atomic data will be removed (and a warning will be issued if `warn=TRUE`). If `discard=FALSE`, then such columns are converted to strings indicating what class of data they originally contained.

Value

A data frame.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

Examples

```
h <- hyperframe(X=1:3, Y=letters[1:3], f=list(sin, cos, tan))
as.data.frame(h, discard=TRUE, warn=FALSE)
as.data.frame(h, discard=FALSE)
```

as.data.frame.im *Convert Pixel Image to Data Frame*

Description

Convert a pixel image to a data frame

Usage

```
## S3 method for class 'im'
as.data.frame(x, ...)
```

Arguments

- x A pixel image (object of class "im").
... Further arguments passed to `as.data.frame.default` to determine the row names and other features.

Details

This function takes the pixel image `x` and returns a data frame with three columns containing the pixel coordinates and the pixel values.

Value

A data frame

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

Examples

```
# artificial image
Z <- setcov(square(1))

Y <- as.data.frame(Z)

head(Y)
```

as.data.frame.ppp*Coerce Point Pattern to a Data Frame*

Description

Extracts the coordinates of the points in a point pattern, and their marks if any, and returns them in a data frame.

Usage

```
## S3 method for class 'ppp'
as.data.frame(x, row.names = NULL, ...)
```

Arguments

- x** Point pattern (object of class "ppp").
- row.names** Optional character vector of row names.
- ...** Ignored.

Details

This is a method for the generic function `as.data.frame` for the class "ppp" of point patterns.

It extracts the coordinates of the points in the point pattern, and returns them as columns named `x` and `y` in a data frame. If the points were marked, the marks are returned as a column named `marks` with the same type as in the point pattern dataset.

Value

A data frame.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

Examples

```
data(amacrine)
df <- as.data.frame(amacrine)
df[1:5,]
```

as.data.frame.psp*Coerce Line Segment Pattern to a Data Frame*

Description

Extracts the coordinates of the endpoints in a line segment pattern, and their marks if any, and returns them in a data frame.

Usage

```
## S3 method for class 'psp'  
as.data.frame(x, row.names = NULL, ...)
```

Arguments

x	Line segment pattern (object of class "psp").
row.names	Optional character vector of row names.
...	Ignored.

Details

This is a method for the generic function `as.data.frame` for the class "psp" of line segment patterns.

It extracts the coordinates of the endpoints of the line segments, and returns them as columns named `x0`, `y0`, `x1` and `y1` in a data frame. If the line segments were marked, the marks are appended as an extra column or columns to the data frame which is returned. If the marks are a vector then a single column named `marks` is appended. in the data frame, with the same type as in the line segment pattern dataset. If the marks are a data frame, then the columns of this data frame are appended (retaining their names).

Value

A data frame with 4 or 5 columns.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

Examples

```
data(copper)  
df <- as.data.frame(copper$Lines)
```

<code>as.function.fv</code>	<i>Convert Function Value Table to Function</i>
-----------------------------	---

Description

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

Usage

```
## S3 method for class 'fv'
as.function(x, ..., value)
```

Arguments

<code>x</code>	Object of class "fv"
<code>...</code>	Ignored.
<code>value</code>	Optional. String selecting one of the columns of <code>x</code> for use as the function value.

Details

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

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

If `f <- as.function(x)` then `f` is a function with one numeric argument, that performs linear interpolation between the values in the table `x`. Argument values lying outside the range of the table yield an NA value.

The command `as.function.fv` is a method for the generic command [as.function](#).

Value

A function with one argument.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[fv](#), [fv.object](#), [plot.fv](#), [Kest](#)

Examples

```
data(cells)
K <- Kest(cells)
f <- as.function(K)
f(0.1)
```

as.hyperframe *Convert Data to Hyperframe*

Description

Converts data from any suitable format into a hyperframe.

Usage

```
as.hyperframe(x, ...)
## Default S3 method:
as.hyperframe(x, ...)
## S3 method for class 'data.frame'
as.hyperframe(x, ..., stringsAsFactors=FALSE)
## S3 method for class 'hyperframe'
as.hyperframe(x, ...)
## S3 method for class 'listof'
as.hyperframe(x, ...)
```

Arguments

x Data in some other format.
... Optional arguments passed to [hyperframe](#).
stringsAsFactors Logical. If TRUE, any column of the data frame x that contains character strings will be converted to a factor. If FALSE, no such conversion will occur.

Details

A hyperframe is like a data frame, except that its entries can be objects of any kind.

The generic function `as.hyperframe` converts any suitable kind of data into a hyperframe.

There are methods for the classes `data.frame` and `listof`, and a default method, all of which convert data that is like a hyperframe into a hyperframe object. (The method for the class `listof` converts a list of objects, of arbitrary type, into a hyperframe with one column.) These methods do not discard any information.

There are also methods for other classes (see [as.hyperframe.ppx](#)) which extract the coordinates from a spatial dataset. These methods do discard some information.

Value

An object of class "hyperframe" created by [hyperframe](#).

Conversion of Strings to Factors

Note that `as.hyperframe.default` will convert a character vector to a factor. It behaves like [as.data.frame](#).

However `as.hyperframe.data.frame` does not convert strings to factors; it respects the structure of the data frame x.

The behaviour can be changed using the argument `stringsAsFactors`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[hyperframe](#), [as.hyperframe.ppx](#)

Examples

```
df <- data.frame(x=runif(4),y=letters[1:4])
as.hyperframe(df)

sims <- list()
for(i in 1:3) sims[[i]] <- rpoispp(42)
as.hyperframe(as.listof(sims))
```

[as.hyperframe.ppx](#)

Extract coordinates and marks of multidimensional point pattern

Description

Given any kind of spatial or space-time point pattern, extract the coordinates and marks of the points.

Usage

```
## S3 method for class 'ppx'
as.hyperframe(x, ...)
## S3 method for class 'ppx'
as.data.frame(x, ...)
## S3 method for class 'ppx'
as.matrix(x, ...)
```

Arguments

x	A general multidimensional space-time point pattern (object of class "ppx").
...	Ignored.

Details

An object of class "ppx" (see [ppx](#)) represents a marked point pattern in multidimensional space and/or time. There may be any number of spatial coordinates, any number of temporal coordinates, and any number of mark variables. The individual marks may be atomic (numeric values, factor values, etc) or objects of any kind.

The function [as.hyperframe.ppx](#) extracts the coordinates and the marks as a "hyperframe" (see [hyperframe](#)) with one row of data for each point in the pattern. This is a method for the generic function [as.hyperframe](#).

The function [as.data.frame.ppx](#) discards those mark variables which are not atomic values, and extracts the coordinates and the remaining marks as a [data.frame](#) with one row of data for each point in the pattern. This is a method for the generic function [as.data.frame](#).

Finally `as.matrix(x)` is equivalent to `as.matrix(as.data.frame(x))` for an object of class "ppx". Be warned that, if there are any columns of non-numeric data (i.e. if there are mark variables that are factors), the result will be a matrix of character values.

Value

A hyperframe, data.frame or matrix as appropriate.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`ppx`, `hyperframe`, `as.hyperframe`.

Examples

```
df <- data.frame(x=runif(4),y=runif(4),t=runif(4))
X <- ppx(data=df, coord.type=c("s","s","t"))
as.data.frame(X)
val <- runif(4)
E <- lapply(val, function(s) { rpoispp(s) })
hf <- hyperframe(t=val, e=as.listof(E))
Z <- ppx(data=hf, domain=c(0,1))
as.hyperframe(Z)
as.data.frame(Z)
```

as.im

Convert to Pixel Image

Description

Converts various kinds of data to a pixel image

Usage

```
as.im(X, ...)
## S3 method for class 'im'
as.im(X, W=NULL, ...,
      eps=NULL, dimyx=NULL, xy=NULL,
      na.replace=NULL)
## S3 method for class 'owin'
as.im(X, W=NULL, ...,
      eps=NULL, dimyx=NULL, xy=NULL,
      na.replace=NULL, value=1)
## S3 method for class 'matrix'
as.im(X, W=NULL, ...)
## S3 method for class 'tess'
as.im(X, W=NULL, ...,
      eps=NULL, dimyx=NULL, xy=NULL,
      na.replace=NULL)
```

```

## S3 method for class 'function'
as.im(X, W=NULL, ...,
      eps=NULL, dimyx=NULL, xy=NULL,
      na.replace=NULL)
## S3 method for class 'distfun'
as.im(X, W=NULL, ...,
      eps=NULL, dimyx=NULL, xy=NULL,
      na.replace=NULL)
## S3 method for class 'leverage.ppm'
as.im(X, ...)
## Default S3 method:
as.im(X, W=NULL, ...,
      eps=NULL, dimyx=NULL, xy=NULL,
      na.replace=NULL)

```

Arguments

X	Data to be converted to a pixel image.
W	Window object which determines the spatial domain and pixel array geometry.
...	Additional arguments passed to X when X is a function.
eps, dimyx, xy	Optional parameters passed to as.mask which determine the pixel array geometry. See as.mask .
na.replace	Optional value to replace NA entries in the output image.
value	Optional. The value to be assigned to pixels inside the window, if X is a window.

Details

This function converts the data X into a pixel image object of class "im" (see [im.object](#)). The function `as.im` is generic, with methods for the classes listed above.

Currently X may be any of the following:

- a pixel image object, of class "im".
- a window object, of class "owin" (see [owin.object](#)). The result is an image with all pixel entries equal to value inside the window X, and NA outside.
- a matrix.
- a tessellation (object of class "tess"). The result is a factor-valued image, with one factor level corresponding to each tile of the tessellation. Pixels are classified according to the tile of the tessellation into which they fall.
- a single number (or a single logical, complex, factor or character value). The result is an image with all pixel entries equal to this constant value inside the window W (and NA outside, unless the argument `na.replace` is given). Argument W is required.
- a function of the form `function(x, y, ...)` which is to be evaluated to yield the image pixel values. In this case, the additional argument W must be present. This window will be converted to a binary image mask. Then the function X will be evaluated in the form `X(x, y, ...)` where x and y are **vectors** containing the x and y coordinates of all the pixels in the image mask, and ... are any extra arguments given. This function must return a vector or factor of the same length as the input vectors, giving the pixel values.
- an object of class "distfun" representing a distance function (created by the command [distfun](#)).

- a list with entries x , y , z in the format expected by the standard R functions `image.default` and `contour.default`. That is, z is a matrix of pixel values, x and y are vectors of x and y coordinates respectively, and $z[i, j]$ is the pixel value for the location $(x[i], y[j])$.
- a point pattern (object of class "ppp"). See the separate documentation for `as.im.ppp`.

The spatial domain (enclosing rectangle) of the pixel image is determined by the argument W . If W is absent, the spatial domain is determined by X . When X is a function, a matrix, or a single numerical value, W is required.

The pixel array dimensions of the final resulting image are determined by (in priority order)

- the argument eps , $dimyx$ or xy if present;
- the pixel dimensions of the window W , if it is present and if it is a binary mask;
- the pixel dimensions of X if it is an image, a binary mask, or a `list(x, y, z)`;
- the default pixel dimensions, controlled by `spatstat.options`.

Note that if eps , $dimyx$ or xy is given, this will override the pixel dimensions of X if it has them. Thus, `as.im` can be used to change an image's pixel dimensions.

If the argument `na.replace` is given, then all NA entries in the image will be replaced by this value. The resulting image is then defined everywhere on the full rectangular domain, instead of a smaller window. Here `na.replace` should be a single value, of the same type as the other entries in the image.

If X is a pixel image that was created by an older version of `spatstat`, the command $X <- \text{as.im}(X)$ will repair the internal format of X so that it conforms to the current version of `spatstat`.

Value

An image object of class "im".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

Separate documentation for `as.im.ppp`

Examples

```
data(demopat)
# window object
W <- demopat>window
plot(W)
Z <- as.im(W)
image(Z)
# function
Z <- as.im(function(x,y) {x^2 + y^2}, unit.square())
image(Z)
# function with extra arguments
f <- function(x, y, x0, y0) {
  sqrt((x - x0)^2 + (y-y0)^2)
}
Z <- as.im(f, unit.square(), x0=0.5, y0=0.5)
```

```

image(Z)
# Revisit the Sixties
data(letterR)
Z <- as.im(f, letterR, x0=2.5, y0=2)
image(Z)
# usual convention in S
stuff <- list(x=1:10, y=1:10, z=matrix(1:100, nrow=10))
Z <- as.im(stuff)
# convert to finer grid
Z <- as.im(Z, dimyx=256)

# pixellate the Dirichlet tessellation
Di <- dirichlet(runifpoint(10))
plot(as.im(Di))
plot(Di, add=TRUE)

```

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[fitin](#), [ppm](#).

Examples

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

as.mask

Pixel Image Approximation of a Window

Description

Obtain a discrete (pixel image) approximation of a given window

Usage

```
as.mask(w, eps=NULL, dimyx=NULL, xy=NULL)
```

Arguments

w	A window (object of class "owin") or data acceptable to as.owin .
eps	(optional) width and height of pixels.
dimyx	(optional) pixel array dimensions
xy	(optional) pixel coordinates

Details

This function generates a rectangular grid of locations in the plane, tests whether each of these locations lies inside the window w, and stores the results as a binary pixel image or ‘mask’ (an object of class “owin”, see [owin.object](#)).

The most common use of this function is to approximate the shape of another window w by a binary pixel image. In this case, we will usually want to have a very fine grid of pixels.

This function can also be used to generate a coarsely-spaced grid of locations inside a window, for purposes such as subsampling and prediction.

The grid spacing and location are controlled by the arguments eps, dimyx and xy, which are mutually incompatible.

If `eps` is given, then it determines the grid spacing. If `eps` is a single number, then the grid spacing will be approximately `eps` in both the `x` and `y` directions. If `eps` is a vector of length 2, then the grid spacing will be approximately `eps[1]` in the `x` direction and `eps[2]` in the `y` direction.

If `dimyx` is given, then the pixel grid will be an $m \times n$ rectangular grid where m, n are given by `dimyx[2], dimyx[1]` respectively. **Warning:** `dimyx[1]` is the number of pixels in the `y` direction, and `dimyx[2]` is the number in the `x` direction.

If `xy` is given, then this should be a structure containing two elements `x` and `y` which are the vectors of `x` and `y` coordinates of the margins of the grid. The pixel coordinates will be generated from these two vectors. In this case `w` may be omitted.

If neither `eps` nor `dimyx` nor `xy` is given, the pixel raster dimensions are obtained from `spatstat.options("npixel")`.

There is no inverse of this function. However, the function `as.polygonal` will compute a polygonal approximation of a binary mask.

Value

A window (object of class "owin") of type "mask" representing a binary pixel image.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`owin.object, as.rectangle, as.polygonal, spatstat.options`

Examples

```
w <- owin(c(0,10),c(0,10), poly=list(x=c(1,2,3,2,1), y=c(2,3,4,6,7)))
## Not run: plot(w)
m <- as.mask(w)
## Not run: plot(m)
x <- 1:9
y <- seq(0.25, 9.75, by=0.5)
m <- as.mask(w, xy=list(x=x, y=y))
```

Description

Converts a line segment pattern to a binary pixel mask by determining which pixels intersect the lines.

Usage

`as.mask.psp(x, W=NULL, ...)`

Arguments

- x Line segment pattern (object of class "psp").
- W Optional window (object of class "owin") determining the pixel raster.
- ... Optional extra arguments passed to `as.mask` to determine the pixel resolution.

Details

This function converts a line segment pattern to a binary pixel mask by determining which pixels intersect the lines.

The pixel raster is determined by W and the optional arguments If W is missing or NULL, it defaults to the window containing x. Then W is converted to a binary pixel mask using `as.mask`. The arguments ... are passed to `as.mask` to control the pixel resolution.

Value

A window (object of class "owin") which is a binary pixel mask (type "mask").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`pixellate.psp`, `as.mask`.

Use `pixellate.psp` if you want to measure the length of line in each pixel.

Examples

```
X <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
plot(as.mask.psp(X))
plot(X, add=TRUE, col="red")
```

as.matrix.im

Convert Pixel Image to Matrix or Array

Description

Converts a pixel image to a matrix or an array.

Usage

```
## S3 method for class 'im'
as.matrix(x, ...)
## S3 method for class 'im'
as.array(x, ...)
```

Arguments

- x A pixel image (object of class "im").
- ... See below.

Details

The function `as.matrix.im` converts the pixel image `x` into a matrix containing the pixel values. It is handy when you want to extract a summary of the pixel values. See the Examples.

The function `as.array.im` converts the pixel image to an array. By default this is a three-dimensional array of dimension n by m by 1. If the extra arguments `...` are given, they will be passed to `array`, and they may change the dimensions of the array.

Value

A matrix or array.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`as.matrix.owin`

Examples

```
# artificial image
Z <- setcov(square(1))

M <- as.matrix(Z)

median(M)

## Not run:
# plot the cumulative distribution function of pixel values
plot(ecdf(as.matrix(Z)))

## End(Not run)
```

`as.matrix.owin`

Convert Pixel Image to Matrix

Description

Converts a pixel image to a matrix.

Usage

```
## S3 method for class 'owin'
as.matrix(x, ...)
```

Arguments

- `x` A window (object of class "owin").
- `...` Arguments passed to `as.mask` to control the pixel resolution.

Details

The function `as.matrix.owin` converts a window to a logical matrix.

It first converts the window x into a binary pixel mask using `as.mask`. It then extracts the pixel entries as a logical matrix.

The resulting matrix has entries that are TRUE if the corresponding pixel is inside the window, and FALSE if it is outside.

The function `as.matrix` is generic. The function `as.matrix.owin` is the method for windows (objects of class "owin").

Use `as.im` to convert a window to a pixel image.

Value

A logical matrix.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`as.matrix.im`, `as.im`

Examples

```
m <- as.matrix(letterR)
```

as.owin

Convert Data To Class owin

Description

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

Usage

```
as.owin(W, ..., fatal=TRUE)

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

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

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

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

```

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

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

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

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

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

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

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

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

## S3 method for class 'rmhmodel'
as.owin(W, ..., fatal=FALSE)

## Default S3 method:
as.owin(W, ..., fatal=TRUE)

```

Arguments

<code>W</code>	Data specifying an observation window, in any of several formats described under <i>Details</i> below.
<code>fatal</code>	Logical flag determining what to do if the data cannot be converted to an observation window. See Details.
<code>...</code>	Ignored.

Details

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

This function converts data in any of several formats into an object of class "owin" for use by the **spatstat** package. 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 four-element vector (interpreted as `(xmin, xmax, ymin, ymax)`) specifying the x and y dimensions of a rectangle

- a structure with entries $x1$, xu , $y1$, yu specifying the x and y dimensions of a rectangle as $(xmin, xmax) = (x1, xu)$ and $(ymin, ymax) = (y1, yu)$. This will accept objects of class `spp` used in the Venables and Ripley **spatial** library.
- an object of class "gpc.poly" from the **gpclib** package, representing a polygonal window.
- 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" or "kppm" representing a fitted point process model. In this case, `as.owin` extracts the original point pattern data to which the model was fitted, and returns the observation window of this point pattern.
- 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.
- an object of class "distfun" representing a distance function. 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.

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`).

The function `as.owin` is generic, with methods for "owin", "im" and "ppp" as well as the default method.

Value

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

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[owin.object](#), [owin](#)

Examples

```
w <- as.owin(c(0,1,0,1))
w <- as.owin(list(xrange=c(0,5),yrange=c(0,10)))
# point pattern
data(demopat)
w <- as.owin(demopat)
# image
Z <- as.im(function(x,y) { x + 3}, unit.square())
w <- as.owin(Z)

# Venables & Ripley 'spatial' package
require(spatial)
towns <- ppinit("towns.dat")
w <- as.owin(towns)
detach(package:spatial)
```

as.polygonal

Convert a Window to a Polygonal Window

Description

Given a window W of any geometric type (rectangular, polygonal or binary mask), this function returns a polygonal window that represents the same spatial domain.

Usage

```
as.polygonal(W)
```

Arguments

W A window (object of class "owin").

Details

Given a window W of any geometric type (rectangular, polygonal or binary mask), this function returns a polygonal window that represents the same spatial domain.

If W is already polygonal, it is returned without change.

If W is a rectangle, it is converted to a polygon with 4 vertices.

If W is a binary mask, then each pixel in the mask is replaced by a small square or rectangle, and the union of these squares or rectangles is computed. The result is a polygonal window that has only horizontal and vertical edges. (Use [simplify.owin](#) to remove the staircase appearance, if desired).

Value

A polygonal window (object of class "owin" and of type "polygonal").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[owin](#), [as.owin](#), [as.mask](#), [simplify.owin](#)

Examples

```
data(letterR)
m <- as.mask(letterR, dimyx=32)
p <- as.polygonal(m)
if(interactive()) {
  plot(m)
  plot(p, add=TRUE, lwd=2)
}
```

as.ppp

Convert Data To Class ppp

Description

Tries to coerce any reasonable kind of data to a point pattern (an object of class "ppp") for use by the **spatstat** package).

Usage

```
as.ppp(X, ..., fatal=TRUE)
## S3 method for class 'ppp'
as.ppp(X, ..., fatal=TRUE)
## S3 method for class 'psp'
as.ppp(X, ..., fatal=TRUE)
## S3 method for class 'quad'
as.ppp(X, ..., fatal=TRUE)
## S3 method for class 'matrix'
as.ppp(X, W=NULL, ..., fatal=TRUE)
## S3 method for class 'data.frame'
as.ppp(X, W=NULL, ..., fatal=TRUE)
## S3 method for class 'influence.ppm'
as.ppp(X, ...)
## Default S3 method:
as.ppp(X, W=NULL, ..., fatal=TRUE)
```

Arguments

- | | |
|--------------|---|
| X | Data which will be converted into a point pattern |
| W | Data which define a window for the pattern when X does not contain a window |
| ... | Ignored. |
| fatal | Logical value. See Details. |

Details

Converts the dataset X to a point pattern (an object of class "ppp"; see [ppp.object](#) for an overview).

This function is normally used to convert an existing point pattern dataset, stored in another format, to the "ppp" format. To create a new point pattern from raw data such as x, y coordinates, it is normally easier to use the creator function [ppp](#).

The dataset X may be:

- an object of class "ppp"
- an object of class "psp"
- an object of class "spp" as defined in the **spatial** library
- an object of class "quad" representing a quadrature scheme (see [quad.object](#))
- a matrix or data frame with at least two columns
- a structure with entries x, y which are numeric vectors of equal length
- a numeric vector of length 2, interpreted as the coordinates of a single point.

In the last three cases, we need the second argument W which is converted to a window object by the function [as.owin](#). In the first four cases, W will be ignored.

If X is a line segment pattern (an object of class psp) the point pattern returned consists of the endpoints of the segments. If X is marked then the point pattern returned will also be marked, the mark associated with a point being the mark of the segment of which that point was an endpoint.

If X is a matrix or data frame, the first and second columns will be interpreted as the x and y coordinates respectively. Any additional columns will be interpreted as marks.

The argument `fatal` indicates what to do when W is missing and X contains no information about the window. If `fatal=TRUE`, a fatal error will be generated; if `fatal=FALSE`, the value `NULL` is returned.

An spp object is the representation of a point pattern in the **spatial** library. Our implementation recognises the following formats:

- a structure with entries x, y, xl, xu, yl, yu
- a structure with entries x, y and `area`, where `area` is a structure with entries xl, xu, yl, yu

(used in **spatial** versions 1 to 6 and version 7.1 respectively) where x and y are vectors of equal length giving the point coordinates, and xl, xu, yl, yu are numbers giving the dimensions of a rectangular window.

The function `as.ppp` is generic, with methods for the classes "ppp", "psp", "quad", "matrix", "data.frame" and a default method.

Point pattern datasets can also be created by the function [ppp](#).

Value

An object of class "ppp" (see [ppp.object](#)) describing the point pattern and its window of observation. The value `NULL` may also be returned; see Details.

Warnings

If the format of spp objects is changed in future versions of the **spatial** library, then `as.ppp` may not be able to interpret them. It currently handles all versions of **spatial** up to 7.1-4.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppp](#), [ppp.object](#), [as.owin](#), [owin.object](#)

Examples

```
xy <- matrix(runif(40), ncol=2)
pp <- as.ppp(xy, c(0,1,0,1))

# Venables-Ripley format
# check for 'spatial' package
spatialpath <- .find.package("spatial", quiet=TRUE)
if(length(spatialpath) != 0) {
  require(spatial)
  towns <- ppinit("towns.dat")
  pp <- as.ppp(towns) # converted to our format
  detach(package:spatial)
}

xyzt <- matrix(runif(40), ncol=4)
Z <- as.ppp(xyzt, square(1))
```

as.psp

Convert Data To Class psp

Description

Tries to coerce any reasonable kind of data object to a line segment pattern (an object of class "psp") for use by the **spatstat** package.

Usage

```
as.psp(x, ..., from=NULL, to=NULL)
## S3 method for class 'psp'
as.psp(x, ..., check=FALSE, fatal=TRUE)
## S3 method for class 'data.frame'
as.psp(x, ..., window=NULL, marks=NULL, check=spatstat.options("checksegments"), fatal=TRUE)
## S3 method for class 'matrix'
as.psp(x, ..., window=NULL, marks=NULL, check=spatstat.options("checksegments"),
fatal=TRUE)
## S3 method for class 'owin'
as.psp(x, ..., check=spatstat.options("checksegments"), fatal=TRUE)
## Default S3 method:
as.psp(x, ..., window=NULL, marks=NULL,
check=spatstat.options("checksegments"), fatal=TRUE)
```

Arguments

<code>x</code>	Data which will be converted into a line segment pattern
<code>window</code>	Data which define a window for the pattern when <code>x</code> does not contain a window
<code>...</code>	Ignored.
<code>marks</code>	(Optional) vector or data frame of marks for the pattern
<code>check</code>	Logical value indicating whether to check the validity of the data, e.g. to check that the line segments lie inside the window.
<code>fatal</code>	Logical value. See Details.
<code>from, to</code>	Point patterns (object of class "ppp") containing the first and second endpoints (respectively) of each segment. Incompatible with <code>x</code> .

Details

Converts the dataset `x` to a line segment pattern (an object of class "psp"; see [psp.object](#) for an overview).

This function is normally used to convert an existing line segment pattern dataset, stored in another format, to the "psp" format. To create a new point pattern from raw data such as x, y coordinates, it is normally easier to use the creator function [psp](#).

The dataset `x` may be:

- an object of class "psp"
- a data frame with at least 4 columns
- a structure (list) with elements named `x0, y0, x1, y1` or elements named `xmid, ymid, length, angle` and possibly a fifth element named `marks`
- an object of class "owin" representing a spatial window; it must be of type "rectangle" or "polygonal". The boundary edges of the window will be extracted as a line segment pattern.

If `x` is a data frame the interpretation of its columns is as follows:

- If there are columns named `x0, y0, x1, y1` then these will be interpreted as the coordinates of the endpoints of the segments and used to form the `ends` component of the psp object to be returned.
- If there are columns named `xmid, ymid, length, angle` then these will be interpreted as the coordinates of the segment midpoints, the lengths of the segments, and the orientations of the segments in radians and used to form the `ends` component of the psp object to be returned.
- If there is a column named `marks` then this will be interpreted as the marks of the pattern provided that the argument `marks` of this function is NULL. If argument `marks` is not NULL then the value of this argument is taken to be the marks of the pattern and the column named `marks` is ignored (with a warning). In either case the column named `marks` is deleted and omitted from further consideration.
- If there is no column named `marks` and if the `marks` argument of this function is NULL, and if after interpreting 4 columns of `x` as determining the `ends` component of the psp object to be returned, there remain other columns of `x`, then these remaining columns will be taken to form a data frame of marks for the psp object to be returned.

If `x` is a structure (list) with elements named `x0, y0, x1, y1, marks` or `xmid, ymid, length, angle, marks`, then the element named `marks` will be interpreted as the marks of the pattern provide that the argument `marks` of this function is NULL. If this argument is non-NULL then it is interpreted as the marks of the pattern and the element `marks` of `x` is ignored — with a warning.

Alternatively, you may specify two point patterns `from` and `to` containing the first and second endpoints of the line segments.

The argument `window` is converted to a window object by the function [as.owin](#).

The argument `fatal` indicates what to do when the data cannot be converted to a line segment pattern. If `fatal=TRUE`, a fatal error will be generated; if `fatal=FALSE`, the value `NULL` is returned.

The function `as.psp` is generic, with methods for the classes "`psp`", "`data.frame`", "`matrix`" and a default method.

Point pattern datasets can also be created by the function [psp](#).

Value

An object of class "`psp`" (see [psp.object](#)) describing the line segment pattern and its window of observation. The value `NULL` may also be returned; see Details.

Warnings

If only a proper subset of the names `x0,y0,x1,y1` or `xmid,ymid,length,angle` appear amongst the names of the columns of `x` where `x` is a data frame, then these special names are ignored.

For example if the names of the columns were `xmid,ymid,length,degrees`, then these columns would be interpreted as if the represented `x0,y0,x1,y1` in that order.

Whether it gets used or not, column named `marks` is *always* removed from `x` before any attempt to form the `ends` component of the `psp` object that is returned.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[psp](#), [psp.object](#), [as.owin](#), [owin.object](#)

Examples

```
mat <- matrix(runif(40), ncol=4)
mx <- data.frame(v1=sample(1:4,10,TRUE),
                  v2=factor(sample(letters[1:4],10,TRUE),levels=letters[1:4]))
a <- as.psp(mat, window=owin(),marks=mx)
mat <- cbind(as.data.frame(mat),mx)
b <- as.psp(mat, window=owin()) # a and b are identical.
stuff <- list(xmid=runif(10),
              ymid=runif(10),
              length=rep(0.1, 10),
              angle=runif(10, 0, 2 * pi))
a <- as.psp(stuff, window=owin())
b <- as.psp(from=runifpoint(10), to=runifpoint(10))
```

`as.rectangle` *Window Frame*

Description

Extract the window frame of a window or other spatial dataset

Usage

```
as.rectangle(w, ...)
```

Arguments

- w A window, or a dataset that has a window. Either a window (object of class "owin"), a pixel image (object of class "im") or other data determining such a window.
- ... Optional. Auxiliary data to help determine the window. If w does not belong to a recognised class, the arguments w and ... are passed to `as.owin` to determine the window.

Details

This function is the quickest way to determine a bounding rectangle for a spatial dataset.

If w is a window, the function just extracts the outer bounding rectangle of w as given by its elements `xrange`, `yrange`.

The function can also be applied to any spatial dataset that has a window: for example, a point pattern (object of class "ppp") or a line segment pattern (object of class "psp"). The bounding rectangle of the window of the dataset is extracted.

Use the function `bounding.box` to compute the *smallest* bounding rectangle of a dataset.

Value

A window (object of class "owin") of type "rectangle" representing a rectangle.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`owin`, `as.owin`, `bounding.box`

Examples

```
w <- owin(c(0,10),c(0,10), poly=list(x=c(1,2,3,2,1), y=c(2,3,4,6,7)))
r <- as.rectangle(w)
# returns a 10 x 10 rectangle

data(lansing)
as.rectangle(lansing)
```

```
data(copper)
as.rectangle(copper$SouthLines)
```

as.tess*Convert Data To Tessellation***Description**

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

Usage

```
as.tess(X)
## S3 method for class 'tess'
as.tess(X)
## S3 method for class 'im'
as.tess(X)
## S3 method for class 'owin'
as.tess(X)
## S3 method for class 'quadratcount'
as.tess(X)
## S3 method for class 'quadrattest'
as.tess(X)
## S3 method for class 'list'
as.tess(X)
```

Arguments

X Data to be converted to a tessellation.

Details

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

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

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

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

Value

An object of class "tess" specifying a tessellation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[tess](#)

Examples

```
# pixel image
v <- as.im(function(x,y){factor(round(5 * (x^2 + y^2)))}, W=owin())
levels(v) <- letters[seq(length(levels(v)))]
as.tess(v)
# quadrat counts
data(nztrees)
qNZ <- quadratcount(nztrees, nx=4, ny=3)
as.tess(qNZ)
```

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

<code>r</code>	vector of interaction radii
<code>sat</code>	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, \dots, r_k , saturation thresholds s_1, \dots, s_k , intensity parameter β and interaction parameters $\gamma_1, \dots, gamma_k$, is the point process in which each point x_i in the pattern X contributes a factor

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

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.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz> in collaboration with Hao Wang and Jeff Picka

See Also

[ppm](#), [pairsat.family](#), [Geyer](#), [PairPiece](#), [SatPiece](#)

Examples

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

# 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
## Not run:
ppm(cells, ~polynom(x,y,3), BadGey(c(0.07, 0.1, 0.13), 2))

## End(Not run)
```

bdist.pixels*Distance to Boundary of Window***Description**

Computes the distances from each pixel in a window to the boundary of the window.

Usage

```
bdist.pixels(w, ..., style="image")
```

Arguments

w	A window (object of class "owin").
...	Arguments passed to <code>as.mask</code> to determine the pixel resolution.
style	Character string determining the format of the output: either "matrix", "coords" or "image".

Details

This function computes, for each pixel u in the window w , the shortest distance $d(u, W^c)$ from u to the boundary of W .

If the window is not of type "mask" then it is first converted to that type. The arguments "..." are passed to `as.mask` to determine the pixel resolution.

Value

If `style="image"`, a pixel image (object of class "im") containing the distances from each pixel in the image raster to the boundary of the window.

If `style="matrix"`, a matrix giving the distances from each pixel in the image raster to the boundary of the window. Rows of this matrix correspond to the y coordinate and columns to the x coordinate.

If `style="coords"`, a list with three components x, y, z , where x, y are vectors of length m, n giving the x and y coordinates respectively, and z is an $m \times n$ matrix such that $z[i, j]$ is the distance from $(x[i], y[j])$ to the boundary of the window. Rows of this matrix correspond to the x coordinate and columns to the y coordinate. This result can be plotted with `persp`, `image` or `contour`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`owin.object`, `erosion`, `bdist.points`, `bdist.tiles`.

Examples

```
u <- owin(c(0,1),c(0,1))
d <- bdist.pixels(u, eps=0.01)
image(d)
d <- bdist.pixels(u, eps=0.01, style="matrix")
mean(d >= 0.1)
# value is approx (1 - 2 * 0.1)^2 = 0.64
```

bdist.points

Distance to Boundary of Window

Description

Computes the distances from each point of a point pattern to the boundary of the window.

Usage

```
bdist.points(X)
```

Arguments

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

Details

This function computes, for each point x_i in the point pattern X, the shortest distance $d(x_i, W^c)$ from x_i to the boundary of the window W of observation.

If the window X>window is of type "rectangle" or "polygonal", then these distances are computed by analytic geometry and are exact, up to rounding errors. If the window is of type "mask" then the distances are computed using the real-valued distance transform, which is an approximation with maximum error equal to the width of one pixel in the mask.

Value

A numeric vector, giving the distances from each point of the pattern to the boundary of the window.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[bdist.pixels](#), [bdist.tiles](#), [ppp.object](#), [erosion](#)

Examples

```
data(cells)
d <- bdist.points(cells)
```

bdist.tiles*Distance to Boundary of Window***Description**

Computes the shortest distances from each tile in a tessellation to the boundary of the window.

Usage

```
bdist.tiles(X)
```

Arguments

X	A tessellation (object of class "tess").
---	--

Details

This function computes, for each tile s_i in the tessellation X, the shortest distance from s_i to the boundary of the window W containing the tessellation.

Value

A numeric vector, giving the shortest distance from each tile in the tessellation to the boundary of the window. Entries of the vector correspond to the entries of tiles(X).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[tess](#), [bdist.points](#), [bdist.pixels](#)

Examples

```
P <- runifpoint(15)
X <- dirichlet(P)
plot(X, col="red")
B <- bdist.tiles(X)
# identify tiles that do not touch the boundary
plot(X[B > 0], add=TRUE, col="green", lwd=3)
```

bei	<i>Tropical rain forest trees</i>
-----	-----------------------------------

Description

A point pattern giving the locations of 3605 trees in a tropical rain forest. Accompanied by covariate data giving the elevation (altitude) and slope of elevation in the study region.

Usage

```
data(bei)
```

Format

bei is an object of class "ppp" representing the point pattern of tree locations. See [ppp.object](#) for details of the format.

bei.extra is a list containing two pixel images, elev (elevation in metres) and grad (norm of elevation gradient). These pixel images are objects of class "im", see [im.object](#).

Notes

The dataset bei gives the positions of 3605 trees of the species *Beilschmiedia pendula* (Lauraceae) in a 1000 by 500 metre rectangular sampling region in the tropical rainforest of Barro Colorado Island.

The accompanying dataset bei.extra gives information about the altitude (elevation) in the study region. It is a list containing two pixel images, elev (elevation in metres) and grad (norm of elevation gradient).

These data are part of a much larger dataset containing the positions of hundreds of thousands of trees belong to thousands of species; see Hubbell and Foster (1983), Condit, Hubbell and Foster (1996) and Condit (1998).

The present data were analysed by Moller and Waagepetersen (2007).

Source

Hubbell and Foster (1983), Condit, Hubbell and Foster (1996) and Condit (1998). Data files kindly supplied by Rasmus Waagepetersen. The data were collected in the forest dynamics plot of Barro Colorado Island. The study was made possible through the generous support of the U.S. National Science Foundation, the John D. and Catherine T. MacArthur Foundation, and the Smithsonian Tropical Research Institute.

References

- Condit, R. (1998) *Tropical Forest Census Plots*. Springer-Verlag, Berlin and R.G. Landes Company, Georgetown, Texas.
- Condit, R., Hubbell, S.P and Foster, R.B. (1996) Changes in tree species abundance in a neotropical forest: impact of climate change. *Journal of Tropical Ecology* **12**, 231–256.
- Hubbell, S.P and Foster, R.B. (1983) Diversity of canopy trees in a neotropical forest and implications for conservation. In: *Tropical Rain Forest: Ecology and Management* (eds. S.L. Sutton, T.C. Whitmore and A.C. Chadwick), Blackwell Scientific Publications, Oxford, 25–41.

Moller, J. and Waagepetersen, R.P. (2007) Modern spatial point process modelling and inference (with discussion). *Scandinavian Journal of Statistics* **34**, 643–711.

bermantest

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

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

Arguments

X	A point pattern (object of class "ppp").
model	A fitted point process model (object of class "ppm").
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.
...	Ignored.

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 function `bermantest` is generic, with methods for point patterns ("ppp") and point process models ("ppm").

- If X is a point pattern dataset (object of class "ppp"), then `bermantest(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") then `bermantest(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 $Z_1 = (S - \mu)/\sigma$.
- 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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

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

See Also

`kstest`, `quadrat.test`, `ppm`

Examples

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

betacells

Beta Ganglion Cells in Cat Retina

Description

Point pattern of cells in the retina, each cell classified as ‘on’ or ‘off’. A bivariate point pattern.

Usage

```
data(betacells)
```

Format

`betacells` is an object of class “`ppp`” representing the point pattern of cell locations. Entries include

<code>x</code>	Cartesian <i>x</i> -coordinate of cell
<code>y</code>	Cartesian <i>y</i> -coordinate of cell
<code>marks</code>	factor with levels <code>off</code> and <code>on</code> indicating “off” and “on” cells

See [ppp.object](#) for details of the format. Cartesian coordinates are given in microns.

`betacells.extra` is a list with one component `area` which is the vector of areas (in square microns) of the cells in the pattern.

Notes

This is a new, corrected version of the old dataset [ganglia](#). See below.

These data represent a pattern of beta-type ganglion cells in the retina of a cat recorded by W\"assle et al. (1981). Beta cells are associated with the resolution of fine detail in the cat’s visual system. They can be classified anatomically as “on” or “off”.

Statistical independence of the arrangement of the “on”- and “off”-components would strengthen the evidence for Hering’s (1878) ‘opponent theory’ that there are two separate channels for sensing “brightness” and “darkness”. See W\"assle et al (1981). There is considerable current interest in the arrangement of cell mosaics in the retina, see Rockhill et al (2000).

The dataset is a multitype point pattern giving the locations and types (“on” or “off”) of beta cells observed in a rectangle of dimensions 750×990 microns. Coordinates are given in microns (thousandths of a millimetre).

The original source is Figure 6 of W\"assle et al (1981), which is a manual drawing of the beta mosaic observed in a microscope field-of-view of a whole mount of the retina. Thus, all beta cells in the retina were effectively projected onto the same two-dimensional plane.

The data were scanned in 2004 by Stephen Eglen from Figure 6(a) of W\"assle et al (1981). Image analysis software was used to identify the soma (cell body). The x, y location of each cell was taken to be the centroid of the soma. The type of each cell ("on" or "off") was identified by referring to Figures 6(b) and 6(d).

The area of each soma (in square microns) was also computed, and is provided in the dataset `betacells.extra`.

Note that this is a corrected version of the `ganglia` dataset provided in earlier versions of `spatstat`. The earlier data `ganglia` were not faithful to the scale in the original paper and contain some scanning errors.

Source

W\"assle et al (1981), Figure 6(a), scanned and processed by Stephen Eglen <S.J.Eglen@damtp.cam.ac.uk>

References

- Hering, E. (1878) Zur Lehre von Lichtsinn. Vienna.
- Van Lieshout, M.N.M. and Baddeley, A.J. (1999) Indices of dependence between types in multivariate point patterns. *Scandinavian Journal of Statistics* **26**, 511–532.
- Rockhill, R.L., Euler, T. and Masland, R.H. (2000) Spatial order within but not between types of retinal neurons. *Proc. Nat. Acad. Sci. USA* **97**(5), 2303–2307.
- W\"assle, H., Boycott, B. B. & Illing, R.-B. (1981). Morphology and mosaic of on- and off-beta cells in the cat retina and some functional considerations. *Proc. Roy. Soc. London Ser. B* **212**, 177–195.

Examples

```
data(betacells)
plot(betacells)
plot(betacells$window, main="beta cells")
symbols(betacells$x, betacells$y,
        circles=sqrt(betacells.extra$area/pi),
        inches=FALSE, add=TRUE)
```

Description

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

Usage

```
## S3 method for class 'fv'
cbind(...)
bind.fv(x, y, labl = NULL, desc = NULL, preferred = NULL)
```

Arguments

...	Any number of arguments, which are objects of class "fv".
x	An object of class "fv".
y	Either a data frame or an object of class "fv".
labl	Plot labels (see fv) for columns of y. A character vector.
desc	Descriptions (see fv) for columns of y. A character vector.
preferred	Character string specifying the column which is to be the new recommended value of the function.

Details

This documentation is provided for experienced programmers who want to modify the internal behaviour of [spatstat](#).

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

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

- If the additional dataset `y` is an object of class "fv", then `x` and `y` must be compatible as described above. Then the columns of `y` that contain function values will be appended to the object `x`.
- Alternatively if `y` is a data frame, then `y` must have the same number of rows as `x`. All columns of `y` will be appended to `x`.

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

Value

An object of class "fv".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[fv](#), [with.fv](#).

Undocumented functions for modifying an "fv" object include `fvnames`, `fvnames<-`, `tweak.fv.entry` and `rebadge.fv`.

Examples

```
data(cells)
K1 <- Kest(cells, correction="border")
K2 <- Kest(cells, correction="iso")
# remove column 'theo' to avoid duplication
K2 <- K2[, names(K2) != "theo"]
```

```

cbind(K1, K2)

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

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

```

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

Applies a Gaussian blur to a pixel image.

Usage

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

Arguments

x	The pixel image. An object of class "im".
sigma	Standard deviation of isotropic Gaussian smoothing kernel.
...	Ignored.
normalise	Logical flag indicating whether the output values should be divided by the corresponding blurred image of the window itself. See Details.
bleed	Logical flag indicating whether to allow blur to extend outside the original domain of the image. See Details.
varcov	Variance-covariance matrix of anisotropic Gaussian kernel. Incompatible with sigma .

Details

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

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

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

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

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

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

Computation is performed using the Fast Fourier Transform.

Value

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

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[interp.im](#) for interpolating a pixel image to a finer resolution, [density.ppp](#) for blurring a point pattern, [smooth.ppp](#) for interpolating marks attached to points.

Examples

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

border

Border Region of a Window

Description

Computes the border region of a window, that is, the region lying within a specified distance of the boundary of a window.

Usage

```
border(w, r, outside=FALSE, ...)
```

Arguments

<code>w</code>	A window (object of class "owin") or something acceptable to as.owin .
<code>r</code>	Numerical value.
<code>outside</code>	Logical value determining whether to compute the border outside or inside <code>w</code> .
<code>...</code>	Optional arguments passed to erosion (if <code>outside=FALSE</code>) or to dilation (if <code>outside=TRUE</code>).

Details

By default (if `outside=FALSE`), the border region is the subset of w lying within a distance r of the boundary of w . It is computed by eroding w by the distance r (using [erosion](#)) and subtracting this eroded window from the original window w .

If `outside=TRUE`, the border region is the set of locations outside w lying within a distance r of w . It is computed by dilating w by the distance r (using [dilation](#)) and subtracting the original window w from the dilated window.

Value

A window (object of class "owin").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[erosion](#), [dilation](#)

Examples

```
# rectangle
u <- unit.square()
border(u, 0.1)
border(u, 0.1, outside=TRUE)

# polygon
data(letterR)
plot(letterR)
plot(border(letterR, 0.1), add=TRUE)
plot(border(letterR, 0.1, outside=TRUE), add=TRUE)
```

bounding.box

Bounding Box of a Window or Point Pattern

Description

Find the smallest rectangle containing a given window(s) or point pattern(s).

Usage

`bounding.box(...)`

Arguments

`...` One or more windows (objects of class "owin"), pixel images (objects of class "im") or point patterns (objects of class "ppp").

Details

This function finds the smallest rectangle (with sides parallel to the coordinate axes) that contains all the given objects.

For a window (object of class "owin"), the bounding box is the smallest rectangle that contains all the vertices of the window (this is generally smaller than the enclosing frame, which is returned by [as.rectangle](#)).

For a point pattern (object of class "ppp"), the bounding box is the smallest rectangle that contains all the points of the pattern, and is computed by [bounding.box.xy](#).

For a pixel image (object of class "im"), the image will be converted to a window using [as.owin](#), and the bounding box of this window is obtained.

If the argument is a list of several objects, then this function finds the smallest rectangle that contains all the bounding boxes of the objects.

Value

A window (object of class "owin") of type "rectangle" representing a rectangle.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[owin](#), [as.owin](#), [as.rectangle](#), [bounding.box.xy](#)

Examples

```
w <- owin(c(0,10),c(0,10), poly=list(x=c(1,2,3,2,1), y=c(2,3,4,6,7)))
r <- bounding.box(w)
# returns rectangle [1,3] x [2,7]

w2 <- unit.square()
r <- bounding.box(w, w2)
# returns rectangle [0,3] x [0,7]
```

Description

Computes the smallest rectangle containing a set of points.

Usage

`bounding.box.xy(x, y=NULL)`

Arguments

- x vector of x coordinates of observed points, or a 2-column matrix giving x,y coordinates, or a list with components x,y giving coordinates (such as a point pattern object of class "ppp".)
- y (optional) vector of y coordinates of observed points, if x is a vector.

Details

Given an observed pattern of points with coordinates given by x and y, this function finds the smallest rectangle, with sides parallel to the coordinate axes, that contains all the points, and returns it as a window.

Value

A window (an object of class "owin").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[owin](#), [as.owin](#), [convexhull.xy](#), [ripras](#)

Examples

```
x <- runif(30)
y <- runif(30)
w <- bounding.box.xy(x,y)
plot(owin(), main="bounding.box.xy(x,y)")
plot(w, add=TRUE)
points(x,y)

X <- rpoispp(30)
plot(X, main="bounding.box.xy(X)")
plot(bounding.box.xy(X), add=TRUE)
```

Description

Creates an object representing a three-dimensional box.

Usage

```
box3(xrange = c(0, 1), yrange = xrange, zrange = yrange, unitname = NULL)
```

Arguments

- `xrange, yrange, zrange`
 Dimensions of the box in the x, y, z directions. Each of these arguments should be a numeric vector of length 2.
- `unitname` Optional. Name of the unit of length. See Details.

Details

This function creates an object representing a three-dimensional rectangular parallelepiped (box) with sides parallel to the coordinate axes.

The object can be used to specify the domain of a three-dimensional point pattern (see `pp3`) and in various geometrical calculations (see `volume.box3`, `diameter.box3`, `eroded.volumes`).

The optional argument `unitname` specifies the name of the unit of length. See `unitname` for valid formats.

The function `as.box3` can be used to convert other kinds of data to this format.

Value

An object of class "box3". There is a print method for this class.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`as.box3`, `pp3`, `volume.box3`, `diameter.box3`, `eroded.volumes`.

Examples

```
box3()
box3(c(0,10),c(0,10),c(0,5), unitname=c("metre","metres"))
box3(c(-1,1))
```

Description

Creates an object representing a multi-dimensional box.

Usage

```
boxx(..., unitname = NULL)
```

Arguments

- `...` Dimensions of the box. Vectors of length 2.
- `unitname` Optional. Name of the unit of length. See Details.

Details

This function creates an object representing a multi-dimensional rectangular parallelepiped (box) with sides parallel to the coordinate axes.

The object can be used to specify the domain of a multi-dimensional point pattern (see [ppx](#)) and in various geometrical calculations (see [volume.boxx](#), [diameter.boxx](#), [eroded.volumes](#)).

The optional argument `unitname` specifies the name of the unit of length. See [unitname](#) for valid formats.

Value

An object of class "boxx". There is a print method for this class.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppx](#), [volume.boxx](#), [diameter.boxx](#), [eroded.volumes.boxx](#).

Examples

```
boxx(c(0,10),c(0,10),c(0,5),c(0,1), unitname=c("metre","metres"))
```

bramblecanes

Hutchings' Bramble Canes data

Description

Data giving the locations and ages of bramble canes in a field. A marked point pattern.

Usage

```
data(bramblecanes)
```

Format

An object of class "ppp" representing the point pattern of plant locations. Entries include

x	Cartesian x -coordinate of plant
y	Cartesian y -coordinate of plant
marks	factor with levels 0,1, 2 indicating age

See [ppp.object](#) for details of the format.

Notes

These data record the (x, y) locations and ages of bramble canes in a field 9 metres square, rescaled to the unit square. The canes were classified according to age as either newly emergent, one or two years old. These are encoded as marks 0, 1 and 2 respectively in the dataset.

The data were recorded and analysed by Hutchings (1979) and further analysed by Diggle (1981a, 1981b, 1983), Diggle and Milne (1983), and Van Lieshout and Baddeley (1999). All analyses found that the pattern of newly emergent canes exhibits clustering, which Hutchings attributes to “vigorous vegetative reproduction”.

Source

Hutchings (1979), data published in Diggle (1983)

References

- Diggle, P. J. (1981a) Some graphical methods in the analysis of spatial point patterns. In *Interpreting multivariate data*, V. Barnett (Ed.) John Wiley and Sons.
- Diggle, P. J. (1981b). Statistical analysis of spatial point patterns. *N.Z. Statist.* **16**, 22–41.
- Diggle, P.J. (1983) *Statistical analysis of spatial point patterns*. Academic Press.
- Diggle, P. J. and Milne, R. K. (1983) Bivariate Cox processes: some models for bivariate spatial point patterns. *Journal of the Royal Statistical Soc. Series B* **45**, 11–21.
- Hutchings, M. J. (1979) Standing crop and pattern in pure stands of Mercurialis perennis and Rubus fruticosus in mixed deciduous woodland. *Oikos* **31**, 351–357.
- Van Lieshout, M.N.M. and Baddeley, A.J. (1999) Indices of dependence between types in multivariate point patterns. *Scandinavian Journal of Statistics* **26**, 511–532.

bronzefilter

Bronze gradient filter data

Description

These data represent a spatially inhomogeneous pattern of circular section profiles of particles, observed in a longitudinal plane section through a gradient sinter filter made from bronze powder, prepared by Ricardo Bernhardt, Dresden.

The material was produced by sedimentation of bronze powder with varying grain diameter and subsequent sintering, as described in Bernhardt et al. (1997).

The data are supplied as a marked point pattern of circle centres marked by circle radii. The coordinates of the centres and the radii are recorded in mm. The field of view is an 18×7 mm rectangle.

The data were first analysed by Hahn et al. (1999).

Usage

```
data(bronzefilter)
```

Format

An object of class "ppp" representing the point pattern of cell locations. Entries include

x	Cartesian x -coordinate of bronze grain profile centre
y	Cartesian y -coordinate of bronze grain profile centre
marks	radius of bronze grain profile

See [ppp.object](#) for details of the format. All coordinates are recorded in mm.

Source

R.\ Bernhardt (section image), H.\ Wendrock (coordinate measurement). Adjusted, formatted and communicated by U.\ Hahn.

References

- Bernhardt, R., Meyer-Olbersleben, F. and Kieback, B. (1997) Fundamental investigation on the preparation of gradient structures by sedimentation of different powder fractions under gravity. *Proc. of the 4th Int. Conf. On Composite Engineering, July 6–12 1997, ICCE/4*, Hawaii, Ed. David Hui, 147–148.
- Hahn U., Micheletti, A., Pohlink, R., Stoyan D. and Wendrock, H.(1999) Stereological analysis and modelling of gradient structures. *Journal of Microscopy*, **195**, 113–124.

Examples

```
data(bronzefilter)
plot(bronzefilter, markscale=1)
```

bw.diggle

Cross Validated Bandwidth Selection for Kernel Density

Description

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

Usage

```
bw.diggle(X)
```

Arguments

- | | |
|---|--|
| X | A point pattern (object of class "ppp"). |
|---|--|

Details

This function selects an appropriate bandwidth *sigma* for the kernel estimator of point process intensity computed by [density.ppp](#).

The bandwidth σ is chosen to minimise the mean-square error criterion defined by Diggle (1985). The algorithm computes the mean-square error by the method of Berman and Diggle (1989). See Diggle (2003, pages 115-118) for a summary of this method.

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

Value

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

Definition of bandwidth

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

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Berman, M. and Diggle, P. (1989) Estimating weighted integrals of the second-order intensity of a spatial point process. *Journal of the Royal Statistical Society, series B* **51**, 81–92.
- Diggle, P.J. (1985) A kernel method for smoothing point process data. *Applied Statistics (Journal of the Royal Statistical Society, Series C)* **34** (1985) 138–147.
- Diggle, P.J. (2003) *Statistical analysis of spatial point patterns*, Second edition. Arnold.

See Also

`density.ppp`, `bw.scott`

Examples

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

plot(density(hickory, b))
```

Description

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

Usage

```
bw.relrisk(X, method = "likelihood", nh = spatstat.options("n.bandwidth"),
hmin=NULL, hmax=NULL, warn=TRUE)
```

Arguments

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

Details

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

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

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

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

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

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

Value

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

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Diggle, P.J. (2003) *Statistical analysis of spatial point patterns*, Second edition. Arnold.
 Kelsall, J.E. and Diggle, P.J. (1995) Kernel estimation of relative risk. *Bernoulli* **1**, 3–16.

See Also

`relrisk`, `bw.stoyan`

Examples

```
data(urkiola)

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

bw.scott

Scott's Rule for Bandwidth Selection for Kernel Density

Description

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

Usage

```
bw.scott(X)
```

Arguments

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

Details

This function selects a bandwidth σ for the kernel estimator of point process intensity computed by [density.ppp](#).

The bandwidth σ is computed by the rule of thumb of Scott (1992, page 152). It is very fast to compute.

This rule is designed for density estimation, and typically produces a larger bandwidth than [bw.diggle](#). It is useful for estimating gradual trend.

Value

A numerical vector of two elements giving the selected bandwidths in the x and y directions.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

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

See Also

[density.ppp](#), [bw.diggle](#).

Examples

```
data(lansing)
attach(split(lansing))
b <- bw.scott(hickory)
b

plot(density(hickory, b))
```

bw.smoothppp

Cross Validated Bandwidth Selection for Spatial Smoothing

Description

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

Usage

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

Arguments

X	A marked point pattern with numeric marks.
nh	Number of trial values of smoothing bandwidth <code>sigma</code> to consider. The default is 32.
hmin, hmax	Optional. Numeric values. Range of trial values of smoothing bandwidth <code>sigma</code> to consider. There is a sensible default.
warn	Logical. If TRUE, issue a warning if the minimum of the cross-validation criterion occurs at one of the ends of the search interval.

Details

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

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

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

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

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

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

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

Value

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

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[smooth.ppp](#)

Examples

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

Description

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

Usage

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

Arguments

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

Details

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

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

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

The coefficient 0.15 can be tweaked using the argument `co`.

Value

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

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

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

See Also

`pcf`, `bw.relrisk`

Examples

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

by.im

Apply Function to Image Broken Down by Factor

Description

Splits a pixel image into sub-images and applies a function to each sub-image.

Usage

```
## S3 method for class 'im'
by(data, INDICES, FUN, ...)
```

Arguments

- | | |
|----------------------|---|
| <code>data</code> | A pixel image (object of class "im"). |
| <code>INDICES</code> | Grouping variable. Either a tessellation (object of class "tess") or a factor-valued pixel image. |
| <code>FUN</code> | Function to be applied to each sub-image of <code>data</code> . |
| <code>...</code> | Extra arguments passed to <code>FUN</code> . |

Details

This is a method for the generic function `by` for pixel images (class "im").

The pixel image `data` is first divided into sub-images according to `INDICES`. Then the function `FUN` is applied to each subset. The results of each computation are returned in a list.

The grouping variable `INDICES` may be either

- a tessellation (object of class "tess"). Each tile of the tessellation delineates a subset of the spatial domain.
- a pixel image (object of class "im") with factor values. The levels of the factor determine subsets of the spatial domain.

Value

A list containing the results of each evaluation of `FUN`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`split.im`, `tess`, `im`

Examples

```
W <- square(1)
X <- as.im(function(x,y){sqrt(x^2+y^2)}, W)
Y <- dirichlet(runifpoint(12, W))
# mean pixel value in each subset
unlist(by(X, Y, mean))
# trimmed mean
unlist(by(X, Y, mean, trim=0.05))
```

by.hpp

Apply a Function to a Point Pattern Broken Down by Factor

Description

Splits a point pattern into sub-patterns, and applies the function to each sub-pattern.

Usage

```
## S3 method for class 'ppp'
by(data, INDICES=marks(data), FUN, ...)
```

Arguments

<code>data</code>	Point pattern (object of class "ppp").
<code>INDICES</code>	Grouping variable. Either a factor, a pixel image with factor values, or a tessellation.
<code>FUN</code>	Function to be applied to subsets of <code>data</code> .
<code>...</code>	Additional arguments to <code>FUN</code> .

Details

This is a method for the generic function `by` for point patterns (class "ppp").

The point pattern data is first divided into subsets according to INDICES. Then the function FUN is applied to each subset. The results of each computation are returned in a list.

The argument INDICES may be

- a factor, of length equal to the number of points in data. The levels of INDICES determine the destination of each point in data. The i th point of data will be placed in the sub-pattern `split.ppp(data)$l` where $l = f[i]$.
- a pixel image (object of class "im") with factor values. The pixel value of INDICES at each point of data will be used as the classifying variable.
- a tessellation (object of class "tess"). Each point of data will be classified according to the tile of the tessellation into which it falls.

If INDICES is missing, then data must be a multitype point pattern (a marked point pattern whose marks vector is a factor). Then the effect is that the points of each type are separated into different point patterns.

Value

A list (also of class "listof") containing the results returned from FUN for each of the subpatterns.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`ppp, split.ppp, cut.ppp, tess, im.`

Examples

```
# multitype point pattern, broken down by type
data(amacrine)
by(amacrine, FUN=density)
by(amacrine, FUN=function(x) { min(nndist(x)) } )

# how to pass additional arguments to FUN
by(amacrine, FUN=clarkevans, correction=c("Donnelly","cdf"))

# point pattern broken down by tessellation
data(swedishpines)
tes <- quadrats(swedishpines, 5, 5)
B <- by(swedishpines, tes, clarkevans, correction="Donnelly")
unlist(lapply(B, as.numeric))
```

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,eta2=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 Matern Cluster model which give the closest match between the theoretical K function of the Matern 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 (2011). 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 (2011).

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 $4 * \text{omega}^2$ where `omega` 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:

- | | |
|------------------|---|
| <code>par</code> | Vector of fitted parameter values. |
| <code>fit</code> | 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@csiro.au>
<http://www.maths.uwa.edu.au/~adrian/>

References

Jalilian, A., Guan, Y. and Waagepetersen, R. (2011) Decomposition of variance for spatial Cox processes. Manuscript submitted for publication.

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,eta2=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 Matern Cluster model which give the closest match between the theoretical pair correlation function of the Matern 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 (2011). 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 (2011).

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 $4 * \text{omega}^2$ where `omega` 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:

<code>par</code>	Vector of fitted parameter values.
<code>fit</code>	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@csiro.au>
<http://www.maths.uwa.edu.au/~adrian/>

References

- Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.
- Jalilian, A., Guan, Y. and Waagepetersen, R. (2011) Decomposition of variance for spatial Cox processes. Manuscript submitted for publication.

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")
```

cbind.hyperframe*Combine Hyperframes by Rows or by Columns***Description**

Methods for `cbind` and `rbind` for hyperframes.

Usage

```
## S3 method for class 'hyperframe'
cbind(...)
## S3 method for class 'hyperframe'
rbind(...)
```

Arguments

... Any number of hyperframes (objects of class `hyperframe`).

Details

These are methods for `cbind` and `rbind` for hyperframes.

Note that *all* the arguments must be hyperframes (because of the peculiar dispatch rules of `cbind` and `rbind`).

To combine a hyperframe with a data frame, one should either convert the data frame to a hyperframe using `as.hyperframe`, or explicitly invoke the function `cbind.hyperframe` or `rbind.hyperframe`.

In other words: if `h` is a hyperframe and `d` is a data frame, the result of `cbind(h, d)` will be the same as `cbind(as.data.frame(h), d)`, so that all hypercolumns of `h` will be deleted (and a warning will be issued). To combine `h` with `d` so that all columns of `h` are retained, type either `cbind(h, as.hyperframe(d))` or `cbind.hyperframe(h, d)`.

Value

Another hyperframe.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`hyperframe`, `as.hyperframe`

Examples

```
lambda <- runif(5, min=10, max=30)
X <- lapply(as.list(lambda), function(x) { rpoispp(x) })
h <- hyperframe(lambda=lambda, X=X)
g <- hyperframe(id=letters[1:5], Y=rev(X))
gh <- cbind(h, g)
hh <- rbind(h, h)
```

cells*Biological Cells Point Pattern*

Description

The data record the locations of the centres of 42 biological cells observed under optical microscopy in a histological section. The microscope field-of-view has been rescaled to the unit square.

The data were recorded by F.H.C. Crick and B.D. Ripley, and analysed in Ripley (1977, 1981) and Diggle (1983). They are often used as a canonical example of an ‘ordered’ point pattern.

Usage

```
data(cells)
```

Format

An object of class “`ppp`” representing the point pattern of cell centres. See [ppp.object](#) for details of the format.

Source

Crick and Ripley, see Ripley (1977)

References

- Diggle, P.J. (1983) *Statistical analysis of spatial point patterns*. Academic Press.
Ripley, B.D. (1977) Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, Series B* **39**, 172–212.
Ripley, B.D. (1981) *Spatial statistics*. John Wiley and Sons.

centroid.owin*Centroid of a window*

Description

Computes the centroid (centre of mass) of a window

Usage

```
centroid.owin(w)
```

Arguments

w A window

Details

The centroid of the window w is computed. The centroid (“centre of mass”) is the point whose x and y coordinates are the mean values of the x and y coordinates of all points in the window.

The argument w should be a window (an object of class “*owin*”, see [owin.object](#) for details) or can be given in any format acceptable to [as.owin\(\)](#).

The calculation uses an exact analytic formula for the case of polygonal windows.

Note that the centroid of a window is not necessarily inside the window. If the window is convex then it does contain its centroid.

Value

A list with components x , y giving the coordinates of the centroid of the window w .

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[owin](#), [as.owin](#)

Examples

```
w <- owin(c(0,1),c(0,1))
centroid.owin(w)
# returns 0.5, 0.5

data(demopat)
w <- demopat>window
# an irregular window
## Not run:
plot(w)
# plot the window
points(centroid.owin(w))
# mark its centroid

## End(Not run)

wapprox <- as.mask(w)
# pixel approximation of window
## Not run:
points(centroid.owin(wapprox))
# should be indistinguishable

## End(Not run)
```

chicago

Chicago Street Crime Data

Description

This dataset is a record of street crimes reported in the period 25 April to 8 May 2002, in an area of Chicago (Illinois, USA) close to the University of Chicago. The original street crime map was published in the Chicago Weekly News in 2002.

The data give the spatial location of each crime, and the type of crime. The type labels are interpreted as follows:

assault & battery/assault
burglary & burglary
cartheft & motor vehicle theft
damage & criminal damage
robbery & robbery
theft & theft
trespass & criminal trespass

All crimes occurred on or near a street. The data give the coordinates of all streets in the survey area, and their connectivity.

The dataset `chicago` is an object of class "lpp" representing a point pattern on a linear network. See [lpp](#) for further information on the format.

These data were published and analysed in Ang, Baddeley and Nair (2012).

Usage

```
data(chicago)
```

Format

Object of class "lpp". See [lpp](#).

Source

Chicago Weekly News, 2002. Manually digitised by Adrian Baddeley.

References

Ang, Q.W. (2010) *Statistical methodology for events on a network*. Master's thesis, School of Mathematics and Statistics, University of Western Australia.

Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. To appear in *Scandinavian Journal of Statistics*.

Chicago Weekly News website: www.chicagoweeklynews.com

Examples

```
data(chicago)
plot(chicago)
plot(as.linnet(chicago), main="Chicago Street Crimes", col="green")
plot(as.ppp(chicago), add=TRUE, col="red", chars=c(16,2,22,17,24,15,6))
```

chop.tess

Subdivide a Window or Tessellation using a Set of Lines

Description

Divide a given window into tiles delineated by a set of infinite straight lines, obtaining a tessellation of the window. Alternatively, given a tessellation, divide each tile of the tessellation into sub-tiles delineated by the lines.

Usage

```
chop.tess(X, L)
```

Arguments

- | | |
|---|---|
| X | A window (object of class "owin") or tessellation (object of class "tess") to be subdivided by lines. |
| L | A set of infinite straight lines (object of class "inflne") |

Details

The argument L should be a set of infinite straight lines in the plane (stored in an object L of class "inflne" created by the function [inflne](#)).

If X is a window, then it is divided into tiles delineated by the lines in L.

If X is a tessellation, then each tile of X is subdivided into sub-tiles delineated by the lines in L.

The result is a tessellation.

Value

A tessellation (object of class "tess").

Warning

If X is a non-convex window, or a tessellation containing non-convex tiles, then chop.tess(X,L) may contain a tile which consists of several unconnected pieces.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[inflne](#), [clip.inflne](#)

Examples

```
L <- infline(p=1:3, theta=pi/4)
W <- square(4)
chop.tess(W, L)
```

chorley

Chorley-Ribble Cancer Data

Description

Spatial locations of cases of cancer of the larynx and cancer of the lung, and the location of a disused industrial incinerator. A marked point pattern.

Usage

```
data(chorley)
```

Format

The dataset chorley is an object of class "ppp" representing a marked point pattern. Entries include

x	Cartesian x -coordinate of home address
y	Cartesian y -coordinate of home address
marks	factor with levels larynx and lung indicating whether this is a case of cancer of the larynx or cancer of the lung.

See [ppp.object](#) for details of the format.

The dataset chorley.extra is a list with two components. The first component `plot.it` is a function which will plot the data in a sensible fashion. The second component `incin` is a list with entries `x` and `y` giving the location of the industrial incinerator.

Coordinates are given in kilometres, and the resolution is 100 metres (0.1 km)

Notes

The data give the precise domicile addresses of new cases of cancer of the larynx (58 cases) and cancer of the lung (978 cases), recorded in the Chorley and South Ribble Health Authority of Lancashire (England) between 1974 and 1983. The supplementary data give the location of a disused industrial incinerator.

The data were first presented and analysed by Diggle (1990). They have subsequently been analysed by Diggle and Rowlingson (1994) and Baddeley et al. (2005).

The aim is to assess evidence for an increase in the incidence of cancer of the larynx in the vicinity of the now-disused industrial incinerator. The lung cancer cases serve as a surrogate for the spatially-varying density of the susceptible population.

The data are represented as a marked point pattern, with the points giving the spatial location of each individual's home address and the marks identifying whether each point is a case of laryngeal cancer or lung cancer.

Coordinates are in kilometres, and the resolution is 100 metres (0.1 km).

The dataset `chorley` has a polygonal window with 132 edges which closely approximates the boundary of the Chorley and South Ribble Health Authority.

Source

Coordinates of cases were provided by the Chorley and South Ribble Health Authority, and were kindly supplied by Professor Peter Diggle. Region boundary was digitised by Adrian Baddeley, 2005, from a photograph of an Ordnance Survey map.

References

- Baddeley, A., Turner, R., Moller, J. and Hazelton, M. (2005) Residual analysis for spatial point processes. *Journal of the Royal Statistical Society, Series B* **67**, 617–666.
- Diggle, P. (1990) A point process modelling approach to raised incidence of a rare phenomenon in the vicinity of a prespecified point. *Journal of the Royal Statistical Soc. Series A* **153**, 349–362.
- Diggle, P. and Rowlingson, B. (1994) A conditional approach to point process modelling of elevated risk. *Journal of the Royal Statistical Soc. Series A* **157**, 433–440.

Examples

```
data(chorley)
chorley.extra$plotit()
```

circumradius

Circumradius and Diameter of a Linear Network

Description

Compute the circumradius or diameter of a linear network measured using the shortest path distance.

Usage

```
circumradius(x)
## S3 method for class 'linnet'
diameter(x)
```

Arguments

x Linear network (object of class "linnet").

Details

The diameter of a linear network (in the shortest path distance) is the maximum value of the shortest-path distance between any two points u and v on the network.

The circumradius of a linear network (in the shortest path distance) is the minimum value, over all points u on the network, of the maximum shortest-path distance from u to another point v on the network.

The function `diameter` is generic; the function `diameter.linnet` is the method for objects of class `linnet`.

Value

A single numeric value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

See Also

[linnet](#)

Examples

```
data(simplicenet)
diameter(simplicenet)
circumradius(simplicenet)
```

clarkevans

Clark and Evans Aggregation Index

Description

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

Usage

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

Arguments

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

Details

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

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

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

"none": No edge correction is applied.

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

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

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

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

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

Value

A numeric value or numeric vector, with named components

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

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

Author(s)

John Rudge <rudge@esc.cam.ac.uk> with modifications by Adrian Baddeley <Adrian.Baddeley@csiro.au>
<http://www.maths.uwa.edu.au/~adrian/>

References

- Clark, P.J. and Evans, F.C. (1954) Distance to nearest neighbour as a measure of spatial relationships in populations *Ecology* **35**, 445–453.
 Donnelly, K. (1978) Simulations to determine the variance and edge-effect of total nearest neighbour distance. In *Simulation methods in archaeology*, Cambridge University Press, pp 91–95.

See Also

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

Examples

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

# Example of an ordered pattern
data(cells)
clarkevans(cells)
```

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

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

`clarkevans.test` *Clark and Evans Test*

Description

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

Usage

```
clarkevans.test(X, ...,
                 correction="none",
                 clipregion=NULL,
                 alternative=c("two.sided", "less", "greater"),
                 nsim=1000)
```

Arguments

<code>X</code>	A spatial point pattern (object of class "ppp").
<code>...</code>	Ignored.
<code>correction</code>	Character string. The type of edge correction to be applied. See clarkevans
<code>clipregion</code>	Clipping region for the guard area correction. A window (object of class "owin"). See clarkevans
<code>alternative</code>	String indicating the type of alternative for the hypothesis test.
<code>nsim</code>	Number of Monte Carlo simulations to perform, if a Monte Carlo p-value is required.

Details

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

The Clark-Evans index is computed by the function [clarkevans](#). See the help for [clarkevans](#) for information about the Clark-Evans index R and about the arguments `correction` and `clipregion`.

This command performs a hypothesis test of clustering or ordering of the point pattern `X`. The null hypothesis is Complete Spatial Randomness, i.e.\ a uniform Poisson process. The alternative hypothesis is specified by the argument `alternative`:

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

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

The Clark-Evans index R is computed for the data as described in [clarkevans](#).

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

For other edge corrections, the p -value for the test is computed by Monte Carlo simulation of `nsim` realisations of Complete Spatial Randomness.

Value

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

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

References

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

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

See Also

[clarkevans](#)

Examples

```
# Example of a clustered pattern
data(redwood)
clarkevans.test(redwood)
clarkevans.test(redwood, alternative="less")
```

clf.test

Cressie-Loosmore-Ford and Maximum Absolute Deviation Tests

Description

Perform the Cressie (1991)/Loosmore and Ford (2006) test or the Maximum Absolute Deviation test for a spatial point pattern.

Usage

```
clf.test(X, ..., rinterval = NULL, use.theo=FALSE)
mad.test(X, ..., rinterval = NULL, use.theo=FALSE)
```

Arguments

X	Either a point pattern (object of class "ppp", "lpp" or other class), a fitted point process model (object of class "ppm", "kppm" or other class) or an envelope object (class "envelope").
...	Arguments passed to envelope . Useful arguments include <code>fun</code> to determine the summary function, <code>nsim</code> to specify the number of Monte Carlo simulations, and <code>verbose=FALSE</code> to turn off the messages.
rinterval	Interval of values of the summary function argument <code>r</code> over which the maximum absolute deviation, or the integral, will be computed for the test. A numeric vector of length 2.
use.theo	Logical value determining whether to compare the summary function for the data to its theoretical value for CSR (<code>use.theo=TRUE</code>) or to the sample mean of simulations from CSR (<code>use.theo=FALSE</code>).

Details

These functions perform hypothesis tests for goodness-of-fit of a point pattern dataset to a point process model, based on Monte Carlo simulation from the model.

`clf.test` performs the test advocated by Loosmore and Ford (2006) which is also described in Cressie (1991, page 667, equation (8.5.42)).

`mad.test` performs the ‘global’ or ‘Maximum Absolute Deviation’ test described by Ripley (1977, 1981).

The type of test depends on the type of argument `X`.

- If `X` is some kind of point pattern, then a test of Complete Spatial Randomness (CSR) will be performed. That is, the null hypothesis is that the point pattern is completely random.
- If `X` is a fitted point process model, then a test of goodness-of-fit for the fitted model will be performed. The model object contains the data point pattern to which it was originally fitted. The null hypothesis is that the data point pattern is a realisation of the model.
- If `X` is an envelope object generated by [envelope](#), then it should have been generated with `savefuns=TRUE` or `savepatterns=TRUE` so that it contains simulation results. These simulations will be treated as realisations from the null hypothesis.

In all cases, the algorithm will first call [envelope](#) to generate or extract the simulated summary functions. The number of simulations that will be generated or extracted, is determined by the argument `nsim`, and defaults to 99. The summary function that will be computed is determined by the argument `fun` (or the first unnamed argument in the list ...) and defaults to [Kest](#) (except when `X` is an envelope object generated with `savefuns=TRUE`, when these functions will be taken).

The choice of summary function `fun` affects the power of the test. It is normally recommended to apply a variance-stabilising transformation (Ripley, 1981). If you are using the K function, the normal practice is to replace this by the L function (Besag, 1977) computed by [Lest](#). If you are using the F or G functions, the recommended practice is to apply Fisher’s variance-stabilising transformation $\sin^{-1} \sqrt{x}$ using the argument `transform`. See the Examples.

Value

An object of class “htest”. Printing this object gives a report on the result of the test. The p -value is contained in the component `p.value`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Andrew Hardegen.

References

- Besag, J. (1977) Discussion of Dr Ripley's paper. *Journal of the Royal Statistical Society, Series B*, **39**, 193–195.
- Cressie, N.A.C. (1991) *Statistics for spatial data*. John Wiley and Sons, 1991.
- Loosmore, N.B. and Ford, E.D. (2006) Statistical inference using the G or K point pattern spatial statistics. *Ecology* **87**, 1925–1931.
- Ripley, B.D. (1977) Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, Series B*, **39**, 172 – 212.
- Ripley, B.D. (1981) *Spatial statistics*. John Wiley and Sons.

See Also

[envelope](#)

Examples

```
clf.test(cells, Lest)
m <- mad.test(cells, Lest, verbose=FALSE, rinterval=c(0, 0.1))
m
# extract the p-value
m$p.value
# variance stabilised G function
clf.test(cells, Gest, transform=expression(asin(sqrt(.))), verbose=FALSE)
```

clickjoin

Interactively join vertices on a plot

Description

Given a point pattern representing a set of vertices, this command gives a point-and-click interface allowing the user to join pairs of selected vertices by edges.

Usage

```
clickjoin(x, ..., add = TRUE, m = NULL, join = TRUE)
```

Arguments

- | | |
|------|--|
| X | Point pattern of vertices. An object of class "ppp". |
| ... | Arguments passed to segments to control the plotting of the new edges. |
| add | Logical. Whether the point pattern X should be added to the existing plot (add=TRUE) or a new plot should be created (add=FALSE). |
| m | Optional. Logical matrix specifying an initial set of edges. There is an edge between vertices i and j if m[i,j] = TRUE. |
| join | Optional. If TRUE, then each user click will join a pair of vertices. If FALSE, then each user click will delete an existing edge. This is only relevant if m is supplied. |

Details

This function makes it easier for the user to create a linear network or a planar graph, given a set of vertices.

The function first displays the point pattern X , then repeatedly prompts the user to click on a pair of points in X . Each selected pair of points will be joined by an edge. The function returns a logical matrix which has entries equal to TRUE for each pair of vertices joined by an edge.

The selection of points is performed using [identify.ppp](#) which typically expects the user to click the left mouse button. This point-and-click interaction continues until the user terminates it, by pressing the middle mouse button, or pressing the right mouse button and selecting stop.

The return value can be used in [linnet](#) to create a linear network.

Value

Logical matrix m with value $m[i, j] = \text{TRUE}$ for every pair of vertices $X[i]$ and $X[j]$ that should be joined by an edge.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

See Also

[linnet](#), [clickppp](#)

clickpoly

Interactively Define a Polygon

Description

Allows the user to create a polygon by point-and-click in the display.

Usage

```
clickpoly(add=FALSE, nv=NULL, np=1, ...)
```

Arguments

add	Logical value indicating whether to create a new plot (add=FALSE) or draw over the existing plot (add=TRUE).
nv	Number of vertices of the polygon (if this is predetermined).
np	Number of polygons to create.
...	Arguments passed to locator to control the interactive plot.

Details

This function allows the user to create a polygonal window by interactively clicking on the screen display.

The user is prompted to point the mouse at any desired locations for the polygon vertices, and click the left mouse button to add each point. Interactive input stops after `nv` clicks (if `nv` was given) or when the middle mouse button is pressed.

The return value is a window (object of class "owin") representing the polygon.

This function uses the R command `locator` to input the mouse clicks. It only works on screen devices such as 'X11', 'windows' and 'quartz'. Arguments that can be passed to `locator` through ... include `pch` (plotting character), `cex` (character expansion factor) and `col` (colour). See `locator` and `par`.

Multiple polygons can also be drawn, by specifying `np > 1`. The polygons must be disjoint. The result is a single window object consisting of all the polygons.

Value

A window (object of class "owin") representing the polygon.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

`identify.ppp`, `clickppp`, `locator`

`clickppp`

Interactively Add Points

Description

Allows the user to create a point pattern by point-and-click in the display.

Usage

```
clickppp(n=NULL, win=square(1), types=NULL, ..., add=FALSE,
main=NULL, hook=NULL)
```

Arguments

<code>n</code>	Number of points to be added (if this is predetermined).
<code>win</code>	Window in which to create the point pattern. An object of class "owin".
<code>types</code>	Vector of types, when creating a multitype point pattern.
<code>...</code>	Optional extra arguments to be passed to <code>locator</code> to control the display.
<code>add</code>	Logical value indicating whether to create a new plot (<code>add=FALSE</code>) or draw over the existing plot (<code>add=TRUE</code>).
<code>main</code>	Main heading for plot.
<code>hook</code>	For internal use only. Do not use this argument.

Details

This function allows the user to create a point pattern by interactively clicking on the screen display.

First the window `win` is plotted on the current screen device. Then the user is prompted to point the mouse at any desired locations and click the left mouse button to add each point. Interactive input stops after `n` clicks (if `n` was given) or when the middle mouse button is pressed.

The return value is a point pattern containing the locations of all the clicked points inside the original window `win`, provided that all of the clicked locations were inside this window. Otherwise, the window is expanded to a box large enough to contain all the points (as well as containing the original window).

If the argument `types` is given, then a multitype point pattern will be created. The user is prompted to input the locations of points of type `type[i]`, for each successive index `i`. (If the argument `n` was given, there will be `n` points of *each* type.) The return value is a multitype point pattern.

This function uses the R command `locator` to input the mouse clicks. It only works on screen devices such as ‘X11’, ‘windows’ and ‘quartz’. Arguments that can be passed to `locator` through ... include `pch` (plotting character), `cex` (character expansion factor) and `col` (colour). See `locator` and `par`.

Value

A point pattern (object of class “`ppp`”).

Author(s)

Original by Dominic Schuhmacher. Adapted by Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[identify.ppp](#), [locator](#), [clickpoly](#)

`clip.inflne`

Intersect Infinite Straight Lines with a Window

Description

Take the intersection between a set of infinite straight lines and a window, yielding a set of line segments.

Usage

`clip.inflne(L, win)`

Arguments

- | | |
|------------------|---|
| <code>L</code> | Object of class “ <code>inflne</code> ” specifying a set of infinite straight lines in the plane. |
| <code>win</code> | Window (object of class “ <code>owin</code> ”). |

Details

This function computes the intersection between a set of infinite straight lines in the plane (stored in an object *L* of class "inflne" created by the function [inflne](#)) and a window *win*. The result is a pattern of line segments.

Value

A line segment pattern (object of class "psp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[inflne](#),[psp](#).

To divide a window into pieces using infinite lines, use [chop.tess](#).

Examples

```
L <- inflne(p=1:3, theta=pi/4)
W <- square(4)
clip.inflne(L, W)
```

closing

Morphological Closing

Description

Perform morphological closing of a window, a line segment pattern or a point pattern.

Usage

```
closing(w, r, ...)
## S3 method for class 'owin'
closing(w, r, ..., polygonal=NULL)
## S3 method for class 'ppp'
closing(w, r, ..., polygonal=TRUE)
## S3 method for class 'psp'
closing(w, r, ..., polygonal=TRUE)
```

Arguments

<i>w</i>	A window (object of class "owin" or a line segment pattern (object of class "psp") or a point pattern (object of class "ppp").
<i>r</i>	positive number: the radius of the closing.
...	extra arguments passed to as.mask controlling the pixel resolution, if a pixel approximation is used
<i>polygonal</i>	Logical flag indicating whether to compute a polygonal approximation to the erosion (<i>polygonal</i> =TRUE) or a pixel grid approximation (<i>polygonal</i> =FALSE).

Details

The morphological closing (Serra, 1982) of a set W by a distance $r > 0$ is the set of all points that cannot be separated from W by any circle of radius r . That is, a point x belongs to the closing W^* if it is impossible to draw any circle of radius r that has x on the inside and W on the outside. The closing W^* contains the original set W .

For a small radius r , the closing operation has the effect of smoothing out irregularities in the boundary of W . For larger radii, the closing operation smooths out concave features in the boundary. For very large radii, the closed set W^* becomes more and more convex.

The algorithm applies [dilation](#) followed by [erosion](#).

Value

If $r > 0$, an object of class "owin" representing the closed region. If $r=0$, the result is identical to w .

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

Serra, J. (1982) Image analysis and mathematical morphology. Academic Press.

See Also

[opening](#) for the opposite operation.
[dilation](#), [erosion](#) for the basic operations.
[owin](#), [as.owin](#) for information about windows.

Examples

```
data(letterR)
v <- closing(letterR, 0.25, dimyx=256)
plot(v, main="closing")
plot(letterR, add=TRUE)
```

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[print.ppm](#), [ppm.object](#), [ppm](#)

Examples

```
data(cells)

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

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

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

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

Description

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

Usage

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

Arguments

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

Details

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

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

Value

Numeric vector of coefficients.

Author(s)

Adrian Baddeley <adrian@maths.uwa.edu.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

`slrm`

Examples

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

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

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

Usage

```
collapse.fv(..., same = NULL, different = NULL)
```

Arguments

...	Arguments which are objects of class "fv", or a list of objects of class "fv".
same	Character string or character vector specifying a column or columns, present in each "fv" object, that are identical in each object. This column or columns will be included only once.
different	Character string or character vector specifying a column or columns, present in each "fv" object, that contain different values in each object. Each of these columns of data will be included, with labels that distinguish them from each other.

Details

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

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

The arguments ... should be function tables (objects of class "fv", see [fv.object](#)) that are compatible in the sense that they have the same values of the function argument.

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

The argument `different` identifies any columns that are present in each function table, and which contain different numerical values in each table. Each of these columns will be included, with labels to distinguish them.

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

Value

Object of class "fv".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[fv.object](#), [cbind.fv](#)

Examples

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

colourmap

*Colour Lookup Tables***Description**

Create a colour map (colour lookup table).

Usage

```
colourmap(col, ..., range=NULL, breaks=NULL, inputs=NULL)
```

Arguments

col	Vector of values specifying colours
...	Ignored.
range	Interval to be mapped. A numeric vector of length 2, specifying the endpoints of the range of values to be mapped. Incompatible with breaks or inputs.
inputs	Values to which the colours are associated. A factor or vector of the same length as col. Incompatible with breaks or range.
breaks	Breakpoints for the colour map. A numeric vector of length equal to length(col)+1. Incompatible with range or inputs.

Details

A colour map is a mechanism for associating colours with data. It can be regarded as a function, mapping data to colours.

The command `colourmap` creates an object representing a colour map, which can then be used to control the plot commands in the `spatstat` package. It can also be used to compute the colour assigned to any data value.

The argument `col` specifies the colours to which data values will be mapped. It should be a vector whose entries can be interpreted as colours by the standard R graphics system. The entries can be string names of colours like "red", or integers that refer to colours in the standard palette, or strings containing six-letter hexadecimal codes like "#F0A0FF".

Exactly one of the arguments `range`, `inputs` or `breaks` must be specified by name.

If `inputs` is given, then it should be a vector or factor, of the same length as `col`. The entries of `inputs` can be any atomic type (e.g. numeric, logical, character, complex) or factor values. The resulting colour map associates the value `inputs[i]` with the colour `col[i]`.

If `range` is given, then it determines the interval of the real number line that will be mapped. It should be a numeric vector of length 2.

If `breaks` is given, then it determines the precise intervals of the real number line which are mapped to each colour. It should be a numeric vector, of length at least 2, with entries that are in increasing order. Infinite values are allowed. Any number in the range between `breaks[i]` and `breaks[i+1]` will be mapped to the colour `col[i]`.

The result is an object of class "colourmap". There are `print` and `plot` methods for this class. Some plot commands in the **spatstat** package accept an object of this class as a specification of the colour map.

The result is also a function `f` which can be used to compute the colour assigned to any data value. That is, `f(x)` returns the character value of the colour assigned to `x`. This also works for vectors of data values.

Value

A function, which is also an object of class "colourmap".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

The plot method `plot.colourmap`.

See the R help file on `colours` for information about the colours that R recognises, and how to manipulate them.

See `colourtools` for more tools to manipulate colours.

See `lut` for lookup tables.

Examples

```
# colour map for real numbers, using breakpoints
cr <- colourmap(c("red", "blue", "green"), breaks=c(0,5,10,15))
cr
cr(3.2)
cr(c(3,5,7))
# a large colour map
co <- colourmap(rainbow(100), range=c(-1,1))
co(0.2)
# colour map for discrete set of values
ct <- colourmap(c("red", "green"), inputs=c(FALSE, TRUE))
ct(TRUE)
```

Description

These functions convert between different formats for specifying a colour in R, and determine whether colours are equivalent.

Usage

```
col2hex(x)
rgb2hex(v)
paletteindex(x)
samecolour(x,y)
```

Arguments

x, y	Any valid specification for a colour or sequence of colours accepted by col2rgb .
v	A numeric vector of length 3, giving the RGB values (0 to 255) of a single colour, or a 3-column matrix giving the RGB values of several colours.

Details

`col2hex` converts colours specified in any format into their hexadecimal character codes.

`rgb2hex` converts RGB colour values into their hexadecimal character codes.

`paletteindex` checks whether the colour or colours specified by `x` are available in the default palette returned by [palette\(\)](#). If so, it returns the index or indices of the colours in the palette. If not, it returns NA.

`samecolour` decides whether two colours `x` and `y` are equivalent.

Value

For `col2hex` and `rgb2hex`, a character vector containing hexadecimal colour codes.

For `paletteindex`, an integer vector, possibly containing NA values.

For `samecolour`, a logical value or logical vector.

Warning

`paletteindex("green")` returns NA because the green colour in the default palette is called "green3".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[col2rgb](#), [palette](#)

Examples

```
samecolour("grey", "gray")
paletteindex("grey")
```

commonGrid

Determine A Common Spatial Domain And Pixel Resolution

Description

Determine a common spatial domain and pixel resolution for several spatial objects such as images, masks, windows and point patterns.

Usage

```
commonGrid(...)
```

Arguments

... Any number of pixel images (objects of class "im"), binary masks (objects of class "owin" of type "mask") or data which can be converted to binary masks by [as.mask](#).

Details

This function determines a common spatial resolution and spatial domain for several spatial objects.

The arguments ... may be pixel images, binary masks, or other spatial objects acceptable to [as.mask](#).

The common pixel grid is determined by inspecting all the pixel images and binary masks in the argument list, finding the pixel grid with the highest spatial resolution, and extending this pixel grid to cover the bounding box of all the spatial objects.

The return value is a binary mask M, representing the bounding box at the chosen pixel resolution. Use [as.im\(X, W=M\)](#) to convert a pixel image X to this new pixel resolution. Use [as.mask\(W, xy=M\)](#) to convert a window W to a binary mask at this new pixel resolution. See the Examples.

Value

A binary mask (object of class "owin" and type "mask").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[harmonise.im](#), [compatible.im](#), [as.im](#)

Examples

```
A <- setcov(square(1))
G <- density(runifpoint(42), dimyx=16)
H <- commonGrid(A, letterR, G)
newR <- as.mask(letterR, xy=H)
newG <- as.im(G, W=H)
```

compareFit

Residual Diagnostics for Multiple Fitted Models

Description

C.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

<code>object</code>	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.
<code>Fun</code>	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.
<code>r</code>	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.
<code>breaks</code>	Optional alternative to <code>r</code> for advanced use.
<code>...</code>	Extra arguments passed to <code>Fun</code> .
<code>trend, interaction, rbord</code>	Optional. Arguments passed to <code>ppm</code> to fit a point process model to the data, if <code>object</code> is a point pattern or list of point patterns. See <code>ppm</code> for details. Each of these arguments can be a list, specifying different <code>trend</code> , <code>interaction</code> and/or <code>rbord</code> values to be used to generate different fitted models.
<code>modelnames</code>	Character vector. Short descriptive names for the different models.
<code>same, different</code>	Character strings or character vectors passed to <code>collapse.fv</code> 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, Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Jesper Moller.

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,
                 same="iso", different="icom", modelnames=iname, nd=nd)
K
```

compatible

Test Whether Objects Are Compatible

Description

Tests whether two or more objects of the same class are compatible.

Usage

```
compatible(A, B, ...)
```

Arguments

A, B, ...	Two or more objects of the same class
-----------	---------------------------------------

Details

This generic function is used to check whether the objects A and B (and any additional objects ...) are compatible.

What is meant by ‘compatible’ depends on the class of object.

There are methods for the classes “fv”, “fasp”, “im” and “units”.

Value

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

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[compatible.fv](#), [compatible.fasp](#), [compatible.im](#), [compatible.units](#)

compatible.fasp

Test Whether Function Arrays Are Compatible

Description

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

Usage

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

Arguments

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

Details

An object of class “fasp” can be regarded as an array of functions. Such objects are returned by the command [alltypes](#).

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

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

Value

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

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[eval.fasp](#)

compatible.fv

Test Whether Function Objects Are Compatible

Description

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

Usage

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

Arguments

A, B, ... Two or more function value objects (class "fv").

Details

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

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

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

Value

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

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[eval.fv](#)

compatible.im

*Test Whether Pixel Images Are Compatible***Description**

Tests whether two or more pixel image objects have compatible dimensions.

Usage

```
## S3 method for class 'im'
compatible(A, B, ..., tol=1e-6)
```

Arguments

A, B, ...	Two or more pixel images (objects of class "im").
tol	Tolerance factor

Details

This function tests whether the pixel images A and B (and any additional images ...) have compatible pixel dimensions. They are compatible if they have the same number of rows and columns, the same physical pixel dimensions, and occupy the same rectangle in the plane.

The argument `tol` specifies the maximum tolerated error in the pixel coordinates, expressed as a fraction of the dimensions of a single pixel.

Value

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

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[eval.im](#), [harmonise.im](#), [commonGrid](#)

complement.owin

*Take Complement of a Window***Description**

Take the set complement of a window, within its enclosing rectangle or in a larger rectangle.

Usage

```
complement.owin(w, frame=as.rectangle(w))
```

Arguments

w	an object of class "owin" describing a window of observation for a point pattern.
frame	Optional. The enclosing rectangle, with respect to which the set complement is taken.

Details

This yields a window object (of class "owin", see [owin.object](#)) representing the set complement of w with respect to the rectangle frame.

By default, frame is the enclosing box of w (originally specified by the arguments xrange and yrange given to [owin](#) when w was created). If frame is specified, it must be a rectangle (an object of class "owin" whose type is "rectangle") and it must be larger than the enclosing box of w. This rectangle becomes the enclosing box for the resulting window.

If w is a rectangle, then frame must be specified. Otherwise an error will occur (since the complement of w in itself is empty).

For rectangular and polygonal windows, the complement is computed by reversing the sign of each boundary polygon, while for binary masks it is computed by negating the pixel values.

Value

Another object of class "owin" representing the complement of the window, i.e. the inside of the window becomes the outside.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[owin](#), [owin.object](#)

Examples

```
# rectangular
a <- owin(c(0,1),c(0,1))
b <- owin(c(-1,2),c(-1,2))
bmina <- complement.owin(a, frame=b)
# polygonal
data(demopat)
w <- demopat>window
outside <- complement.owin(w)
# mask
w <- as.mask(demopat>window)
outside <- complement.owin(w)
```

concatxy*Concatenate x,y Coordinate Vectors*

Description

Concatenate any number of pairs of x and y coordinate vectors.

Usage

```
concatxy(...)
```

Arguments

... Any number of arguments, each of which is a structure containing elements x and y.

Details

This function can be used to superimpose two or more point patterns of unmarked points (but see also [superimpose](#) which is recommended).

It assumes that each of the arguments in ... is a structure containing (at least) the elements x and y. It concatenates all the x elements into a vector x, and similarly for y, and returns these concatenated vectors.

Value

A list with two components x and y, which are the concatenations of all the corresponding x and y vectors in the argument list.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[superimpose](#), [quadscheme](#)

Examples

```
dat <- runifrect(30)
xy <- list(x=runif(10),y=runif(10))
new <- concatxy(dat, xy)
```

connected

*Connected components of an image or window***Description**

Finds the topologically-connected clumps of pixels in an image or window.

Usage

```
connected(X, background = NA, method="C")
```

Arguments

X	Image (object of class "im") or window (object of class "owin").
background	Optional. Treat pixels with this value as being part of the background.
method	String indicating the algorithm to be used. Either "C" or "interpreted". See Details.

Details

This function computes the connected component transform (Rosenfeld and Pfalz, 1966) of a binary image or binary mask. The argument X is first converted into a pixel image with logical values. Then the algorithm identifies the connected components (topologically-connected clumps of pixels) in the foreground.

Two pixels belong to the same connected component if they have the value TRUE and if they are neighbours (in the 8-connected sense). This rule is applied repeatedly until it terminates. Then each connected component contains all the pixels that can be reached by stepping from neighbour to neighbour.

If `method="C"`, the computation is performed by a compiled C language implementation of the classical algorithm of Rosenfeld and Pfalz (1966). If `method="interpreted"`, the computation is performed by an R implementation of the algorithm of Park et al (2000).

The result is a factor-valued image, with levels that correspond to the connected components. The Examples show how to extract each connected component as a separate window object.

Value

A pixel image (object of class "im") with factor values. The levels of the factor correspond to the connected components.

Warnings

It may be hard to distinguish different components in the default plot because the colours of nearby components may be very similar. See the Examples for a randomised colour map.

The algorithm for `method="interpreted"` can be very slow for large images (or images where the connected components include a large number of pixels).

Author(s)

Original R code by Julian Burgos, University of Washington. Adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Park, J.-M., Looney, C.G. and Chen, H.-C. (2000) Fast connected component labeling algorithm using a divide and conquer technique. Pages 373-376 in S.Y. Shin (ed) *Computers and Their Applications: Proceedings of the ISCA 15th International Conference on Computers and Their Applications*, March 29-31, 2000, New Orleans, Louisiana USA. ISCA 2000, ISBN 1-880843-32-3.
- Rosenfeld, A. and Pfalz, J.L. (1966) Sequential operations in digital processing. *Journal of the Association for Computing Machinery* **13** 471-494.

See Also

[im.object](#), [tess](#)

Examples

```
data(cells)
d <- distmap(cells, dimyx=256)
X <- levelset(d, 0.06)
plot(X)
Z <- connected(X)
plot(Z)

# number of components
nc <- length(levels(Z))
# plot with randomised colour map
plot(Z, col=hsv(h=sample(seq(0,1,length=nc), nc)))

# how to extract the components as a list of windows
W <- tiles(tess(image=Z))
```

contour.im

Contour plot of pixel image

Description

Generates a contour plot of a pixel image.

Usage

```
## S3 method for class 'im'
contour(x, ..., main, axes=TRUE, add=FALSE)
```

Arguments

- | | |
|------|--|
| x | Pixel image to be plotted. An object of class "im". |
| main | Character string to be displayed as the main title. |
| axes | Logical. If TRUE, coordinate axes are plotted (with tick marks) around a region slightly larger than the image window. If FALSE, no axes are plotted, and a box is drawn tightly around the image window. Ignored if add=TRUE. |
| add | Logical. If FALSE, a new plot is created. If TRUE, the contours are drawn over the existing plot. |
| ... | Other arguments passed to contour.default controlling the contour plot; see Details. |

Details

This is a method for the generic `contour` function, for objects of the class "im".

An object of class "im" represents a pixel image; see [im.object](#).

This function displays the values of the pixel image `x` as a contour plot on the current plot device, using equal scales on the x and y axes.

The appearance of the plot can be modified using any of the arguments listed in the help for [contour.default](#). Useful ones include:

nlevels Number of contour levels to plot.

drawlabels Whether to label the contour lines with text.

col, lty, lwd Colour, type, and width of contour lines.

See [contour.default](#) for a full list of these arguments.

The defaults for any of the abovementioned arguments can be reset using [spatstat.options](#).

Value

none.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[im.object](#), [plot.im](#), [persp.im](#)

Examples

```
# an image
Z <- setcov(owin())
contour(Z)
contour(Z, axes=FALSE)
```

contour.listof

Plot a List of Things

Description

Plots a list of things, either as an array of contour plots, or as an array of images.

Usage

```
## S3 method for class 'listof'
contour(x, ...)
## S3 method for class 'listof'
image(x, ..., equal.ribbon=FALSE)
```

Arguments

- `x` An object of the class "listof". Essentially a list of objects.
`...` Arguments passed to `plot.listof` to control the plot.
`equal.ribbon` Logical. If TRUE, the colour maps of all the images are the same. If FALSE, the colour map of each image is adjusted to the range of values of that image.

Details

These are methods for the generic plot commands `contour` and `image`, for the class "listof". The commands will display each object in the list `x`, either as a contour plot (`contour.listof`) or a pixel image plot (`image.listof`), with the displays laid out in a grid. See `plot.listof` for more information.

Value

Null.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`plot.listof`

Examples

```
# Multitype point pattern
contour(D <- density(split(amacrine)))
image(D, equal.ribbon=TRUE, main="")
```

convexhull

Convex Hull

Description

Computes the convex hull of a spatial object.

Usage

```
convexhull(x)
```

Arguments

- `x` a window (object of class "owin"), a point pattern (object of class "ppp"), a line segment pattern (object of class "psp"), or an object that can be converted to a window by `as.owin`.

Details

This function computes the convex hull of the spatial object `x`.

Value

A window (an object of class "owin").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[owin](#), [convexhull.xy](#), [is.convex](#)

Examples

```
data(demopat)
W <- demopat>window
plot(convexhull(W), col="lightblue", border=NA)
plot(W, add=TRUE, lwd=2)
```

convexhull.xy

Convex Hull of Points

Description

Computes the convex hull of a set of points in two dimensions.

Usage

```
convexhull.xy(x, y=NULL)
```

Arguments

- | | |
|---|--|
| x | vector of x coordinates of observed points, or a 2-column matrix giving x,y coordinates, or a list with components x,y giving coordinates (such as a point pattern object of class "ppp".) |
| y | (optional) vector of y coordinates of observed points, if x is a vector. |

Details

Given an observed pattern of points with coordinates given by x and y, this function computes the convex hull of the points, and returns it as a window.

Value

A window (an object of class "owin").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[owin](#), [as.owin](#), [convexhull](#), [bounding.box.xy](#), [ripras](#)

Examples

```
x <- runif(30)
y <- runif(30)
w <- convexhull.xy(x,y)
plot(owin(), main="convexhull.xy(x,y)", lty=2)
plot(w, add=TRUE)
points(x,y)

X <- rpoispp(30)
plot(X, main="convexhull.xy(X)")
plot(convexhull.xy(X), add=TRUE)
```

convolve.im

*Convolution of Pixel Images***Description**

Computes the convolution of two pixel images.

Usage

```
convolve.im(X, Y=X, ..., reflectX=FALSE, reflectY=FALSE)
```

Arguments

- X A pixel image (object of class "im").
- Y Optional. Another pixel image.
- ... Ignored.
- reflectX, reflectY Logical values specifying whether the images X and Y (respectively) should be reflected in the origin before computing the convolution.

Details

The *convolution* of two pixel images X and Y in the plane is the function $C(v)$ defined for each vector v as

$$C(v) = \int X(u)Y(v-u) du$$

where the integral is over all spatial locations u , and where $X(u)$ and $Y(u)$ denote the pixel values of X and Y respectively at location u .

This command computes a discretised approximation to the convolution, using the Fast Fourier Transform. The return value is another pixel image (object of class "im") whose greyscale values are values of the convolution.

If `reflectX = TRUE` then the pixel image X is reflected in the origin (see [reflect](#)) before the convolution is computed, so that `convolve.im(X, Y, reflectX=TRUE)` is mathematically equivalent to `convolve.im(reflect(X), Y)`. (These two commands are not exactly equivalent, because the

reflection is performed in the Fourier domain in the first command, and reflection is performed in the spatial domain in the second command).

Similarly if `reflectY = TRUE` then the pixel image `Y` is reflected in the origin before the convolution is computed, so that `convolve.im(X, Y, reflectY=TRUE)` is mathematically equivalent to `convolve.im(X, reflect(Y))`.

Value

A pixel image (an object of class "im") representing the convolution of `X` and `Y`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[imcov](#), [reflect](#)

Examples

```
X <- as.im(letterR)
Y <- as.im(square(1))
plot(convolve.im(X, Y))
plot(convolve.im(X, Y, reflectX=TRUE))
plot(convolve.im(X))
```

coords

Extract or Change Coordinates of a Spatial or Spatiotemporal Point Pattern

Description

Given any kind of spatial or space-time point pattern, this function extracts the (space and/or time and/or local) coordinates of the points and returns them as a data frame.

Usage

```
coords(x, ...)
## S3 method for class 'ppp'
coords(x, ...)
## S3 method for class 'ppx'
coords(x, ..., spatial = TRUE, temporal = TRUE, local=TRUE)
coords(x, ...) <- value
## S3 replacement method for class 'ppp'
coords(x, ...) <- value
## S3 replacement method for class 'ppx'
coords(x, ..., spatial = TRUE, temporal = TRUE, local=TRUE) <- value
```

Arguments

- `x` A point pattern: either a two-dimensional point pattern (object of class "ppp"), a three-dimensional point pattern (object of class "pp3"), or a general multidimensional space-time point pattern (object of class "ppx").
- `...` Further arguments passed to methods.
- `spatial, temporal, local` Logical values indicating whether to extract spatial, temporal and local coordinates, respectively. The default is to return all such coordinates. (Only relevant to ppx objects).
- `value` New values of the coordinates. A numeric vector with one entry for each point in `x`, or a numeric matrix or data frame with one row for each point in `x`.

Details

The function `coords` extracts the coordinates from a point pattern. The function `coords<-` replaces the coordinates of the point pattern with new values.

Both functions `coords` and `coords<-` are generic, with methods for the classes "ppp") and "ppx". An object of class "pp3" also inherits from "ppx" and is handled by the method for "ppx".

Value

`coords` returns a `data.frame` with one row for each point, containing the coordinates. `coords<-` returns the altered point pattern.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppx](#), [pp3](#), [ppp](#), [as.hyperframe.ppx](#), [as.data.frame.ppx](#).

Examples

```
df <- data.frame(x=runif(4),y=runif(4),t=runif(4))
X <- ppx(data=df, coord.type=c("s","s","t"))
coords(X)
coords(X, temporal=FALSE)
coords(X) <- matrix(runif(12), ncol=3)
```

Description

These data come from an intensive geological survey of a 70 x 158 km region in central Queensland, Australia. They consist of 67 points representing copper ore deposits, and 146 line segments representing geological ‘lineaments’. Lineaments are linear features, visible on a satellite image, that are believed to consist largely of geological faults (Berman, 1986, p. 55). It would be of great interest to predict the occurrence of copper deposits from the lineament pattern, since the latter can easily be observed on satellite images.

These data were introduced and analysed by Berman (1986). They have also been studied by Berman and Diggle (1989), Berman and Turner (1992), Baddeley and Turner (2000, 2005), Foxall and Baddeley (2002) and Baddeley et al (2005).

Many analyses have been performed on the southern half of the data only. This subset is also provided.

Usage

```
data(copper)
```

Format

`copper` is a list with the following entries:

Points a point pattern (object of class "ppp") representing the full point pattern of copper deposits.
See [ppp.object](#) for details of the format.

Lines a line segment pattern (object of class "psp") representing the lineaments in the full dataset.
See [psp.object](#) for details of the format.

SouthWindow the window delineating the southern half of the study region. An object of class "owin".

SouthPoints the point pattern of copper deposits in the southern half of the study region. An object of class "ppp".

SouthLines the line segment pattern of the lineaments in the southern half of the study region. An object of class "psp".

Source

Dr J. Huntington. Coordinates kindly provided by Dr. Mark Berman and Dr. A. Green, CSIRO, Sydney, Australia.

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., Turner, R., Moller, J. and Hazelton, M. (2005) Residual analysis for spatial point processes. *Journal of the Royal Statistical Society, Series B* **67**, 617–666.
- 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.
- Berman, M. (1986). Testing for spatial association between a point process and another stochastic process. *Applied Statistics* **35**, 54–62.

- Berman, M. and Diggle, P.J. (1989) Estimating Weighted Integrals of the Second-order Intensity of a Spatial Point Process. *Journal of the Royal Statistical Society, series B* **51**, 81–92.
- Berman, M. and Turner, T.R. (1992) Approximating point process likelihoods with GLIM. *Applied Statistics* **41**, 31–38.
- Foxall, R. and Baddeley, A. (2002) Nonparametric measures of association between a spatial point process and a random set, with geological applications. *Applied Statistics* **51**, 165–182.

Examples

```
data(copper)

# Plot full dataset

plot(copper$Points)
plot(copper$Lines, add=TRUE)

# Plot southern half of data
plot(copper$SouthPoints)
plot(copper$SouthLines, add=TRUE)

## Not run:
Z <- distmap(copper$SouthLines)
plot(Z)
X <- copper$SouthPoints
ppm(X, ~D, covariates=list(D=Z))

## End(Not run)
```

corners

Corners of a rectangle

Description

Returns the four corners of a rectangle

Usage

```
corners(window)
```

Arguments

window	A window. An object of class <code>owin</code> , or data in any format acceptable to <code>as.owin()</code> .
--------	---

Details

This trivial function is occasionally convenient. If `window` is of type "rectangle" this returns the four corners of the window itself; otherwise, it returns the corners of the bounding rectangle of the window.

Value

A list with two components `x` and `y`, which are numeric vectors of length 4 giving the coordinates of the four corner points of the (bounding rectangle of the) window.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[quad.object](#), [quadscheme](#)

Examples

```
w <- unit.square()
corners(w)
# returns list(x=c(0,1,0,1),y=c(0,0,1,1))
```

crossdist

Pairwise distances

Description

Computes the distances between pairs of ‘things’ taken from two different datasets.

Usage

```
crossdist(X, Y, ...)
```

Arguments

- | | |
|------|---|
| X, Y | Two objects of the same class. |
| ... | Additional arguments depending on the method. |

Details

Given two datasets X and Y (representing either two point patterns or two line segment patterns) `crossdist` computes the Euclidean distance from each thing in the first dataset to each thing in the second dataset, and returns a matrix containing these distances.

The function `crossdist` is generic, with methods for point patterns (objects of class “`ppp`”), line segment patterns (objects of class “`psp`”), and a default method. See the documentation for [crossdist.ppp](#), [crossdist.psp](#) or [crossdist.default](#) for further details.

Value

A matrix whose [i,j] entry is the distance from the i-th thing in the first dataset to the j-th thing in the second dataset.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

See Also

[crossdist.ppp](#), [crossdist.psp](#), [crossdist.default](#), [pairdist](#), [nndist](#)

<code>crossdist.default</code>	<i>Pairwise distances between two different sets of points</i>
--------------------------------	--

Description

Computes the distances between each pair of points taken from two different sets of points.

Usage

```
## Default S3 method:  
crossdist(X, Y, x2, y2, ..., period=NULL, method="C")
```

Arguments

<code>X, Y</code>	Numeric vectors of equal length specifying the coordinates of the first set of points.
<code>x2,y2</code>	Numeric vectors of equal length specifying the coordinates of the second set of points.
<code>...</code>	Ignored.
<code>period</code>	Optional. Dimensions for periodic edge correction.
<code>method</code>	String specifying which method of calculation to use. Values are "C" and "interpreted".

Details

Given two sets of points, this function computes the Euclidean distance from each point in the first set to each point in the second set, and returns a matrix containing these distances.

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

This function expects `X` and `Y` to be numeric vectors of equal length specifying the coordinates of the first set of points. The arguments `x2,y2` specify the coordinates of the second set of points.

Alternatively if `period` is given, then the distances will be computed in the 'periodic' sense (also known as 'torus' distance). The points will be treated as if they are in a rectangle of width `period[1]` and height `period[2]`. Opposite edges of the rectangle are regarded as equivalent.

The argument `method` is not normally used. It is retained only for checking the validity of the software. If `method = "interpreted"` then the distances are computed using interpreted R code only. If `method="C"` (the default) then C code is used. The C code is faster by a factor of 4.

Value

A matrix whose $[i, j]$ entry is the distance from the i -th point in the first set of points to the j -th point in the second set of points.

Author(s)

Pavel Grabarnik <pavel.grabar@issp.serpukhov.su> and Adrian Baddeley <Adrian.Baddeley@csiro.au>
<http://www.maths.uwa.edu.au/~adrian/>

See Also

`crossdist`, `crossdist.ppp`, `crossdist.psp`, `pairdist`, `nndist`, `Gest`

Examples

```
d <- crossdist(runif(7), runif(7), runif(12), runif(12))
d <- crossdist(runif(7), runif(7), runif(12), runif(12), period=c(1,1))
```

crossdist.pp3

Pairwise distances between two different three-dimensional point patterns

Description

Computes the distances between pairs of points taken from two different three-dimensional point patterns.

Usage

```
## S3 method for class 'pp3'
crossdist(X, Y, ..., periodic=FALSE)
```

Arguments

- | | |
|----------|---|
| X, Y | Point patterns in three dimensions (objects of class "pp3"). |
| ... | Ignored. |
| periodic | Logical. Specifies whether to apply a periodic edge correction. |

Details

Given two point patterns in three-dimensional space, this function computes the Euclidean distance from each point in the first pattern to each point in the second pattern, and returns a matrix containing these distances.

This is a method for the generic function `crossdist` for three-dimensional point patterns (objects of class "pp3").

This function expects two point patterns `X` and `Y`, and returns the matrix whose $[i,j]$ entry is the distance from `X[i]` to `Y[j]`.

Alternatively if `periodic=TRUE`, then provided the windows containing `X` and `Y` are identical and are rectangular, then the distances will be computed in the 'periodic' sense (also known as 'torus' distance): opposite edges of the rectangle are regarded as equivalent. This is meaningless if the window is not a rectangle.

Value

A matrix whose $[i,j]$ entry is the distance from the i -th point in `X` to the j -th point in `Y`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> based on code for two dimensions by Pavel Grabarnik.

See Also

`crossdist`, `pairdist`, `nndist`, `G3est`

Examples

```
X <- runifpoint3(20)
Y <- runifpoint3(30)
d <- crossdist(X, Y)
d <- crossdist(X, Y, periodic=TRUE)
```

crossdist.ppp

Pairwise distances between two different point patterns

Description

Computes the distances between pairs of points taken from two different point patterns.

Usage

```
## S3 method for class 'ppp'
crossdist(X, Y, ..., periodic=FALSE, method="C")
```

Arguments

X, Y	Point patterns (objects of class "ppp").
...	Ignored.
periodic	Logical. Specifies whether to apply a periodic edge correction.
method	String specifying which method of calculation to use. Values are "C" and "interpreted".

Details

Given two point patterns, this function computes the Euclidean distance from each point in the first pattern to each point in the second pattern, and returns a matrix containing these distances.

This is a method for the generic function `crossdist` for point patterns (objects of class "ppp").

This function expects two point patterns X and Y, and returns the matrix whose [i,j] entry is the distance from X[i] to Y[j].

Alternatively if `periodic=TRUE`, then provided the windows containing X and Y are identical and are rectangular, then the distances will be computed in the 'periodic' sense (also known as 'torus' distance): opposite edges of the rectangle are regarded as equivalent. This is meaningless if the window is not a rectangle.

The argument `method` is not normally used. It is retained only for checking the validity of the software. If `method = "interpreted"` then the distances are computed using interpreted R code only. If `method="C"` (the default) then C code is used. The C code is faster by a factor of 4.

Value

A matrix whose [i,j] entry is the distance from the i-th point in X to the j-th point in Y.

Author(s)

Pavel Grabarnik <pavel.grabar@issp.serpukhov.su> and Adrian Baddeley <Adrian.Baddeley@csiro.au>
<http://www.maths.uwa.edu.au/~adrian/>

See Also

[crossdist](#), [crossdist.default](#), [crossdist.psp](#), [pairdist](#), [nndist](#), [Gest](#)

Examples

```
data(cells)
d <- crossdist(cells, runifpoint(6))
d <- crossdist(cells, runifpoint(6), periodic=TRUE)
```

crossdist.ppx

Pairwise Distances Between Two Different Multi-Dimensional Point Patterns

Description

Computes the distances between pairs of points taken from two different multi-dimensional point patterns.

Usage

```
## S3 method for class 'ppx'
crossdist(X, Y, ...)
```

Arguments

X, Y	Multi-dimensional point patterns (objects of class "ppx").
...	Arguments passed to coords.ppx to determine which coordinates should be used.

Details

Given two point patterns in multi-dimensional space, this function computes the Euclidean distance from each point in the first pattern to each point in the second pattern, and returns a matrix containing these distances.

This is a method for the generic function [crossdist](#) for three-dimensional point patterns (objects of class "ppx").

This function expects two multidimensional point patterns X and Y, and returns the matrix whose [i, j] entry is the distance from X[i] to Y[j].

By default, both spatial and temporal coordinates are extracted. To obtain the spatial distance between points in a space-time point pattern, set `temporal=FALSE`.

Value

A matrix whose [i, j] entry is the distance from the i-th point in X to the j-th point in Y.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

See Also

[crossdist](#), [pairdist](#), [nndist](#)

Examples

```
df <- data.frame(x=runif(3),y=runif(3),z=runif(3),w=runif(3))
X <- ppx(data=df)
df <- data.frame(x=runif(5),y=runif(5),z=runif(5),w=runif(5))
Y <- ppx(data=df)
d <- crossdist(X, Y)
```

`crossdist.psp`

Pairwise distances between two different line segment patterns

Description

Computes the distances between all pairs of line segments taken from two different line segment patterns.

Usage

```
## S3 method for class 'psp'
crossdist(X, Y, ..., method="Fortran", type="Hausdorff")
```

Arguments

<code>X, Y</code>	Line segment patterns (objects of class "psp").
<code>...</code>	Ignored.
<code>method</code>	String specifying which method of calculation to use. Values are "Fortran" and "interpreted".
<code>type</code>	Type of distance to be computed. Options are "Hausdorff" and "separation". Partial matching is used.

Details

This is a method for the generic function [crossdist](#).

Given two line segment patterns, this function computes the distance from each line segment in the first pattern to each line segment in the second pattern, and returns a matrix containing these distances.

The distances between line segments are measured in one of two ways:

- if `type="Hausdorff"`, distances are computed in the Hausdorff metric. The Hausdorff distance between two line segments is the *maximum* distance from any point on one of the segments to the nearest point on the other segment.
- if `type="separation"`, distances are computed as the *minimum* distance from a point on one line segment to a point on the other line segment. For example, line segments which cross over each other have separation zero.

The argument `method` is not normally used. It is retained only for checking the validity of the software. If `method = "interpreted"` then the distances are computed using interpreted R code only. If `method="Fortran"` (the default) then Fortran code is used. The Fortran code is several times faster.

Value

A matrix whose [i, j] entry is the distance from the i-th line segment in X to the j-th line segment in Y.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[pairdist](#), [nndist](#), [Gest](#)

Examples

```
L1 <- psp(runif(5), runif(5), runif(5), runif(5), owin())
L2 <- psp(runif(10), runif(10), runif(10), runif(10), owin())
D <- crossdist(L1, L2)
#result is a 5 x 10 matrix
S <- crossdist(L1, L2, type="sep")
```

crossing.psp

Crossing Points of Two Line Segment Patterns

Description

Finds any crossing points between two line segment patterns.

Usage

`crossing.psp(A, B)`

Arguments

`A, B` Line segment patterns (objects of class "psp").

Details

This function finds any crossing points between the line segment patterns A and B.

A crossing point occurs whenever one of the line segments in A intersects one of the line segments in B, at a nonzero angle of intersection.

Value

Point pattern (object of class "ppp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[selfcrossing.psp](#), [psp.object](#), [ppp.object](#).

Examples

```
a <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
b <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
plot(a, col="green", main="crossing.psp")
plot(b, add=TRUE, col="blue")
P <- crossing.psp(a,b)
plot(P, add=TRUE, col="red")
```

cut.im

Convert Pixel Image from Numeric to Factor

Description

Transform the values of a pixel image from numeric values into a factor.

Usage

```
## S3 method for class 'im'
cut(x, ...)
```

Arguments

- x A pixel image. An object of class "im".
- ... Arguments passed to [cut.default](#). They determine the breakpoints for the mapping from numerical values to factor values. See [cut.default](#).

Details

This simple function applies the generic [cut](#) operation to the pixel values of the image x. The range of pixel values is divided into several intervals, and each interval is associated with a level of a factor. The result is another pixel image, with the same window and pixel grid as x, but with the numeric value of each pixel discretised by replacing it by the factor level.

This function is a convenient way to inspect an image and to obtain summary statistics. See the examples.

To select a subset of an image, use the subset operator [\[.im](#) instead.

Value

A pixel image (object of class "im") with pixel values that are a factor. See [im.object](#).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[cut](#), [im.object](#)

Examples

```
# artificial image data
Z <- setcov(square(1))

Y <- cut(Z, 3)
Y <- cut(Z, breaks=seq(0,1,length=5))

# cut at the quartiles
# (divides the image into 4 equal areas)
Y <- cut(Z, quantile(Z))
```

cut.hpp

Classify Points in a Point Pattern

Description

Classifies the points in a point pattern into distinct types according to the numerical marks in the pattern, or according to another variable.

Usage

```
## S3 method for class 'ppp'
cut(x, z=marks(x), ...)
```

Arguments

- x A two-dimensional point pattern. An object of class "ppp".
- z Data determining the classification. A numeric vector, a factor, a pixel image, a tessellation, or a string giving the name of a column of marks.
- ... Arguments passed to [cut.default](#). They determine the breakpoints for the mapping from numerical values in z to factor values in the output. See [cut.default](#).

Details

This function has the effect of classifying each point in the point pattern x into one of several possible types. The classification is based on the dataset z, which may be either

- a factor (of length equal to the number of points in z) determining the classification of each point in x. Levels of the factor determine the classification.
- a numeric vector (of length equal to the number of points in z). The range of values of z will be divided into bands (the number of bands is determined by ...) and z will be converted to a factor using [cut.default](#).
- a pixel image (object of class "im"). The value of z at each point of x will be used as the classifying variable.
- a tessellation (object of class "tess", see [tess](#)). Each point of x will be classified according to the tile of the tessellation into which it falls.
- a character string, giving the name of one of the columns of `marks(x)`, if this is a data frame.

The default is to take z to be the vector of marks in x (or the first column in the data frame of marks of x , if it is a data frame). If the marks are numeric, then the range of values of the numerical marks is divided into several intervals, and each interval is associated with a level of a factor. The result is a marked point pattern, with the same window and point locations as x , but with the numeric mark of each point discretised by replacing it by the factor level. This is a convenient way to transform a marked point pattern which has numeric marks into a multitype point pattern, for example to plot it or analyse it. See the examples.

To select some points from a point pattern, use the subset operator `[.ppp` instead.

Value

A multitype point pattern, that is, a point pattern object (of class "ppp") with a `marks` vector that is a factor.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`cut`, `ppp.object`, `tess`

Examples

```
# (1) cutting based on numeric marks of point pattern

data(longleaf)
# Longleaf Pines data
# the marks are positive real numbers indicating tree diameters.

## Not run:
plot(longleaf)

## End(Not run)

# cut the range of tree diameters into three intervals
long3 <- cut(longleaf, breaks=3)
## Not run:
plot(long3)

## End(Not run)

# adult trees defined to have diameter at least 30 cm
long2 <- cut(longleaf, breaks=c(0,30,100), labels=c("Sapling", "Adult"))
plot(long2)
plot(long2, cols=c("green", "blue"))

# (2) cutting based on another numeric vector
# Divide Swedish Pines data into 3 classes
# according to nearest neighbour distance

data(swedishpines)
plot(cut(swedishpines, nndist(swedishpines), breaks=3))
```

```

# (3) cutting based on tessellation
# Divide Swedish Pines study region into a 4 x 4 grid of rectangles
# and classify points accordingly

tes <- tess(xgrid=seq(0,96,length=5),ygrid=seq(0,100,length=5))
plot(cut(swedishpines, tes))
plot(tes, lty=2, add=TRUE)

# (4) multivariate marks
data(finlpines)
cut(finlpines, "height", breaks=4)

```

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppm.object](#), [ppp.object](#)

Examples

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

`default.dummy`

Generate a Default Pattern of Dummy Points

Description

Generates a default pattern of dummy points for use in a quadrature scheme.

Usage

```
default.dummy(X, nd, random=FALSE, ntile=NULL, npix=NULL, ..., verbose=FALSE)
```

Arguments

<code>X</code>	The observed data point pattern. An object of class "ppp" or in a format recognised by as.ppp()
<code>nd</code>	Optional. Integer, or integer vector of length 2, specifying an <code>nd * nd</code> or <code>nd[1] * nd[2]</code> rectangular array of dummy points.
<code>random</code>	Logical value. If TRUE, the dummy points are randomised.
<code>ntile</code>	Optional. Integer or pair of integers specifying the number of rows and columns of tiles used in the counting rule.
<code>npix</code>	Optional. Integer or pair of integers specifying the number of rows and columns of pixels used in computing approximate areas.
<code>...</code>	Ignored.
<code>verbose</code>	If TRUE, information about the construction of the quadrature scheme is printed.

Details

This function provides a sensible default for the dummy points in a quadrature scheme.

A quadrature scheme consists of the original data point pattern, an additional pattern of dummy points, and a vector of quadrature weights for all these points. See [quad.object](#) for further information about quadrature schemes.

If `random` is false (the default), then the function creates dummy points in an `nd[1]` by `nd[1]` rectangular grid. If `random` is true, then the frame of the window is divided into an `nd[1]` by `nd[1]` array of tiles, and one dummy point is generated at random inside each tile. In either case, the four corner points of the frame of the window are added. Then if the window is not rectangular, any dummy points lying outside it are deleted.

If `nd` is missing, a default value (depending on the data pattern `X`) is computed by [default.ngrid](#).

Alternative functions for creating dummy patterns include [corners](#), [gridcentres](#), [stratrand](#) and [spokes](#).

Value

A point pattern (an object of class "ppp", see [ppp.object](#)) containing the dummy points.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[quad.object](#), [quadscheme](#), [corners](#), [gridcentres](#), [stratrand](#), [spokes](#)

Examples

```
data(simdat)
P <- simdat
D <- default.dummy(P, 100)
## Not run: plot(D)
Q <- quadscheme(P, D, "grid")
## Not run: plot(union.quad(Q))
```

default.expand

Default Expansion Rule for Simulation of Model

Description

Defines the default expansion window or expansion rule for simulation of a fitted point process model.

Usage

```
default.expand(object, m=2, epsilon=1e-6)
```

Arguments

- | | |
|---------|--|
| object | A point process model (object of class "ppm" or "rmhmodel"). |
| m | A single numeric value. The window will be expanded by a distance $m * \text{reach}(\text{object})$ along each side. |
| epsilon | Threshold argument passed to reach to determine $\text{reach}(\text{object})$. |

Details

This function computes a default value for the expansion rule (the argument `expand` in [rmhcontrol](#)) given a fitted point process model `object`. This default is used by [envelope](#), [qqplot.ppm](#), [simulate.ppm](#) and other functions.

Suppose we wish to generate simulated realisations of a fitted point process model inside a window w . It is advisable to first simulate the pattern on a larger window, and then clip it to the original window w . This avoids edge effects in the simulation. It is called *expansion* of the simulation window.

Accordingly, for the Metropolis-Hastings simulation algorithm [rmh](#), the algorithm control parameters specified by [rmhcontrol](#) include an argument `expand` that determines the expansion of the simulation window.

The function `default.expand` determines the default expansion rule for a fitted point process model `object`.

If the model is Poisson, then no expansion is necessary. No expansion is performed by default, and `default.expand` returns a rule representing no expansion. The simulation window is the original window `w = as.owin(object)`.

If the model depends on external covariates (i.e.\ covariates other than the Cartesian covariates `x` and `y` and the marks) then no expansion is feasible, in general, because the spatial domain of the covariates is not guaranteed to be large enough. `default.expand` returns a rule representing no expansion. The simulation window is the original window `w = as.owin(object)`.

If the model depends on the Cartesian covariates `x` and `y`, it would be feasible to expand the simulation window, and this was the default for **spatstat** version 1.24-1 and earlier. However this sometimes produces artefacts (such as an empty point pattern) or memory overflow, because the fitted trend, extrapolated outside the original window of the data, may become very large. In **spatstat** version 1.24-2 and later, the default rule is *not* to expand if the model depends on `x` or `y`. Again `default.expand` returns a rule representing no expansion.

Otherwise, expansion will occur. The original window `w = as.owin(object)` is expanded by a distance `m * rr`, where `rr` is the interaction range of the model, computed by `reach`. If `w` is a rectangle then each edge of `w` is displaced outward by distance `m * rr`. If `w` is not a rectangle then `w` is dilated by distance `m * rr` using `dilation`.

Value

A window expansion rule (object of class "rmhexpand").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`rmhexpand`, `rmhcontrol`, `rmh`, `envelope`, `qqplot.ppm`

Examples

```
data(cells)
fit <- ppm(cells, ~1, Strauss(0.07))
default.expand(fit)
mod <- rmhmodel(cif="strauss", par=list(beta=100, gamma=0.5, r=0.07))
default.expand(fit)
```

`default.rmhcontrol` *Set Default Control Parameters for Metropolis-Hastings Algorithm.*

Description

Given a fitted point process model, this command sets appropriate default values of the parameters controlling the iterative behaviour of the Metropolis-Hastings algorithm.

Usage

```
default.rmhcontrol(model)
```

Arguments

model	A fitted point process model (object of class "ppm")
-------	--

Details

This function sets the values of the parameters controlling the iterative behaviour of the Metropolis-Hastings simulation algorithm. It uses default values that would be appropriate for the fitted point process model `model`.

The expansion parameter `expand` is set to `default.expand(model)`.

All other parameters revert to their defaults given in `rmhcontrol.default`.

See `rmhcontrol` for the full list of control parameters. To override default parameters, use `update.rmhcontrol`.

Value

An object of class "rmhcontrol". See `rmhcontrol`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`rmhcontrol`, `update.rmhcontrol`, `ppm`, `default.expand`

Examples

```
fit <- ppm(cells, ~1, Strauss(0.1))
default.rmhcontrol(fit)
```

delaunay

*Delaunay Triangulation of Point Pattern***Description**

Computes the Delaunay triangulation of a spatial point pattern.

Usage

```
delaunay(X)
```

Arguments

X	Spatial point pattern (object of class "ppp").
---	--

Details

The Delaunay triangulation of a spatial point pattern `X` is defined as follows. First the Dirichlet/Voronoi tessellation of `X` computed; see `dirichlet`. Then two points of `X` are defined to be Delaunay neighbours if their Dirichlet/Voronoi tiles share a common boundary. Every pair of Delaunay neighbours is joined by a straight line. The result is a tessellation, consisting of disjoint triangles. The union of these triangles is the convex hull of `X`.

Value

A tessellation (object of class "tess"). The window of the tessellation is the convex hull of X , not the original window of X .

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[tess](#), [dirichlet](#), [convexhull.xy](#), [ppp](#)

Examples

```
X <- runifpoint(42)
plot(delaunay(X))
plot(X, add=TRUE)
```

deltametric

Delta Metric

Description

Computes the discrepancy between two sets A and B according to Baddeley's delta-metric.

Usage

```
deltametric(A, B, p = 2, c = Inf, ...)
```

Arguments

A, B	The two sets which will be compared. Windows (objects of class "owin"), point patterns (objects of class "ppp") or line segment patterns (objects of class "psp").
p	Index of the L^p metric. Either a positive numeric value, or Inf .
c	Distance threshold. Either a positive numeric value, or Inf .
...	Arguments passed to as.mask to determine the pixel resolution of the distance maps computed by distmap .

Details

Baddeley (1992a, 1992b) defined a distance between two sets A and B contained in a space W by

$$\Delta(A, B) = \left[\frac{1}{|W|} \int_W |\min(c, d(x, A)) - \min(c, d(x, B))|^p dx \right]^{1/p}$$

where $c \geq 0$ is a distance threshold parameter, $0 < p \leq \infty$ is the exponent parameter, and $d(x, A)$ denotes the shortest distance from a point x to the set A . Also $|W|$ denotes the area or volume of the containing space W .

This is defined so that it is a *metric*, i.e.

- $\Delta(A, B) = 0$ if and only if $A = B$
- $\Delta(A, B) = \Delta(B, A)$
- $\Delta(A, C) \leq \Delta(A, B) + \Delta(B, C)$

It is topologically equivalent to the Hausdorff metric (Baddeley, 1992a) but has better stability properties in practical applications (Baddeley, 1992b).

If $p = \infty$ and $c = \infty$ the Delta metric is equal to the Hausdorff metric.

The algorithm uses [distmap](#) to compute the distance maps $d(x, A)$ and $d(x, B)$, then approximates the integral numerically. The accuracy of the computation depends on the pixel resolution which is controlled through the extra arguments ... passed to [as.mask](#).

Value

A numeric value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Baddeley, A.J. (1992a) Errors in binary images and an L^p version of the Hausdorff metric. *Nieuw Archief voor Wiskunde* **10**, 157–183.
- Baddeley, A.J. (1992b) An error metric for binary images. In W. Foerstner and S. Ruwiedel (eds) *Robust Computer Vision*. Karlsruhe: Wichmann. Pages 59–78.

See Also

[distmap](#)

Examples

```
X <- runifpoint(20)
Y <- runifpoint(10)
deltametric(X, Y, p=1, c=0.1)
```

Description

This is an artificial dataset, for use in testing and demonstrating the capabilities of the [spatstat](#) package. It is a multitype point pattern in an irregular polygonal window. There are two types of points. The window contains a polygonal hole.

Usage

```
data(demopat)
```

Format

An object of class "ppp" representing the point pattern.

See [ppp.object](#) for details of the format of a point pattern object.

Source

Adrian Baddeley

density.hpp

Kernel Smoothed Intensity of Point Pattern

Description

Compute a kernel smoothed intensity function from a point pattern.

Usage

```
## S3 method for class 'ppp'
density(x, sigma, ...,
        weights, edge=TRUE, varcov=NULL,
        at="pixels", leaveoneout=TRUE,
        adjust=1, diggle=FALSE)
```

Arguments

x	Point pattern (object of class "ppp").
sigma	Standard deviation of isotropic Gaussian smoothing kernel. Either a numerical value, or a function that computes an appropriate value of sigma .
weights	Optional vector of weights to be attached to the points. May include negative values.
...	Arguments passed to as.mask to determine the pixel resolution.
edge	Logical flag: if TRUE, apply edge correction.
varcov	Variance-covariance matrix of anisotropic Gaussian kernel. Incompatible with sigma .
at	String specifying whether to compute the intensity values at a grid of pixel locations (at ="pixels") or only at the points of x (at ="points").
leaveoneout	Logical value indicating whether to compute a leave-one-out estimator. Applicable only when at ="points".
adjust	Optional. Adjustment factor for the smoothing parameter.
diggle	Logical. If TRUE, use Diggle's edge correction, which is more accurate but slower to compute than the correction described under Details.

Details

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

It computes a fixed-bandwidth kernel estimate (Diggle, 1985) of the intensity function of the point process that generated the point pattern `x`.

By default it computes the convolution of the isotropic Gaussian kernel of standard deviation `sigma` with point masses at each of the data points in `x`. Anisotropic Gaussian kernels are also supported. Each point has unit weight, unless the argument `weights` is given (it should be a numeric vector; weights can be negative or zero).

If `edge=TRUE`, the intensity estimate is corrected for edge effect bias in one of two ways:

- If `diggle=FALSE` (the default) the intensity estimate is corrected by dividing it by the convolution of the Gaussian kernel with the window of observation. Thus the intensity value at a point u is

$$\hat{\lambda}(u) = e(u) \sum_i k(x_i - u) w_i$$

where k is the Gaussian smoothing kernel, $e(u)$ is an edge correction factor, and w_i are the weights.

- If `diggle=TRUE` then the method of Diggle (1985) is followed exactly. The intensity value at a point u is

$$\hat{\lambda}(u) = \sum_i k(x_i - u) w_i e(x_i)$$

where again k is the Gaussian smoothing kernel, $e(x_i)$ is an edge correction factor, and w_i are the weights. This computation is slightly slower but more accurate.

In both cases, the edge correction term $e(u)$ is the reciprocal of the kernel mass inside the window:

$$\frac{1}{e(u)} = \int_W k(v - u) dv$$

where W is the observation window.

The smoothing kernel is determined by the arguments `sigma`, `varcov` and `adjust`.

- if `sigma` is a single numerical value, this is taken as the standard deviation of the isotropic Gaussian kernel.
- alternatively `sigma` may be a function that computes an appropriate bandwidth for the isotropic Gaussian kernel from the data point pattern by calling `sigma(x)`. To perform automatic bandwidth selection using cross-validation, it is recommended to use the function `bw.diggle`.
- The smoothing kernel may be chosen to be any Gaussian kernel, by giving the variance-covariance matrix `varcov`. The arguments `sigma` and `varcov` are incompatible.
- Alternatively `sigma` may be a vector of length 2 giving the standard deviations of two independent Gaussian coordinates, thus equivalent to `varcov = diag(rep(sigma^2, 2))`.
- if neither `sigma` nor `varcov` is specified, an isotropic Gaussian kernel will be used, with a default value of `sigma` calculated by a simple rule of thumb that depends only on the size of the window.
- The argument `adjust` makes it easy for the user to change the bandwidth specified by any of the rules above. The value of `sigma` will be multiplied by the factor `adjust`. The matrix `varcov` will be multiplied by `adjust^2`. To double the smoothing bandwidth, set `adjust=2`.

By default the intensity values are computed at every location u in a fine grid, and are returned as a pixel image. Computation is performed using the Fast Fourier Transform. Accuracy depends on the pixel resolution, controlled by the arguments ... passed to [as.mask](#).

If `at="points"`, the intensity values are computed to high accuracy at the points of x only. Computation is performed by directly evaluating and summing the Gaussian kernel contributions without discretising the data. The result is a numeric vector giving the density values. The intensity value at a point x_i is (if `diggle=FALSE`)

$$\hat{\lambda}(x_i) = e(x_i) \sum_j k(x_j - x_i) w_j$$

or (if `diggle=TRUE`)

$$\hat{\lambda}(x_i) = \sum_j k(x_j - x_i) w_j e(x_j)$$

If `leaveoneout=TRUE` (the default), then the sum in the equation is taken over all j not equal to i , so that the intensity value at a data point is the sum of kernel contributions from all *other* data points. If `leaveoneout=FALSE` then the sum is taken over all j , so that the intensity value at a data point includes a contribution from the same point.

To select the bandwidth `sigma` automatically by cross-validation, use [bw.diggle](#).

To perform spatial interpolation of values that were observed at the points of a point pattern, use [smooth.hpp](#).

For adaptive nonparametric estimation, see [adaptive.density](#). For data sharpening, see [sharpen.hpp](#).

To compute a relative risk surface or probability map for two (or more) types of points, use [relrisk](#).

Value

By default, the result is a pixel image (object of class "im"). Pixel values are estimated intensity values, expressed in "points per unit area".

If `at="points"`, the result is a numeric vector of length equal to the number of points in x . Values are estimated intensity values at the points of x .

In either case, the return value has attributes "sigma" and "varcov" which report the smoothing bandwidth that was used.

Note

This function is often misunderstood.

The result of `density.hpp` is not a spatial smoothing of the marks or weights attached to the point pattern. To perform spatial interpolation of values that were observed at the points of a point pattern, use [smooth.hpp](#).

The result of `density.hpp` is not a probability density. It is an estimate of the *intensity function* of the point process that generated the point pattern data. Intensity is the expected number of random points per unit area. The units of intensity are "points per unit area". Intensity is usually a function of spatial location, and it is this function which is estimated by `density.hpp`. The integral of the intensity function over a spatial region gives the expected number of points falling in this region.

Inspecting an estimate of the intensity function is usually the first step in exploring a spatial point pattern dataset. For more explanation, see the workshop notes (Baddeley, 2008) or Diggle (2003).

If you have two (or more) types of points, and you want a probability map or relative risk surface (the spatially-varying probability of a given type), use [relrisk](#).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

References

- Baddeley, A. (2010) Analysing spatial point patterns in R. Workshop notes. CSIRO online technical publication. URL: www.csiro.au/resources/pf16h.html
- Diggle, P.J. (1985) A kernel method for smoothing point process data. *Applied Statistics* (Journal of the Royal Statistical Society, Series C) **34** (1985) 138–147.
- Diggle, P.J. (2003) *Statistical analysis of spatial point patterns*, Second edition. Arnold.

See Also

[bw.diggle](#), [smooth.ppp](#), [sharpen.ppp](#), [adaptive.density](#), [relrisk](#), [ppp.object](#), [im.object](#)

Examples

```
data(cells)
if(interactive()) {
  opa <- par(mfrow=c(1,2))
  plot(density(cells, 0.05))
  plot(density(cells, 0.05, diggle=TRUE))
  par(opa)
  v <- diag(c(0.05, 0.07)^2)
  plot(density(cells, varcov=v))
}

# automatic bandwidth selection
plot(density(cells, sigma=bw.diggle(cells)))
# equivalent:
plot(density(cells, bw.diggle))
# evaluate intensity at points
density(cells, 0.05, at="points")
```

Description

Compute a kernel smoothed intensity function from a line segment pattern.

Usage

```
## S3 method for class 'psp'
density(x, sigma, ..., edge=TRUE)
```

Arguments

<code>x</code>	Line segment pattern (object of class "psp") to be smoothed.
<code>sigma</code>	Standard deviation of isotropic Gaussian smoothing kernel.
<code>...</code>	Extra arguments passed to <code>as.mask</code> which determine the resolution of the resulting image.
<code>edge</code>	Logical flag indicating whether to apply edge correction.

Details

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

A kernel estimate of the intensity of the line segment pattern is computed. The result is the convolution of the isotropic Gaussian kernel, of standard deviation `sigma`, with the line segments. Computation is performed analytically.

If `edge=TRUE` this result is adjusted for edge effects by dividing it by the convolution of the same Gaussian kernel with the observation window.

Value

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

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`psp.object`, `im.object`, `density`

Examples

```
L <- psp(runif(20),runif(20),runif(20),runif(20), window=owin())
D <- density(L, sigma=0.03)
plot(D, main="density(L)")
plot(L, add=TRUE)
```

Description

Compute a kernel smoothed intensity function for each of the components of a split point pattern.

Usage

```
## S3 method for class 'splitppp'
density(x, ...)
```

Arguments

- x Split point pattern (object of class "splitppp" created by [split.ppp](#)) to be smoothed.
- ... Arguments passed to [density.ppp](#) to control the smoothing, pixel resolution, edge correction etc.

Details

This is a method for the generic function [density](#).

The argument x should be an object of class "splitppp", effectively a list of point patterns.

Typically x is obtained by applying the function [split.ppp](#) to a point pattern y by calling [split\(y\)](#). This splits the points of y into several sub-patterns.

A kernel estimate of the intensity function of each of the point patterns is computed using [density.ppp](#).

The return value is a list, each of whose entries is a pixel image (object of class "im"). The return value also belongs to the class "listof" and can be plotted or printed.

Value

A list of pixel images (objects of class "im"). Can be plotted or printed.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppp.object](#), [im.object](#)

Examples

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

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=list())
```

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, this function computes the influence measure for each parameter, as described in Baddeley, Chang and Song (2011).

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.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

References

Baddeley, A. and Chang, Y.M. and Song, Y. (2011) Leverage and influence diagnostics for spatial point process models. *Scandinavian Journal of Statistics*, in press.

See Also

`leverage.ppm`, `influence.ppm`

Examples

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

plot(dfbetas(fit))
plot(smooth.msr(dfbetas(fit)))
```

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=TRUE,
              compute.cts=TRUE, typename, check=TRUE, repair=TRUE,
              oldstyle=FALSE)
## S3 method for class 'diagppm'
plot(x, ..., which,
      plot.neg="image", plot.smooth="imagecontour",
      plot.sd=TRUE, spacing=0.1,
      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.
cumulative	Logical flag indicating whether the lurking variable plots for the <i>x</i> and <i>y</i> 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	One of "discrete" or "image" indicating how the density part of the residual measure should be plotted.
plot.smooth	One of "image", "persp", "contour" or "imagecontour" 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.
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.
strange	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).
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

This function 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 $E(h(x)) = \text{area of the subset of the window to the left of the line } X = x$.
- 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 $E(h(x)) = \text{length of the intersection between the observation window and the line } X = x$.

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.

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"`. There are methods for `plot` and `print` for such objects. See the Examples.

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`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

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

Baddeley, A., Moller, 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

```
data(cells)
fit <- ppm(cells, ~x, Strauss(r=0.15))
diagnose.ppm(fit)
## Not run:
diagnose.ppm(fit, type="pearson")

## End(Not run)
diagnose.ppm(fit, which="marks")
diagnose.ppm(fit, type="raw", plot.neg="discrete")

# save the diagnostics and plot them later
u <- diagnose.ppm(fit, rbord=0.15, plot.it=FALSE)
## Not run:
plot(u)
plot(u, which="marks")

## End(Not run)
```

diameter

Diameter of an Object

Description

Computes the diameter of an object such as a two-dimensional window or three-dimensional box.

Usage

```
diameter(x)
```

Arguments

x	A window or other object whose diameter will be computed.
---	---

Details

This function computes the diameter of an object such as a two-dimensional window or a three-dimensional box. The diameter is the maximum distance between any two points in the object.

The function `diameter` is generic, with methods for the class "owin" (two-dimensional windows), "box3" (three-dimensional boxes) and "boxx" (multi-dimensional boxes).

Value

The numerical value of the diameter of the object.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[diameter.owin](#), [diameter.box3](#), [diameter.boxx](#)

diameter.box3

Geometrical Calculations for Three-Dimensional Box

Description

Calculates the volume, diameter, shortest side, or eroded volume of a three-dimensional box.

Usage

```
## S3 method for class 'box3'
diameter(x)
## S3 method for class 'box3'
volume(x)
## S3 method for class 'box3'
shortside(x)
## S3 method for class 'box3'
eroded.volumes(x, r)
shortside(x)
eroded.volumes(x, r)
```

Arguments

- x Three-dimensional box (object of class "box3").
- r Numeric value or vector of numeric values for which eroded volumes should be calculated.

Details

`diameter.box3` computes the diameter of the box. `volume.box3` computes the volume of the box. `shortside.box3` finds the shortest of the three side lengths of the box.

`eroded.volumes` computes, for each entry `r[i]`, the volume of the smaller box obtained by removing a slab of thickness `r[i]` from each face of the box. This smaller box is the subset consisting of points that lie at least `r[i]` units away from the boundary of the box.

Value

For `diameter.box3`, `shortside.box3` and `volume.box3`, a single numeric value. For `eroded.volumes`, a numeric vector of the same length as `r`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[as.box3](#)

Examples

```
X <- box3(c(0,10),c(0,10),c(0,5))
diameter(X)
volume(X)
shortside(X)
hd <- shortside(X)/2
eroded.volumes(X, seq(0,hd, length=10))
```

Description

Calculates the volume, diameter, shortest side, or eroded volume of a multi-dimensional box.

Usage

```
## S3 method for class 'boxx'
diameter(x)
## S3 method for class 'boxx'
volume(x)
## S3 method for class 'boxx'
shortside(x)
## S3 method for class 'boxx'
eroded.volumes(x, r)
```

Arguments

- x Multi-dimensional box (object of class "boxx").
- r Numeric value or vector of numeric values for which eroded volumes should be calculated.

Details

diameter.boxx, *volume.boxx* and *shortside.boxx* compute the diameter, volume and shortest side length of the box.

eroded.volumes.boxx computes, for each entry *r[i]*, the volume of the smaller box obtained by removing a slab of thickness *r[i]* from each face of the box. This smaller box is the subset consisting of points that lie at least *r[i]* units away from the boundary of the box.

Value

For `diameter.boxx`, `shortside.boxx` and `volume.boxx`, a single numeric value. For `eroded.volumes.boxx`, a numeric vector of the same length as `r`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[boxx](#)

Examples

```
X <- boxx(c(0,10),c(0,10),c(0,5))
diameter(X)
volume(X)
shortside(X)
hd <- shortside(X)/2
eroded.volumes(X, seq(0,hd, length=10))
```

`diameter.owin`

Diameter of a Window

Description

Computes the diameter of a window.

Usage

```
## S3 method for class 'owin'
diameter(x)
```

Arguments

`x` A window whose diameter will be computed.

Details

This function computes the diameter of a window of arbitrary shape, i.e. the maximum distance between any two points in the window.

The argument `x` should be a window (an object of class "owin", see `owin.object` for details) or can be given in any format acceptable to `as.owin()`.

The function `diameter` is generic. This function is the method for the class "owin".

Value

The numerical value of the diameter of the .

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[area.owin](#), [perimeter](#), [owin](#), [as.owin](#)

Examples

```
w <- owin(c(0,1),c(0,1))
diameter(w)
# returns sqrt(2)
data(letterR)
diameter(letterR)
```

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 `rho` 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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

References

- Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.
- Ripley, B.D. (1981) *Spatial statistics*. John Wiley and Sons.
- 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

data(cells)

## Not run:
ppm(cells, ~1, DiggleGatesStibbard(0.05))
# fit the stationary D-G-S process to 'cells'

## End(Not run)

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, 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

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

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 $\delta \leq \rho$.

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.

Value

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

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

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

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

dilated.areas	<i>Areas of Morphological Dilations</i>
---------------	---

Description

Computes the areas of successive morphological dilations.

Usage

```
dilated.areas(X, r, W=as.owin(X), ..., constrained=TRUE, exact = FALSE)
```

Arguments

X	Object to be dilated. A point pattern (object of class "ppp"), a line segment pattern (object of class "psp"), or a window (object of class "owin").
r	Numeric vector of radii for the dilations.
W	Window (object of class "owin") inside which the areas will be computed, if constrained=TRUE.
...	Ignored.
constrained	Logical flag indicating whether areas should be restricted to the window W.
exact	Logical flag indicating whether areas should be computed using analytic geometry (which is slower but more accurate). Currently available only when X is a point pattern.

Details

This function computes the areas of the dilations of X by each of the radii $r[i]$. Areas may also be computed inside a specified window W.

The morphological dilation of a set X by a distance $r > 0$ is the subset consisting of all points x such that the distance from x to X is less than or equal to r .

When X is a point pattern, the dilation by a distance r is the union of discs of radius r centred at the points of X.

The argument r should be a vector of nonnegative numbers.

If exact=TRUE and if X is a point pattern, then the areas are computed using analytic geometry, which is slower but much more accurate. Otherwise the computation is performed using [distmap](#).

To compute the dilated object itself, use [dilation](#).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[owin](#), [as.owin](#), [dilation](#), [eroded.areas](#)

Examples

```
X <- runifpoint(10)
a <- dilated.areas(X, c(0.1,0.2), W=square(1), exact=TRUE)
```

dilation	<i>Morphological Dilation</i>
----------	-------------------------------

Description

Perform morphological dilation of a window, a line segment pattern or a point pattern

Usage

```
dilation(w, r, ...)
## S3 method for class 'owin'
dilation(w, r, ..., polygonal=NULL, tight=TRUE)
## S3 method for class 'ppp'
dilation(w, r, ..., polygonal=TRUE, tight=TRUE)
## S3 method for class 'psp'
dilation(w, r, ..., polygonal=TRUE, tight=TRUE)
```

Arguments

<code>w</code>	A window (object of class "owin" or a line segment pattern (object of class "psp") or a point pattern (object of class "ppp").
<code>r</code>	positive number: the radius of dilation.
<code>...</code>	extra arguments passed to <code>as.mask</code> controlling the pixel resolution, if the pixel approximation is used.
<code>polygonal</code>	Logical flag indicating whether to compute a polygonal approximation to the erosion (<code>polygonal=TRUE</code>) or a pixel grid approximation (<code>polygonal=FALSE</code>).
<code>tight</code>	Logical flag indicating whether the bounding frame of the window should be taken as the smallest rectangle enclosing the dilated region (<code>tight=TRUE</code>), or should be the dilation of the bounding frame of <code>w</code> (<code>tight=FALSE</code>).

Details

The morphological dilation of a set W by a distance $r > 0$ is the set consisting of all points lying at most r units away from W . Effectively, dilation adds a margin of width r onto the set W .

If `polygonal=TRUE` then a polygonal approximation to the dilation is computed. If `polygonal=FALSE` then a pixel approximation to the dilation is computed from the distance map of `w`. The arguments "`...`" are passed to `as.mask` to control the pixel resolution.

When `w` is a window, the default (when `polygonal=NULL`) is to compute a polygonal approximation if `w` is a rectangle or polygonal window, and to compute a pixel approximation if `w` is a window of type "mask".

Value

If $r > 0$, an object of class "owin" representing the dilated region. If $r=0$, the result is identical to `w`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[erosion](#) for the opposite operation.
[owin](#), [as.owin](#)

Examples

```
w <- owin(c(0,1),c(0,1))
v <- dilation(w, 0.1)
```

dirichlet*Dirichlet Tessellation of Point Pattern*

Description

Computes the Dirichlet/Voronoi tessellation of a spatial point pattern.

Usage

```
dirichlet(X)
```

Arguments

X Spatial point pattern (object of class "ppp").

Details

In a spatial point pattern X, the Dirichlet/Voronoi tile associated with a particular point X[i] is the region of space that is closer to X[i] than to any other point in X. The Dirichlet tiles divide the two-dimensional plane into disjoint regions, forming a tessellation.

This function computes the Dirichlet tessellation (within the original window of X).

Value

A tessellation (object of class "tess").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[tess](#), [delaunay](#), [ppp](#)

Examples

```
X <- runifpoint(42)
plot(dirichlet(X))
plot(X, add=TRUE)
```

dirichlet.weights *Compute Quadrature Weights Based on Dirichlet Tessellation*

Description

Computes quadrature weights for a given set of points, using the areas of tiles in the Dirichlet tessellation.

Usage

```
dirichlet.weights(X, window=NULL, exact=TRUE, ...)
```

Arguments

X	Data defining a point pattern.
window	Default window for the point pattern
exact	Logical value. If TRUE, compute exact areas using the package <code>deldir</code> . If FALSE, compute approximate areas using a pixel raster.
...	Ignored.

Details

This function computes a set of quadrature weights for a given pattern of points (typically comprising both “data” and ‘dummy’ points). See [quad.object](#) for an explanation of quadrature weights and quadrature schemes.

The weights are computed using the Dirichlet tessellation. First X and (optionally) window are converted into a point pattern object. Then the Dirichlet tessellation of the points of X is computed. The weight attached to a point of X is the area of its Dirichlet tile (inside the window X>window).

If exact=TRUE the Dirichlet tessellation is computed exactly by the Lee-Schachter algorithm using the package `deldir`. Otherwise a pixel raster approximation is constructed and the areas are approximations to the true weights. In all cases the sum of the weights is equal to the area of the window.

Value

Vector of nonnegative weights for each point in X.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[quad.object](#), [gridweights](#)

Examples

```
Q <- quadscheme(runifpoispp(10))
X <- as.ppp(Q) # data and dummy points together
w <- dirichlet.weights(X, exact=FALSE)
```

disc*Circular Window*

Description

Creates a circular window

Usage

```
disc(radius=1, centre=c(0,0), ..., mask=FALSE, npoly=128)
```

Arguments

radius	Radius of the circle.
centre	Coordinates of the centre of the circle.
mask	Logical flag controlling the type of approximation to a perfect circle. See Details.
npoly	Number of edges of the polygonal approximation, if <code>mask=FALSE</code> .
...	Arguments passed to <code>as.mask</code> determining the pixel resolution, if <code>mask=TRUE</code> .

Details

This command creates a window object representing a disc, with the given radius and centre.

By default, the circle is approximated by a polygon with `npoly` edges.

If `mask=TRUE`, then the disc is approximated by a binary pixel mask. The resolution of the mask is controlled by the arguments ... which are passed to `as.mask`.

Value

An object of class "owin" (see `owin.object`) specifying a window.

Note

This function can also be used to generate regular polygons, by setting `npoly` to a small integer value. For example `npoly=5` generates a pentagon and `npoly=13` a triskaidecagon.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`owin.object`, `owin`, `as.mask`

Examples

```
# unit disc
W <- disc()
# disc of radius 3 centred at x=10, y=5
W <- disc(3, c(10,5))
#
plot(disc())
plot(disc(mask=TRUE))
# nice smooth circle
plot(disc(npoly=256))
# how to control the resolution of the mask
plot(disc(mask=TRUE, dimyx=256))
# check accuracy of approximation
area.owin(disc())/pi
area.owin(disc(mask=TRUE))/pi
```

discpartarea

Area of Part of Disc

Description

Compute area of intersection between a disc and a window

Usage

```
discpartarea(X, r, W=as.owin(X))
```

Arguments

- | | |
|---|---|
| X | Point pattern (object of class "ppp") specifying the centres of the discs. Alternatively, X may be in any format acceptable to as.ppp . |
| r | Matrix, vector or numeric value specifying the radii of the discs. |
| W | Window (object of class "owin") with which the discs should be intersected. |

Details

This algorithm computes the exact area of the intersection between a window W and a disc (or each of several discs). The centres of the discs are specified by the point pattern X, and their radii are specified by r.

If r is a single numeric value, then the algorithm computes the area of intersection between W and the disc of radius r centred at each point of X, and returns a one-column matrix containing one entry for each point of X.

If r is a vector of length m, then the algorithm returns an n * m matrix in which the entry on row i, column j is the area of the intersection between W and the disc centred at X[i] with radius r[j].

If r is a matrix, it should have one row for each point in X. The algorithm returns a matrix in which the entry on row i, column j is the area of the intersection between W and the disc centred at X[i] with radius r[i, j].

Areas are computed by analytic geometry.

Value

Numeric matrix, with one row for each point of X.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[owin](#), [disc](#)

Examples

```
data(letterR)
X <- runifpoint(3, letterR)
discpartarea(X, 0.2)
```

discretise

Safely Convert Point Pattern Window to Binary Mask

Description

Given a point pattern, discretise its window by converting it to a binary pixel mask, adjusting the mask so that it still contains all the points.

Usage

```
discretise(X, eps = NULL, dimyx = NULL, xy = NULL)
```

Arguments

X	A point pattern (object of class "ppp") to be converted.
eps	(optional) width and height of each pixel
dimyx	(optional) pixel array dimensions
xy	(optional) pixel coordinates

Details

This function modifies the point pattern X by converting its observation window X>window to a binary pixel image (a window of type "mask"). It ensures that no points of X are deleted by the discretisation.

The window is first discretised using [as.mask](#). It can happen that points of X that were inside the original window may fall outside the new mask. The discretise function corrects this by augmenting the mask (so that the mask includes any pixel that contains a point of the pattern).

The arguments eps, dimyx and xy control the fineness of the pixel array. They are passed to [as.mask](#).

If eps, dimyx and xy are all absent or NULL, and if the window of X is of type "mask" to start with, then discretise(X) returns X unchanged.

See [as.mask](#) for further details about the arguments eps, dimyx, and xy, and the process of converting a window to one of type mask.

Value

A point pattern (object of class "ppp"), identical to X, except that its observation window has been converted to one of type mask.

Error checking

Before doing anything, *discretise* checks that all the points of the pattern are actually inside the original window. This is guaranteed to be the case if the pattern was constructed using `ppp` or `as.ppp`. However anomalies are possible if the point pattern was created or manipulated inappropriately. These will cause an error.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`as.mask`

Examples

```
data(demopat)
X <- demopat
plot(X, main="original pattern")
Y <- discretise(X, dimyx=50)
plot(Y, main="discretise(X)")
stopifnot(X$n == Y$n)

# what happens if we just convert the window to a mask?
W <- X>window
M <- as.mask(W, dimyx=50)
plot(M, main="window of X converted to mask")
plot(X, add=TRUE, pch=16)
plot(X[M], add=TRUE, pch=1, cex=1.5)
XM <- X[M]
cat(paste(X$n - XM$n, "points of X lie outside M\n"))
```

Description

Compute the distance function of an object, and return it as a function.

Usage

```
distfun(X, ...)
## S3 method for class 'ppp'
distfun(X, ...)
## S3 method for class 'psp'
distfun(X, ...)
## S3 method for class 'owin'
distfun(X, ..., invert=FALSE)
```

Arguments

X	Any suitable dataset representing a two-dimensional object, such as a point pattern (object of class "ppp"), a window (object of class "owin") or a line segment pattern (object of class "psp").
...	Extra arguments are ignored.
invert	If TRUE, compute the distance transform of the complement of X.

Details

The “distance function” of a set of points A is the mathematical function f such that, for any two-dimensional spatial location (x, y) , the function value $f(x, y)$ is the shortest distance from (x, y) to A .

The command `f <- distfun(X)` returns a *function* in the R language, with arguments `x, y`, that represents the distance function of X . Evaluating the function f in the form `v <- f(x, y)`, where `x` and `y` are any numeric vectors of equal length containing coordinates of spatial locations, yields the values of the distance function at these locations.

This should be contrasted with the related command `distmap` which computes the distance function of X on a grid of locations, and returns the distance values in the form of a pixel image.

A `distfun` object can be converted to a pixel image using `as.im`.

Value

A function with arguments `x, y`. The function also belongs to the class "distfun" for which there are methods for `print`, `plot`, `contour` and `persp`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`distmap`, `plot.distfun`

Examples

```
data(letterR)
f <- distfun(letterR)
f(0.2, 0.3)
```

distmap

Distance Map

Description

Compute the distance map of an object, and return it as a pixel image. Generic.

Usage

```
distmap(X, ...)
```

Arguments

- X Any suitable dataset representing a two-dimensional object, such as a point pattern (object of class "ppp"), a window (object of class "owin") or a line segment pattern (object of class "psp").
 ... Arguments passed to `as.mask` to control pixel resolution.

Details

The “distance map” of a set of points A is the function f whose value $f(x)$ is defined for any two-dimensional location x as the shortest distance from x to A .

This function computes the distance map of the set X and returns the distance map as a pixel image.

This is generic. Methods are provided for point patterns (`distmap.ppp`), line segment patterns (`distmap.psp`) and windows (`distmap.owin`).

Value

A pixel image (object of class "im") whose grey scale values are the values of the distance map.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`distmap.ppp`, `distmap.psp`, `distmap.owin`, `distfun`

Examples

```
data(cells)
U <- distmap(cells)
data(letterR)
V <- distmap(letterR)
## Not run:
plot(U)
plot(V)

## End(Not run)
```

`distmap.owin`

Distance Map of Window

Description

Computes the distance from each pixel to the nearest point in the given window.

Usage

```
## S3 method for class 'owin'
distmap(X, ..., discretise=FALSE, invert=FALSE)
```

Arguments

X	A window (object of class "owin").
...	Arguments passed to <code>as.mask</code> to control pixel resolution.
discretise	Logical flag controlling the choice of algorithm when X is a polygonal window. See Details.
invert	If TRUE, compute the distance transform of the complement of the window.

Details

The “distance map” of a window W is the function f whose value $f(u)$ is defined for any two-dimensional location u as the shortest distance from u to W .

This function computes the distance map of the window X and returns the distance map as a pixel image. The greyscale value at a pixel u equals the distance from u to the nearest pixel in X.

Additionally, the return value has an attribute "bdry" which is also a pixel image. The grey values in "bdry" give the distance from each pixel to the bounding rectangle of the image.

If X is a binary pixel mask, the distance values computed are not the usual Euclidean distances. Instead the distance between two pixels is measured by the length of the shortest path connecting the two pixels. A path is a series of steps between neighbouring pixels (each pixel has 8 neighbours). This is the standard ‘distance transform’ algorithm of image processing (Rosenfeld and Kak, 1968; Borgefors, 1986).

If X is a polygonal window, then exact Euclidean distances will be computed if discretise=FALSE. If discretise=TRUE then the window will first be converted to a binary pixel mask and the discrete path distances will be computed.

The arguments ... are passed to `as.mask` to control the pixel resolution.

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

Value

A pixel image (object of class "im") whose greyscale values are the values of the distance map. The return value has an attribute "bdry" which is a pixel image.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

Borgefors, G. Distance transformations in digital images. *Computer Vision, Graphics and Image Processing* **34** (1986) 344–371.

Rosenfeld, A. and Pfalz, J.L. Distance functions on digital pictures. *Pattern Recognition* **1** (1968) 33-61.

See Also

`distmap`, `distmap.ppp`, `distmap.psp`

Examples

```
data(letterR)
U <- distmap(letterR)
## Not run:
plot(U)
plot(attr(U, "bdry"))

## End(Not run)
```

distmap.hpp

Distance Map of Point Pattern

Description

Computes the distance from each pixel to the nearest point in the given point pattern.

Usage

```
## S3 method for class 'ppp'
distmap(X, ...)
```

Arguments

- X A point pattern (object of class "ppp").
- ... Arguments passed to `as.mask` to control pixel resolution.

Details

The “distance map” of a point pattern X is the function f whose value $f(u)$ is defined for any two-dimensional location u as the shortest distance from u to X .

This function computes the distance map of the point pattern X and returns the distance map as a pixel image. The greyscale value at a pixel u equals the distance from u to the nearest point of the pattern X .

Additionally, the return value has two attributes, “index” and “bdry”, which are also pixel images. The grey values in “bdry” give the distance from each pixel to the bounding rectangle of the image. The grey values in “index” are integers identifying which point of X is closest.

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

Note that this function gives the distance from the *centre of each pixel* to the nearest data point. To compute the exact distance from a given spatial location to the nearest data point in X , use `distfun` or `nncross`.

Value

A pixel image (object of class “im”) whose greyscale values are the values of the distance map. The return value has attributes “index” and “bdry” which are also pixel images.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

Generic function [distmap](#) and other methods [distmap.psp](#), [distmap.owin](#).
 Generic function [distfun](#).
 Nearest neighbour distance [nncross](#)

Examples

```
data(cells)
U <- distmap(cells)
## Not run:
plot(U)
plot(attr(U, "bdry"))
plot(attr(U, "index"))

## End(Not run)
```

distmap.psp*Distance Map of Line Segment Pattern***Description**

Computes the distance from each pixel to the nearest line segment in the given line segment pattern.

Usage

```
## S3 method for class 'psp'
distmap(X, ...)
```

Arguments

X	A line segment pattern (object of class "psp").
...	Arguments passed to as.mask to control pixel resolution.

Details

The “distance map” of a line segment pattern X is the function f whose value $f(u)$ is defined for any two-dimensional location u as the shortest distance from u to X .

This function computes the distance map of the line segment pattern X and returns the distance map as a pixel image. The greyscale value at a pixel u equals the distance from u to the nearest line segment of the pattern X . Distances are computed using analytic geometry.

Additionally, the return value has two attributes, “index” and “bdry”, which are also pixel images. The grey values in “bdry” give the distance from each pixel to the bounding rectangle of the image. The grey values in “index” are integers identifying which line segment of X is closest.

This is a method for the generic function [distmap](#).

Note that this function gives the exact distance from the centre of each pixel to the nearest line segment. To compute the exact distance from the points in a point pattern to the nearest line segment, use [distfun](#) or one of the low-level functions [nncross](#) or [project2segment](#).

Value

A pixel image (object of class "im") whose greyscale values are the values of the distance map. The return value has attributes "index" and "bdry" which are also pixel images.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[distmap](#), [distmap.owin](#), [distmap.ppp](#), [distfun](#), [nncross](#), [nearestsegment](#), [project2segment](#).

Examples

```
a <- psp(runif(20),runif(20),runif(20),runif(20), window=owin())
Z <- distmap(a)
plot(Z)
plot(a, add=TRUE)
```

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppm.object](#), [ppp.object](#), [ppm](#)

Examples

```
data(cells)
fit <- ppm(cells, ~1, Strauss(r=0.1))
X <- dummy.ppm(fit)
X$n
# this is the number of dummy points in the quadrature scheme
```

duplicated.ppp

Determine Duplicated Points in a Spatial Point Pattern

Description

Determines which points in a spatial point pattern are duplicates of previous points, and returns a logical vector.

Usage

```
## S3 method for class 'ppp'
duplicated(x, ...)
```

Arguments

x	A spatial point pattern (object of class "ppp").
...	Ignored.

Details

This is a method for the generic function `duplicated` for point pattern datasets (of class "ppp", see [ppp.object](#)).

Two points in a point pattern are deemed to be identical if their x, y coordinates are the same, and their marks are also the same (if they carry marks). The Examples section illustrates how it is possible for a point pattern to contain a pair of identical points.

This function determines which points in `x` duplicate other points that appeared earlier in the sequence. It returns a logical vector with entries that are TRUE for duplicated points and FALSE for unique (non-duplicated) points.

Value

A logical vector of length equal to the number of points in `x`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppp.object](#), [unique.ppp](#), [multiplicity.ppp](#)

Examples

```
X <- ppp(c(1,1,0.5), c(2,2,1), window=square(3))
duplicated(X)
```

edges2triangles

List Triangles in a Graph

Description

Given a list of edges between vertices, compile a list of all triangles formed by these edges.

Usage

```
edges2triangles(iedge, jedge, nvert=max(iedge, jedge), ...,
                check=TRUE, friendly=rep(TRUE, nvert))
```

Arguments

<code>iedge, jedge</code>	Integer vectors, of equal length, specifying the edges.
<code>nvert</code>	Number of vertices in the network.
<code>...</code>	Ignored
<code>check</code>	Logical. Whether to check validity of input data.
<code>friendly</code>	Optional. For advanced use. See Details.

Details

This low level function finds all the triangles (cliques of size 3) in a finite graph with `nvert` vertices and with edges specified by `iedge, jedge`.

The interpretation of `iedge, jedge` is that each successive pair of entries specifies an edge in the graph. The k th edge joins vertex `iedge[k]` to vertex `jedge[k]`. Entries of `iedge` and `jedge` must be integers from 1 to `nvert`.

To improve efficiency in some applications, the optional argument `friendly` can be used. It should be a logical vector of length `nvert` specifying a labelling of the vertices, such that two vertices j, k which are *not* friendly (`friendly[j] = friendly[k] = FALSE`) are *never* connected by an edge.

Value

A 3-column matrix of integers, in which each row represents a triangle.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

Examples

```
i <- c(1, 2, 5, 5, 1, 4, 2)
j <- c(2, 3, 3, 1, 3, 2, 5)
edges2triangles(i, j)
```

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, 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. |

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’

$$m[i] = 1/\lambda - \hat{\lambda}(x[i], X)$$

where $\lambda - \hat{\lambda}(x[i], X)$ 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"). Such objects are produced by the maximum pseudolikelihood fitting algorithm ppm. 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 data.ppm.

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

References

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

See Also

[diagnose.ppm](#), [ppm.object](#), [data.ppm](#), [residuals.ppm](#), [ppm](#)

Examples

```
data(cells)
fit <- ppm(cells, ~x, Strauss(r=0.15))
ee <- eem(fit)
sum(ee)/area.owin(cells$window) # 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)
```

Arguments

model	A fitted point process model (object of class "ppm").
covname	The name of the covariate. A character string.
...	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.

Details

The object model should be an object of class "ppm" 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.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`ppm`, `predict.ppm`, `fv.object`

Examples

```
data(copper)
X <- copper$SouthPoints
D <- distmap(copper$SouthLines)
fit <- ppm(X, ~polynom(Z, 5), covariates=list(Z=D))

plot(effectfun(fit, "Z"))

plot(effectfun(fit, "Z", se.fit=TRUE), shade=c("hi", "lo"))
fit <- ppm(X, ~x + polynom(Z, 5), covariates=list(Z=D))
plot(effectfun(fit, "Z", x=20))
fit <- ppm(X, ~x)
plot(effectfun(fit, "x"))
```

Description

Estimate the summary functions $E(r)$ and $V(r)$ for a marked point pattern, proposed by Schlather et al (2004) as diagnostics for dependence between the points and the marks.

Usage

```
Emark(X, r=NULL,
       correction=c("isotropic", "Ripley", "translate"),
       method="density", ..., normalise=FALSE)
Vmark(X, r=NULL,
       correction=c("isotropic", "Ripley", "translate"),
       method="density", ..., normalise=FALSE)
```

Arguments

<code>X</code>	The observed point pattern. An object of class "ppp" or something acceptable to as.ppp . The pattern should have numeric marks.
<code>r</code>	Optional. Numeric vector. The values of the argument r at which the function $E(r)$ or $V(r)$ should be evaluated. There is a sensible default.
<code>correction</code>	A character vector containing any selection of the options "isotropic", "Ripley" or "translate". It specifies the edge correction(s) to be applied.
<code>method</code>	A character vector indicating the user's choice of density estimation technique to be used. Options are "density", "loess", "sm" and "smrep".
<code>...</code>	Arguments passed to the density estimation routine (density , loess or sm.density) selected by <code>method</code> .
<code>normalise</code>	If TRUE, normalise the estimate of $E(r)$ or $V(r)$ so that it would have value equal to 1 if the marks are independent of the points.

Details

For a marked point process, Schlather et al (2004) defined the functions $E(r)$ and $V(r)$ to be the conditional mean and conditional variance of the mark attached to a typical random point, given that there exists another random point at a distance r away from it.

More formally,

$$E(r) = E_{0u}[M(0)]$$

and

$$V(r) = E_{0u}[(M(0) - E(u))^2]$$

where E_{0u} denotes the conditional expectation given that there are points of the process at the locations 0 and u separated by a distance r , and where $M(0)$ denotes the mark attached to the point 0.

These functions may serve as diagnostics for dependence between the points and the marks. If the points and marks are independent, then $E(r)$ and $V(r)$ should be constant (not depending on r). See Schlather et al (2004).

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to `as.ppp`. It must be a marked point pattern with numeric marks.

The argument r is the vector of values for the distance r at which $k_f(r)$ is estimated.

This algorithm assumes that X can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in X as $X>window$) may have arbitrary shape.

Biases due to edge effects are treated in the same manner as in `Kest`. The edge corrections implemented here are

isotropic/Ripley Ripley's isotropic correction (see Ripley, 1988; Ohser, 1983). This is implemented only for rectangular and polygonal windows (not for binary masks).

translate Translation correction (Ohser, 1983). Implemented for all window geometries, but slow for complex windows.

Note that the estimator assumes the process is stationary (spatially homogeneous).

The numerator and denominator of the mark correlation function (in the expression above) are estimated using density estimation techniques. The user can choose between

"density" which uses the standard kernel density estimation routine `density`, and works only for evenly-spaced r values;

"loess" which uses the function `loess` in the package `modreg`;

"sm" which uses the function `sm.density` in the package `sm` and is extremely slow;

"smrep" which uses the function `sm.density` in the package `sm` and is relatively fast, but may require manual control of the smoothing parameter `hmult`.

Value

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

Essentially a data frame containing numeric columns

r the values of the argument r at which the function $E(r)$ or $V(r)$ has been estimated

theo the theoretical, constant value of $E(r)$ or $V(r)$ when the marks attached to different points are independent

together with a column or columns named "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $E(r)$ or $V(r)$ obtained by the edge corrections named.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

Schlather, M. and Ribeiro, P. and Diggle, P. (2004) Detecting dependence between marks and locations of marked point processes. *Journal of the Royal Statistical Society, series B* **66** (2004) 79-83.

See Also

Mark correlation [markcorr](#), mark variogram [markvario](#) for numeric marks.
 Mark connection function [markconnect](#) and multitype K-functions [Kcross](#), [Kdot](#) for factor-valued marks.

Examples

```
data(spruces)

plot(Emark(spruces))
E <- Emark(spruces, method="density", kernel="epanechnikov")
plot(Vmark(spruces))
```

`endpoints.psp`

Endpoints of Line Segment Pattern

Description

Extracts the endpoints of each line segment in a line segment pattern.

Usage

```
endpoints.psp(x, which="both")
```

Arguments

- | | |
|--------------------|--|
| <code>x</code> | A line segment pattern (object of class "psp"). |
| <code>which</code> | String specifying which endpoint or endpoints should be returned. See Details. |

Details

This function extracts one endpoint, or both endpoints, from each of the line segments in `x`, and returns these points as a point pattern object.

The argument `which` determines which endpoint or endpoints of each line segment should be returned:

`which="both"` (the default): both endpoints of each line segment are returned. The result is a point pattern with twice as many points as there are line segments in `x`.

`which="first"` select the first endpoint of each line segment (returns the points with coordinates `x$ends$x0, x$ends$y0`).

`which="second"` select the second endpoint of each line segment (returns the points with coordinates `x$ends$x1, x$ends$y1`).

`which="left"` select the left-most endpoint (the endpoint with the smaller *x* coordinate) of each line segment.

`which="right"` select the right-most endpoint (the endpoint with the greater *x* coordinate) of each line segment.

`which="lower"` select the lower endpoint (the endpoint with the smaller *y* coordinate) of each line segment.

`which="upper"` select the upper endpoint (the endpoint with the greater y coordinate) of each line segment.

The result is a point pattern. It also has an attribute "id" which is an integer vector identifying the segment which contributed each point.

Value

Point pattern (object of class "ppp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`psp.object`, `ppp.object`, `midpoints.psp`

Examples

```
a <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
plot(a)
b <- endpoints.psp(a, "left")
plot(b, add=TRUE)
```

Description

Computes simulation envelopes of a summary function.

Usage

```
envelope(Y, fun, ...)

## S3 method for class 'ppp'
envelope(Y, fun=Kest, nsim=99, nrank=1, ...,
simulate=NULL, verbose=TRUE, clipdata=TRUE,
transform=NULL, global=FALSE, ginterval=NULL,
savefuns=FALSE, savepatterns=FALSE,
nsim2=nsim, VARIANCE=FALSE, nSD=2, Yname=NULL, maxnerr=nsim)

## S3 method for class 'ppm'
envelope(Y, fun=Kest, nsim=99, nrank=1, ...,
simulate=NULL, verbose=TRUE, clipdata=TRUE,
start=NULL, control=update(default.rmhcontrol(Y), nrep=nrep), nrep=1e5,
transform=NULL, global=FALSE, ginterval=NULL,
savefuns=FALSE, savepatterns=FALSE,
nsim2=nsim, VARIANCE=FALSE, nSD=2, Yname=NULL, maxnerr=nsim)
```

```
## S3 method for class 'kppm'
envelope(Y, fun=Kest, nsim=99, nrank=1, ...,
simulate=NULL, verbose=TRUE, clipdata=TRUE,
transform=NULL, global=FALSE, ginterval=NULL,
savefuns=FALSE, savepatterns=FALSE,
nsim2=nsim, VARIANCE=FALSE, nSD=2, Yname=NULL, maxnerr=nsim)
```

Arguments

Y	Object containing point pattern data. A point pattern (object of class "ppp") or a fitted point process model (object of class "ppm" or "kppm").
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 <code>nsim</code> simulated values. A rank of 1 means that the minimum and maximum simulated values will be used.
...	Extra arguments passed to <code>fun</code> .
simulate	Optional. Specifies how to generate the simulated point patterns. If <code>simulate</code> is an expression in the R language, then this expression will be evaluated <code>nsim</code> times, to obtain <code>nsim</code> point patterns which are taken as the simulated patterns from which the envelopes are computed. If <code>simulate</code> 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 <code>simulate</code> may be an object produced by the <code>envelope</code> command: see Details.
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 <code>start</code> and <code>control</code> of <code>rmh</code> , giving complete control over the simulation algorithm. Applicable only when <code>Y</code> is a fitted model of class "ppm".
nrep	Number of iterations in the Metropolis-Hastings simulation algorithm. Applicable only when <code>Y</code> 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 (<code>global=FALSE</code>) or simultaneous (<code>global=TRUE</code>).
ginterval	Optional. A vector of length 2 specifying the interval of r values for the simultaneous critical envelopes. Only relevant if <code>global=TRUE</code> .
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 <code>global=TRUE</code> and the simulations are not based on CSR.
VARIANCE	Logical. If TRUE, critical envelopes will be calculated as sample mean plus or minus <code>nSD</code> 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 an 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.

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" and "kppm" described here. There is also a method for the class "pp3" which is described separately as [envelope.pp3](#).

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") 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 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 makes it possible to select the edge correction used to calculate the summary statistic. See the Examples. Selecting only a single edge correction will make the code run much faster.

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. It must be an expression object using the symbol `.` to represent the function value. 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))`. Such transforms are only useful if `global=TRUE`.

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 "fv", see [fv.object](#), which can be printed and plotted directly.

Essentially a data frame containing columns

<code>r</code>	the vector of values of the argument r at which the summary function <code>fun</code> has been estimated
<code>obs</code>	values of the summary function for the data point pattern
<code>lo</code>	lower envelope of simulations
<code>hi</code>	upper envelope of simulations

and either

<code>theo</code>	theoretical value of the summary function under CSR (Complete Spatial Randomness, a uniform Poisson point process) if the simulations were generated according to CSR
<code>mmean</code>	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 mean of the data pattern, use [varblock](#).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.
- Diggle, P.J. *Statistical analysis of spatial point patterns*. 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

[clf.test](#), [mad.test](#) for envelope-based tests.
[fv.object](#), [plot.envelope](#), [plot.fv](#), [envelope.envelope](#), [pool.envelope](#) for handling envelopes.
[Kest](#), [Gest](#), [Fest](#), [Jest](#), [pcf](#), [ppp](#), [ppm](#), [default.expand](#)

Examples

```
data(simdat)
X <- simdat

# Envelope of K function under CSR
## Not run:
plot(envelope(X))

## End(Not run)

# Translation edge correction (this is also FASTER):
## Not run:
plot(envelope(X, correction="translate"))

## End(Not run)

# Envelope of K function for simulations from Gibbs model
data(cells)
fit <- ppm(cells, ~1, Strauss(0.05))
## Not run:
plot(envelope(fit))
plot(envelope(fit), global=TRUE)

## End(Not run)
```

```

# Envelope of K function for simulations from cluster model
data(redwood)
fit <- kppm(redwood, ~1, "Thomas")
## Not run:
plot(envelope(fit, Gest))
plot(envelope(fit, Gest, global=TRUE))

## End(Not run)

# Envelope of G function under CSR
## Not run:
plot(envelope(X, Gest))

## End(Not run)

# Envelope of L function under CSR
# L(r) = sqrt(K(r)/pi)
## Not run:
E <- envelope(X, Kest)
plot(E, sqrt(./pi) ~ r)

## End(Not run)

# Simultaneous critical envelope for L function
# (alternatively, use Lest)
## Not run:
plot(envelope(X, Kest, transform=expression(sqrt(./pi)), global=TRUE))

## End(Not run)

# How to pass arguments needed to compute the summary functions:
# We want envelopes for Jcross(X, "A", "B")
# where "A" and "B" are types of points in the dataset 'demopat'

data(demopat)
## Not run:
plot(envelope(demopat, Jcross, i="A", j="B"))

## End(Not run)

# Use of 'simulate'
## Not run:
plot(envelope(cells, Gest, simulate=expression(runifpoint(42))))
plot(envelope(cells, Gest, simulate=expression(rMaternI(100, 0.02)))))

## End(Not run)

# Envelope under random toroidal shifts
data(amacrine)
## Not run:

```

```

plot(envelope(amacrine, Kcross, i="on", j="off",
              simulate=expression(rshift(amacrine, radius=0.25)))))

## End(Not run)

# Envelope under random shifts with erosion
## Not run:
plot(envelope(amacrine, Kcross, i="on", j="off",
              simulate=expression(rshift(amacrine, radius=0.1, edge="erode"))))

## End(Not run)

# Envelope of INHOMOGENEOUS K-function with fitted trend
## Not run:
trend <- density.ppp(X, 1.5)
plot(envelope(X, Kinhom, lambda=trend,
              simulate=expression(rpoispp(trend)))))

## End(Not run)

# Precomputed list of point patterns
X <- rpoispp(30)
PatList <- list()
for(i in 1:19) PatList[[i]] <- runifpoint(npoints(X))
E <- envelope(X, Kest, nsim=19, simulate=PatList)
if(interactive()) plot(E)

# re-using the same point patterns
EK <- envelope(X, Kest, nsim=10, savepatterns=TRUE)
EG <- envelope(X, Kest, nsim=10, simulate=EK)

```

envelope.envelope *Recompute Envelopes*

Description

Given a simulation envelope (object of class "envelope"), compute another envelope from the same simulation data using different parameters.

Usage

```

## S3 method for class 'envelope'
envelope(Y, fun = NULL, ...)

```

Arguments

Y	A simulation envelope (object of class "envelope").
fun	Optional. Summary function to be applied to the simulated point patterns.
...	Parameters controlling the type of envelope that is re-computed. See envelope .

Details

This function can be used to re-compute a simulation envelope from previously simulated data, using different parameter settings for the envelope: for example, a different significance level, or a global envelope instead of a pointwise envelope.

The function `envelope` is generic. This is the method for the class "envelope".

The argument `Y` should be a simulation envelope (object of class "envelope") produced by any of the methods for `envelope`. Additionally, `Y` must contain either

- the simulated point patterns that were used to create the original envelope (so `Y` should have been created by calling `envelope` with `savepatterns=TRUE`);
- the summary functions of the simulated point patterns that were used to create the original envelope (so `Y` should have been created by calling `envelope` with `savefuns=TRUE`).

If the argument `fun` is given, it should be a summary function that can be applied to the simulated point patterns that were used to create `Y`. The envelope of the summary function `fun` for these point patterns will be computed using the parameters specified in

If `fun` is not given, then:

- If `Y` contains the summary functions that were used to compute the original envelope, then the new envelope will be computed from these original summary functions.
- Otherwise, if `Y` contains the simulated point patterns, then the `K` function `Kest` will be applied to each of these simulated point patterns, and the new envelope will be based on the `K` functions.

The new envelope will be computed using the parameters specified in

See `envelope` for a full list of envelope parameters. Frequently-used parameters include `nrank` and `nsim` (to change the number of simulations used and the significance level of the envelope), `global` (to change from pointwise to global envelopes) and `VARIANCE` (to compute the envelopes from the sample moments instead of the ranks).

Value

An envelope (object of class "envelope").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`envelope`

Examples

```
data(cells)
E <- envelope(cells, Kest, nsim=19, savefuns=TRUE, savepatterns=TRUE)
E2 <- envelope(E, nrank=2)
Eg <- envelope(E, global=TRUE)
EG <- envelope(E, Gest)
```

envelope.lpp

Envelope for Point Patterns on Linear Network

Description

Enables envelopes to be computed for point patterns on a linear network.

Usage

```
## S3 method for class 'lpp'
envelope(Y, fun=linearK, nsim=99, nrank=1, ...,
simulate=NULL, verbose=TRUE,
transform=NULL, global=FALSE, ginterval=NULL,
savefuns=FALSE, savepatterns=FALSE,
nsim2=nsim, VARIANCE=FALSE, nSD=2, Yname=NULL)
## S3 method for class 'lppm'
envelope(Y, fun=linearK, nsim=99, nrank=1, ...,
simulate=NULL, verbose=TRUE,
transform=NULL, global=FALSE, ginterval=NULL,
savefuns=FALSE, savepatterns=FALSE,
nsim2=nsim, VARIANCE=FALSE, nSD=2, Yname=NULL)
```

Arguments

<code>Y</code>	A point pattern on a linear network (object of class "lpp") or a fitted point process model on a linear network (object of class "lppm").
<code>fun</code>	Function that is to be computed for each simulated pattern.
<code>nsim</code>	Number of simulations to perform.
<code>nrank</code>	Integer. Rank of the envelope value amongst the <code>nsim</code> simulated values. A rank of 1 means that the minimum and maximum simulated values will be used.
<code>...</code>	Extra arguments passed to <code>fun</code> .
<code>simulate</code>	Optional. Specifies how to generate the simulated point patterns. If <code>simulate</code> is an expression in the R language, then this expression will be evaluated <code>nsim</code> times, to obtain <code>nsim</code> point patterns which are taken as the simulated patterns from which the envelopes are computed. If <code>simulate</code> 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 <code>simulate</code> may be an object produced by the <code>envelope</code> command: see Details.
<code>verbose</code>	Logical flag indicating whether to print progress reports during the simulations.
<code>transform</code>	Optional. A transformation to be applied to the function values, before the envelopes are computed. An expression object (see Details).
<code>global</code>	Logical flag indicating whether envelopes should be pointwise (<code>global=FALSE</code>) or simultaneous (<code>global=TRUE</code>).
<code>ginterval</code>	Optional. A vector of length 2 specifying the interval of r values for the simultaneous critical envelopes. Only relevant if <code>global=TRUE</code> .
<code>savefuns</code>	Logical flag indicating whether to save all the simulated function values.
<code>savepatterns</code>	Logical flag indicating whether to save all the simulated point patterns.

<code>nsim2</code>	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 <code>global=TRUE</code> and the simulations are not based on CSR.
<code>VARIANCE</code>	Logical. If <code>TRUE</code> , critical envelopes will be calculated as sample mean plus or minus <code>nSD</code> times sample standard deviation.
<code>nSD</code>	Number of estimated standard deviations used to determine the critical envelopes, if <code>VARIANCE=TRUE</code> .
<code>Yname</code>	Character string that should be used as the name of the data point pattern <code>Y</code> when printing or plotting the results.

Details

This is a method for the generic function `envelope` applicable to point patterns on a linear network. The argument `Y` can be either a point pattern on a linear network, or a fitted point process model on a linear network. The function `fun` will be evaluated for the data and also for `nsim` simulated point patterns on the same linear network. The upper and lower envelopes of these evaluated functions will be computed as described in `envelope`.

The type of simulation is determined as follows.

- if `Y` is a point pattern (object of class "lpp") and `simulate` is missing or `NULL`, then random point patterns will be generated according to a Poisson point process on the linear network on which `Y` is defined, with intensity estimated from `Y`.
- if `Y` is a fitted point process model (object of class "lppm") and `simulate` is missing or `NULL`, then random point patterns will be generated by simulating from the fitted model.
- If `simulate` is present, it should be an expression that can be evaluated to yield random point patterns on the same linear network as `Y`.

The function `fun` should accept as its first argument a point pattern on a linear network (object of class "lpp") and should have another argument called `r` or a ... argument.

Value

Function value table (object of class "fv") with additional information, as described in `envelope`.

Author(s)

Ang Qi Wei <aqw07398@hotmail.com> and Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

References

- Ang, Q.W. (2010) *Statistical methodology for events on a network*. Master's thesis, School of Mathematics and Statistics, University of Western Australia.
- Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. To appear in *Scandinavian Journal of Statistics*.
- Okabe, A. and Yamada, I. (2001) The K-function method on a network and its computational implementation. *Geographical Analysis* **33**, 271-290.

See Also

`envelope`, `linearK`

Examples

```
example(lpp)
# uniform Poisson
envelope(X, nsim=4)
# nonuniform Poisson
fit <- lppm(X, ~x)
envelope(fit, nsim=4)
```

envelope.pp3

Simulation Envelopes of Summary Function for 3D Point Pattern

Description

Computes simulation envelopes of a summary function for a three-dimensional point pattern.

Usage

```
## S3 method for class 'pp3'
envelope(Y, fun=K3est, nsim=99, nrank=1, ...,
simulate=NULL, verbose=TRUE,
transform=NULL, global=FALSE, ginterval=NULL,
savefuns=FALSE, savepatterns=FALSE,
nsim2=nsim, VARIANCE=FALSE, nSD=2, Yname=NULL, maxnerr=nsim)
```

Arguments

<code>Y</code>	A three-dimensional point pattern (object of class "pp3").
<code>fun</code>	Function that computes the desired summary statistic for a 3D point pattern.
<code>nsim</code>	Number of simulated point patterns to be generated when computing the envelopes.
<code>nrank</code>	Integer. Rank of the envelope value amongst the <code>nsim</code> simulated values. A rank of 1 means that the minimum and maximum simulated values will be used.
<code>...</code>	Extra arguments passed to <code>fun</code> .
<code>simulate</code>	Optional. Specifies how to generate the simulated point patterns. If <code>simulate</code> is an expression in the R language, then this expression will be evaluated <code>nsim</code> times, to obtain <code>nsim</code> point patterns which are taken as the simulated patterns from which the envelopes are computed. If <code>simulate</code> 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 <code>simulate</code> may be an object produced by the <code>envelope</code> command: see Details.
<code>verbose</code>	Logical flag indicating whether to print progress reports during the simulations.
<code>transform</code>	Optional. A transformation to be applied to the function values, before the envelopes are computed. An expression object (see Details).
<code>global</code>	Logical flag indicating whether envelopes should be pointwise (<code>global=FALSE</code>) or simultaneous (<code>global=TRUE</code>).
<code>ginterval</code>	Optional. A vector of length 2 specifying the interval of r values for the simultaneous critical envelopes. Only relevant if <code>global=TRUE</code> .
<code>savefuns</code>	Logical flag indicating whether to save all the simulated function values.

<code>savepatterns</code>	Logical flag indicating whether to save all the simulated point patterns.
<code>nsim2</code>	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 <code>global=TRUE</code> and the simulations are not based on CSR.
<code>VARIANCE</code>	Logical. If <code>TRUE</code> , critical envelopes will be calculated as sample mean plus or minus <code>nSD</code> times sample standard deviation.
<code>nSD</code>	Number of estimated standard deviations used to determine the critical envelopes, if <code>VARIANCE=TRUE</code> .
<code>Yname</code>	Character string that should be used as the name of the data point pattern <code>Y</code> when printing or plotting the results.
<code>maxnerr</code>	Maximum number of rejected patterns. If <code>fun</code> yields an error when applied to a simulated point pattern (for example, because the pattern is empty and <code>fun</code> requires at least one point), the pattern will be rejected and a new random point pattern will be generated. If this happens more than <code>maxnerr</code> times, the algorithm will give up.

Details

The `envelope` command performs simulations and computes envelopes of a summary statistic based on the simulations. The result is an object that can be plotted to display the envelopes. The envelopes can be used to assess the goodness-of-fit of a point process model to point pattern data.

The `envelope` function is generic, with methods for the classes "`ppp`", "`ppm`" and "`kppm`" described in the help file for [envelope](#). This function `envelope.pp3` is the method for three-dimensional point patterns (objects of class "`pp3`").

For the most basic use, if you have a 3D point pattern `X` and you want to test Complete Spatial Randomness (CSR), type `plot(envelope(X, K3est, nsim=39))` to see the three-dimensional K function for `X` plotted together with the envelopes of the three-dimensional K function for 39 simulations of CSR.

To create simulation envelopes, the command `envelope(Y, ...)` first generates `nsim` random point patterns in one of the following ways.

- If `simulate=NULL`, then we generate `nsim` simulations of Complete Spatial Randomness (i.e. `nsim` simulated point patterns each being a realisation of the uniform Poisson point process) with the same intensity as the pattern `Y`.
- If `simulate` is supplied, then it determines how the simulated point patterns are generated. See [envelope](#) for details.

The summary statistic `fun` is applied to each of these simulated patterns. Typically `fun` is one of the functions `K3est`, `G3est`, `F3est` or `pcf3est`. It may also be a character string containing the name of one of these functions.

For further information, see the documentation for [envelope](#).

Value

A function value table (object of class "`fv`") which can be plotted directly. See [envelope](#) for further details.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A.J., Moyeed, R.A., Howard, C.V. and Boyde, A. (1993) Analysis of a three-dimensional point pattern with replication. *Applied Statistics* **42**, 641–668.

See Also

[pp3](#), [rpoispp3](#), [K3est](#), [G3est](#), [F3est](#), [pcf3est](#).

Examples

```
X <- rpoispp3(20, box3())
## Not run:
plot(envelope(X, nsim=39))

## End(Not run)
```

eroded.areas

Areas of Morphological Erosions

Description

Computes the areas of successive morphological erosions of a window.

Usage

```
eroded.areas(w, r)
```

Arguments

- | | |
|---|--|
| w | A window. |
| r | Numeric vector of radii at which erosions will be performed. |

Details

This function computes the areas of the erosions of the window *w* by each of the radii *r[i]*.

The morphological erosion of a set *W* by a distance *r* > 0 is the subset consisting of all points *x* ∈ *W* such that the distance from *x* to the boundary of *W* is greater than or equal to *r*. In other words it is the result of trimming a margin of width *r* off the set *W*.

The argument *r* should be a vector of positive numbers. The argument *w* should be a window (an object of class "owin", see [owin.object](#) for details) or can be given in any format acceptable to [as.owin\(\)](#).

Unless *w* is a rectangle, the computation is performed using a pixel raster approximation.

To compute the eroded window itself, use [erosion](#).

Value

Numeric vector, of the same length as *r*, giving the areas of the successive erosions.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[owin](#), [as.owin](#), [erosion](#)

Examples

```
w <- owin(c(0,1),c(0,1))
a <- eroded.areas(w, seq(0.01,0.49,by=0.01))
```

erosion

Morphological Erosion

Description

Perform morphological erosion of a window, a line segment pattern or a point pattern.

Usage

```
erosion(w, r, ...)
## S3 method for class 'owin'
erosion(w, r, shrink.frame=TRUE, ...,
        strict=FALSE, polygonal=NULL)
## S3 method for class 'ppp'
erosion(w, r,...)
## S3 method for class 'psp'
erosion(w, r,...)
```

Arguments

- | | |
|---------------------------|---|
| <code>w</code> | A window (object of class "owin" or a line segment pattern (object of class "psp") or a point pattern (object of class "ppp"). |
| <code>r</code> | positive number: the radius of erosion. |
| <code>shrink.frame</code> | logical: if TRUE, erode the bounding rectangle as well. |
| <code>...</code> | extra arguments to as.mask controlling the pixel resolution, if pixel approximation is used. |
| <code>strict</code> | Logical flag determining the fate of boundary pixels, if pixel approximation is used. See details. |
| <code>polygonal</code> | Logical flag indicating whether to compute a polygonal approximation to the erosion (polygonal=TRUE) or a pixel grid approximation (polygonal=FALSE). |

Details

The morphological erosion of a set W by a distance $r > 0$ is the subset consisting of all points $x \in W$ such that the distance from x to the boundary of W is greater than or equal to r . In other words it is the result of trimming a margin of width r off the set W .

If `polygonal=TRUE` then a polygonal approximation to the erosion is computed. If `polygonal=FALSE` then a pixel approximation to the erosion is computed from the distance map of `w`. The arguments "... " are passed to `as.mask` to control the pixel resolution. The erosion consists of all pixels whose distance from the boundary of `w` is strictly greater than `r` (if `strict=TRUE`) or is greater than or equal to `r` (if `strict=FALSE`).

When `w` is a window, the default (when `polygonal=NULL`) is to compute a polygonal approximation if `w` is a rectangle or polygonal window, and to compute a pixel approximation if `w` is a window of type "mask".

If `shrink.frame` is false, the resulting window is given the same outer, bounding rectangle as the original window `w`. If `shrink.frame` is true, the original bounding rectangle is also eroded by the same distance `r`.

To simply compute the area of the eroded window, use `eroded.areas`.

Value

If $r > 0$, an object of class "owin" representing the eroded region (or NULL if this region is empty). If $r=0$, the result is identical to `w`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`dilation` for the opposite operation.

`owin`, `as.owin`, `eroded.areas`

Examples

```
w <- owin(c(0,1),c(0,1))
v <- erosion(w, 0.1)
# returns rectangle [0.1, 0.9] x [0.1,0.9]
## Not run:
v <- erosion(w, 0.6)
# erosion is empty

## End(Not run)
```

Description

Evaluates any expression involving one or more function arrays (fasp objects) and returns another function array.

Usage

```
eval.fasp(expr, envir, dotonly=TRUE)
```

Arguments

<code>expr</code>	An expression.
<code>envir</code>	Optional. The environment in which to evaluate the expression.
<code>dotonly</code>	Logical. Passed to eval.fv .

Details

This is a wrapper to make it easier to perform pointwise calculations with the arrays of summary functions used in spatial statistics.

A function array (object of class "fasp") can be regarded as a matrix whose entries are functions. Objects of this kind are returned by the command [alltypes](#).

Suppose X is an object of class "fasp". Then `eval.fasp(X+3)` effectively adds 3 to the value of every function in the array X , and returns the resulting object.

Suppose X and Y are two objects of class "fasp" which are compatible (for example the arrays must have the same dimensions). Then `eval.im(X + Y)` will add the corresponding functions in each cell of the arrays X and Y , and return the resulting array of functions.

In general, `expr` can be any expression involving (a) the *names* of objects of class "fasp", (b) scalar constants, and (c) functions which are vectorised. See the Examples.

First `eval.fasp` determines which of the *variable names* in the expression `expr` refer to objects of class "fasp". The expression is then evaluated for each cell of the array using [eval.fv](#).

The expression `expr` must be vectorised. There must be at least one object of class "fasp" in the expression. All such objects must be compatible.

Value

Another object of class "fasp".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[fasp.object](#), [Kest](#)

Examples

```
# manipulating the K function
data(amacrine)
K <- alltypes(amacrine, "K")

# expressions involving a fasp object
eval.fasp(K + 3)
L <- eval.fasp(sqrt(K/pi))

# expression involving two fasp objects
eval.fasp(K - L)
```

eval.fv*Evaluate Expression Involving Functions*

Description

Evaluates any expression involving one or more function value (fv) objects, and returns another object of the same kind.

Usage

```
eval.fv(expr, envir, dotonly=TRUE)
```

Arguments

expr	An expression.
envir	Optional. The environment in which to evaluate the expression.
dotonly	Logical. See Details.

Details

This is a wrapper to make it easier to perform pointwise calculations with the summary functions used in spatial statistics.

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

For example, suppose X is an object of class "fv" containing several different estimates of the Ripley's K function $K(r)$, evaluated at a sequence of values of r . Then `eval.fv(X+3)` effectively adds 3 to each function estimate in X , and returns the resulting object.

Suppose X and Y are two objects of class "fv" which are compatible (in particular they have the same vector of r values). Then `eval.im(X + Y)` will add the corresponding function values in X and Y , and return the resulting function.

In general, `expr` can be any expression involving (a) the *names* of objects of class "fv", (b) scalar constants, and (c) functions which are vectorised. See the Examples.

First `eval.fv` determines which of the *variable names* in the expression `expr` refer to objects of class "fv". Each such name is replaced by a vector containing the function values. The expression is then evaluated. The result should be a vector; it is taken as the new vector of function values.

The expression `expr` must be vectorised. There must be at least one object of class "fv" in the expression. All such objects must be compatible.

If `dotonly=TRUE` (the default), the expression will be evaluated only for those columns of an "fv" object that contain values of the function itself (rather than values of the derivative of the function, the hazard rate, etc). If `dotonly=FALSE`, the expression will be evaluated for all columns.

For example the result of [Fest](#) includes several columns containing estimates of the empty space function $F(r)$, but also includes an estimate of the hazard $h(r)$ of $F(r)$. Transformations that are valid for F may not be valid for h . Accordingly, h would normally be omitted from the calculation.

The columns of an object x that represent the function itself are identified by its "dot" names, `fvnames(x, ".")`. They are the columns normally plotted by [plot.fv](#) and identified by the symbol " $.$ " in plot formulas in [plot.fv](#).

Value

Another object of class "fv".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[fv.object](#), [Kest](#)

Examples

```
# manipulating the K function
X <- rpoispp(42)
Ks <- Kest(X)

eval.fv(Ks + 3)
Ls <- eval.fv(sqrt(Ks/pi))

# manipulating two K functions
Y <- rpoispp(20)
Kr <- Kest(Y)

Kdif <- eval.fv(Ks - Kr)
Z <- eval.fv(sqrt(Ks/pi) - sqrt(Kr/pi))
```

eval.im

Evaluate Expression Involving Pixel Images

Description

Evaluates any expression involving one or more pixel images, and returns a pixel image.

Usage

```
eval.im(expr, envir, harmonize=TRUE)
```

Arguments

<code>expr</code>	An expression.
<code>envir</code>	Optional. The environment in which to evaluate the expression.
<code>harmonize</code>	Logical. Whether to resolve inconsistencies between the pixel grids.

Details

This function is a wrapper to make it easier to perform pixel-by-pixel calculations in an image.

Pixel images in **spatstat** are represented by objects of class "im" (see [im.object](#)). These are essentially matrices of pixel values, with extra attributes recording the pixel dimensions, etc.

Suppose X is a pixel image. Then eval.im($X+3$) will add 3 to the value of every pixel in X , and return the resulting pixel image.

Suppose X and Y are two pixel images with compatible dimensions: they have the same number of pixels, the same physical size of pixels, and the same bounding box. Then eval.im($X + Y$) will add the corresponding pixel values in X and Y , and return the resulting pixel image.

In general, expr can be any expression in the R language involving (a) the *names* of pixel images, (b) scalar constants, and (c) functions which are vectorised. See the Examples.

First eval.im determines which of the *variable names* in the expression expr refer to pixel images. Each such name is replaced by a matrix containing the pixel values. The expression is then evaluated. The result should be a matrix; it is taken as the matrix of pixel values.

The expression expr must be vectorised. There must be at least one pixel image in the expression.

All images must have compatible dimensions. If $\text{harmonize}=\text{TRUE}$, images that have incompatible dimensions will be resampled so that they are compatible. If $\text{harmonize}=\text{FALSE}$, images that are incompatible will cause an error.

Value

An image object of class "im".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[as.im](#), [compatible.im](#), [harmonise.im](#), [im.object](#)

Examples

```
# test images
X <- as.im(function(x,y) { x^2 - y^2 }, unit.square())
Y <- as.im(function(x,y) { 3 * x + y }, unit.square())

eval.im(X + 3)
eval.im(X - Y)
eval.im(abs(X - Y))
Z <- eval.im(sin(X * pi) + Y)
```

ewcdf*Weighted Empirical Cumulative Distribution Function***Description**

Compute a weighted version of the empirical cumulative distribution function.

Usage

```
ewcdf(x, weights = rep(1/length(x), length(x)))
```

Arguments

- | | |
|----------------------|---|
| <code>x</code> | Numeric vector of observations. |
| <code>weights</code> | Numeric vector of non-negative weights for <code>x</code> . |

Details

This is a modification of the standard function `ecdf` allowing the observations `x` to have weights.

The weighted e.c.d.f. (empirical cumulative distribution function) F_n is defined so that, for any real number y , the value of $F_n(y)$ is equal to the total weight of all entries of `x` that are less than or equal to y . That is $F_n(y) = \text{sum}(weights[x \leq y])$.

Thus F_n is a step function which jumps at the values of `x`. The height of the jump at a point y is the total weight of all entries in `x` number of tied observations at that value. Missing values are ignored.

If `weights` is omitted, the default is equivalent to `ecdf(x)`.

Value

A function, of class "ecdf", inheriting from "stepfun".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

`ecdf`

Examples

```
x <- rnorm(100)
w <- runif(100)
plot(ewcdf(x,w))
```

exactMPEStrauss	<i>Exact Maximum Pseudolikelihood Estimate for Stationary Strauss Process</i>
-----------------	---

Description

Computes, to very high accuracy, the Maximum Pseudolikelihood Estimates of the parameters of a stationary Strauss point process.

Usage

```
exactMPEStrauss(X, R, ngrid = 2048, plotit = FALSE)
```

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.

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).

Value

Vector of length 2.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.
- Baddeley, A. and Turner, R. (2013) Manuscript in preparation.
- Ripley, B.D. (1988) *Statistical inference for spatial processes*. Cambridge University Press.

See Also

[ppm](#)

Examples

```
exactMLEStrauss(cells, 0.1)
coef(ppm(cells, ~1, Strauss(0.1)))
coef(ppm(cells, ~1, Strauss(0.1), nd=128))
```

[expand.owin](#)

Apply Expansion Rule

Description

Applies an expansion rule to a window.

Usage

```
expand.owin(W, ...)
```

Arguments

- | | |
|----------------|---|
| <code>W</code> | A window. |
| ... | Arguments passed to rmhexpand to determine an expansion rule. |

Details

The argument `W` should be a window (an object of class "owin").

This command applies the expansion rule specified by the arguments `...` to the window `W`, yielding another window.

The arguments `...` are passed to [rmhexpand](#) to determine the expansion rule.

For other transformations of the scale, location and orientation of a window, see [shift](#), [affine](#) and [rotate](#).

Value

A window (object of class "owin").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[rmhexpand](#) about expansion rules.
[shift](#), [rotate](#), [affine](#) for other types of manipulation.

Examples

```
expand.owin(square(1), 9)
expand.owin(square(1), distance=0.5)
expand.owin(letterR, length=2)
expand.owin(letterR, distance=0.1)
```

Extract.fasp

*Extract Subset of Function Array***Description**

Extract a subset of a function array (an object of class "fasp").

Usage

```
## S3 method for class 'fasp'
x[I, J, drop=TRUE, ...]
```

Arguments

x	A function array. An object of class "fasp".
I	any valid expression for a subset of the row indices of the array.
J	any valid expression for a subset of the column indices of the array.
drop	Logical. When the selected subset consists of only one cell of the array, if drop=FALSE the result is still returned as a 1×1 array of functions (class "fasp") while if drop=TRUE it is returned as a function (class "fv").
...	Ignored.

Details

A function array can be regarded as a matrix whose entries are functions. See [fasp.object](#) for an explanation of function arrays.

This routine extracts a sub-array according to the usual conventions for matrix indexing.

Value

A function array (of class "fasp"). Exceptionally, if the array has only one cell, and if drop=TRUE, then the result is a function value table (class "fv").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[fasp.object](#)

Examples

```
# Lansing woods data - multitype points with 6 types
data(lansing)

# compute 6 x 6 array of all cross-type K functions
a <- alltypes(lansing, "K")

# extract first three marks only
b <- a[1:3,1:3]
## Not run: plot(b)
# subset pertaining to hickories
h <- a[levels(lansing$marks) == "hickory", ]
## Not run: plot(h)
```

Extract.fv

Extract Subset of Function Values

Description

Extract a subset of an object of class "fv".

Usage

```
## S3 method for class 'fv'
x[i, j, ..., drop=FALSE]
```

Arguments

x	a function value object, of class "fv" (see fv.object). Essentially a data frame.
i	any appropriate subset index. Selects a subset of the rows of the data frame, i.e. a subset of the domain of the function(s) represented by x.
j	any appropriate subset index for the columns of the data frame. Selects some of the functions present in x.
drop, ...	Ignored.

Details

This is a method for "[" for the class "fv". It is very similar to [\[.data.frame](#) except for a few extra checks on the sanity of the result.

Value

A function value object (of class "fv").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also[fv.object](#)**Examples**

```

data(cells)
K <- Kest(cells)

# discard the estimates of K(r) for r > 0.1
Ksub <- K[K$r <= 0.1, ]

# discard the border method estimator
Ksub <- K[ , names(K) != "border"]

```

Extract.im

*Extract Subset of Image***Description**

Extract a subset or subregion of a pixel image.

Usage

```
## S3 method for class 'im'
x[i, drop=TRUE, ..., raster=NULL]
```

Arguments

x	A two-dimensional pixel image. An object of class "im".
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 a point pattern (an object of class "ppp"), or something that can be converted to a point pattern by as.ppp , or any type of index that applies to a matrix.
...	Ignored.
drop	Logical value. Locations in w that lie outside the spatial domain of the image x return a pixel value of NA if drop=FALSE, and are omitted if drop=TRUE.
raster	Optional. An object of class "owin" or "im" determining a pixel grid.

Details

This function extracts a subset of the pixel values in a pixel image. (To reassign the pixel values, see [\[<- .im\]](#)).

The image x must be an object of class "im" representing a pixel image defined inside a rectangle in two-dimensional space (see [im.object](#)).

The subset to be extracted is determined by the argument i. If i is a spatial window (an object of class "owin"), the values of the image inside this window are extracted (after first clipping the window to the spatial domain of the image if necessary). If i is a pixel image with logical values, it is interpreted as a spatial window (with TRUE values inside the window and FALSE outside). If i

is a point pattern (an object of class "ppp"), then the values of the pixel image at the points of this pattern are extracted.

At locations outside the spatial domain of the image, the pixel value is undefined, and is taken to be NA. The logical argument `drop` determines whether such NA values will be returned or omitted. It also influences the format of the return value.

If `i` is a point pattern (or something that can be converted to a point pattern by `as.ppp` such as a list of `x,y` coordinates), then `X[i, drop=FALSE]` is a numeric vector containing the pixel values at each of the points of the pattern. Its length is equal to the number of points in the pattern `i`. It may contain NAs corresponding to points which lie outside the spatial domain of the image `x`. By contrast, `X[i]` or `X[i, drop=TRUE]` contains only those pixel values which are not NA. It may be shorter.

If `i` is a spatial window then `X[i, drop=FALSE]` is another pixel image of the same dimensions as `x` obtained by setting all pixels outside the window `i` to have value NA. When the result is displayed by `plot.im` the effect is that the pixel image `x` is clipped to the window `i`.

If `i` is a spatial window which is *not* a rectangle (`i$type != "rectangle"`) then `X[i, drop=TRUE]` is a numeric vector containing the pixel values for all pixels that lie inside the window `i`.

If `i` is a rectangle (a spatial window with `i$type = "rectangle"`) then `X[i, drop=TRUE]` is a pixel image. The spatial domain of this image is the intersection of `i` with the spatial domain of `x`.

If the optional argument `raster` is given, then it should be a binary image mask or a pixel image. Then `x` will first be converted to an image defined on the pixel grid implied by `raster`, before the subset operation is carried out. In particular, `x[i, raster=i, drop=FALSE]` will return an image defined on the same pixel array as the object `i`.

If `i` does not satisfy any of the conditions above, then it is assumed to be a valid index for the matrix `as.matrix(x)`. The result is a vector or matrix of pixel values.

Value

Either a pixel image or a vector of pixel values. See Details.

Warning

If `W` is a window or a pixel image, then `x[W, drop=FALSE]` will return an image defined on the same pixel array as the original image `x`. If you want to obtain an image whose pixel dimensions agree with those of `W`, use the `raster` argument, `x[W, raster=W, drop=FALSE]`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`im.object`, `[<-.im`, `ppp.object`, `as.ppp`, `owin.object`, `plot.im`

Examples

```
# make up an image
X <- setcov(unit.square())
plot(X)

# a rectangular subset
W <- owin(c(0,0.5),c(0.2,0.8))
```

```

Y <- X[W]
plot(Y)

# a polygonal subset
data(letterR)
R <- affine(letterR, diag(c(1,1)/2), c(-2,-0.7))
Y <- X[R, drop=FALSE]
plot(Y)

# a point pattern
P <- rpoispp(20)
Y <- X[P]

# look up a specified location
X[list(x=0.1,y=0.2)]

# 10 x 10 pixel array
X <- as.im(function(x,y) { x + y }, owin(c(-1,1),c(-1,1)), dimyx=10)
# 100 x 100
W <- as.mask(disc(1, c(0,0)), dimyx=100)
# 10 x 10 raster
X[W,drop=FALSE]
# 100 x 100 raster
X[W, raster=W, drop=FALSE]

```

Extract.listof*Extract or Replace Subset of a List of Things***Description**

Replace a subset of a list of things.

Usage

```
## S3 replacement method for class 'listof'
x[i] <- value
```

Arguments

- | | |
|--------------|--|
| x | An object of class "listof" representing a list of things which all belong to one class. |
| i | Subset index. Any valid subset index in the usual R sense. |
| value | Replacement value for the subset. |

Details

This is a subset replacement method for the class "listof".

The argument **x** should be an object of class "listof" representing a list of things that all belong to one class.

The method replaces a designated subset of **x**, and returns an object of class "listof".

Value

Another object of class "listof".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[plot.listof](#), [summary.listof](#)

Examples

```
x <- list(A=runif(10), B=runif(10), C=runif(10))
class(x) <- c("listof", class(x))
x[1] <- list(A=rnorm(10))
```

Extract.lpp

Extract Subset of Point Pattern on Linear Network

Description

Extract a subset of a point pattern on a linear network.

Usage

```
## S3 method for class 'lpp'
x[i, ...]
```

Arguments

- x A point pattern on a linear network (object of class "lpp").
- i Subset index. A valid subset index in the usual R sense, indicating which points should be retained.
- ... Ignored.

Details

This function extracts a designated subset of a point pattern on a linear network.

The function `[.lpp` is a method for `[` for the class "lpp". It extracts a designated subset of a point pattern. The argument `i` should be a subset index in the usual R sense: either a numeric vector of positive indices (identifying the points to be retained), a numeric vector of negative indices (identifying the points to be deleted) or a logical vector of length equal to the number of points in the point pattern `x`. In the latter case, the points (`x$x[i]`, `x$y[i]`) for which `subset[i]=TRUE` will be retained, and the others will be deleted.

Use the function `unmark` to remove marks from a marked point pattern.

Value

A point pattern on a linear network (of class "lpp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[lpp](#)

Examples

```
# Chicago crimes data - remove cases of assault
chicago[marks(chicago) != "assault"]
```

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[msr](#)

Examples

```
example(msr)
rp[square(0.5)]
rs[ , 2:3]
```

[Extract.hpp](#)

Extract or Replace Subset of Point Pattern

Description

Extract or replace a subset of a point pattern. Extraction of a subset has the effect of thinning the points and/or trimming the window.

Usage

```
## S3 method for class 'ppp'
x[i, j, drop, ...]
## S3 replacement method for class 'ppp'
x[i, j] <- value
```

Arguments

- x A two-dimensional point pattern. An object of class "ppp".
- i Subset index. Either a valid subset index in the usual R sense, indicating which points should be retained, or a window (an object of class "owin") delineating a subset of the original observation window.
- value Replacement value for the subset. A point pattern.
- j Redundant. Included for backward compatibility.
- drop, ... Ignored. These arguments are required for compatibility with the generic function.

Details

These functions extract a designated subset of a point pattern, or replace the designated subset with another point pattern.

The function `[.ppp` is a method for `[` for the class "ppp". It extracts a designated subset of a point pattern, either by "*thinning*" (retaining/deleting some points of a point pattern) or "*trimming*" (reducing the window of observation to a smaller subregion and retaining only those points which lie in the subregion) or both.

The pattern will be "*thinned*" if i is a subset index in the usual R sense: either a numeric vector of positive indices (identifying the points to be retained), a numeric vector of negative indices (identifying the points to be deleted) or a logical vector of length equal to the number of points in the point pattern x. In the latter case, the points (`x$x[i], x$y[i]`) for which `subset[i]=TRUE` will be retained, and the others will be deleted.

The pattern will be “trimmed” if *i* is an object of class “*owin*” specifying a window of observation. The points of *x* lying inside the new window will be retained. Alternatively *i* may be a pixel image (object of class “*im*”) with logical values; the pixels with the value TRUE will be interpreted as a window.

The function [*<-*.*ppp*] is a method for [*<-*] for the class “*ppp*”. It replaces the designated subset with the point pattern value. The subset of *x* to be replaced is designated by the argument *i* as above.

The replacement point pattern value must lie inside the window of the original pattern *x*. The ordering of points in *x* will be preserved if the replacement pattern value has the same number of points as the subset to be replaced. Otherwise the ordering is unpredictable.

If the original pattern *x* has marks, then the replacement pattern value must also have marks, of the same type.

Use the function `unmark` to remove marks from a marked point pattern.

Use the function `split.ppp` to select those points in a marked point pattern which have a specified mark.

Value

A point pattern (of class “*ppp*”).

Warnings

The function does not check whether *window* is a subset of *x\$window*. Nor does it check whether *value* lies inside *x\$window*.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`ppp.object`, `owin.object`, `unmark`, `split.ppp`, `cut.ppp`

Examples

```
data(longleaf)
# Longleaf pines data
## Not run:
plot(longleaf)

## End(Not run)

# adult trees defined to have diameter at least 30 cm
adult <- (longleaf$marks >= 30)
longadult <- longleaf[adult]
## Not run:
plot(longadult)

## End(Not run)
# note that the marks are still retained.
# Use unmark(longadult) to remove the marks
```

```

# New Zealand trees data
data(nztrees)
## Not run:
plot(nztrees)           # plot shows a line of trees at the far right
abline(v=148, lty=2)    # cut along this line

## End(Not run)
nzw <- owin(c(0,148),c(0,95)) # the subwindow
# trim dataset to this subwindow
nzsub <- nztrees[nzw]
## Not run:
plot(nzsub)

## End(Not run)

# Redwood data
data(redwood)
## Not run:
plot(redwood)

## End(Not run)
# Random thinning: delete 60% of data
retain <- (runif(redwood$n) < 0.4)
thinred <- redwood[retain]
## Not run:
plot(thinred)

## End(Not run)
# Scramble 60% of data
modif <- (runif(redwood$n) < 0.6)
scramble <- function(x) { runifpoint(x$n, x>window) }
redwood[modif] <- scramble(redwood[modif])

# Lansing woods data - multitype points
data(lansing)

# Hickory trees
hicks <- split(lansing)$hickory

# Trees in subwindow
win <- owin(c(0.3, 0.6),c(0.2, 0.5))
lsub <- lansing[win]

# Scramble the locations of trees in subwindow, retaining their marks
lansing[win] <- scramble(lsub) %mark% (lsub$marks)

```

Description

Extract a subset of a multidimensional point pattern.

Usage

```
## S3 method for class 'ppx'
x[i, ...]
```

Arguments

x	A multidimensional point pattern (object of class "ppx").
i	Subset index. A valid subset index in the usual R sense, indicating which points should be retained.
...	Ignored.

Details

This function extracts a designated subset of a multidimensional point pattern.

The function `[.ppx` is a method for `[` for the class "ppx". It extracts a designated subset of a point pattern. The argument `i` should be a subset index in the usual R sense: either a numeric vector of positive indices (identifying the points to be retained), a numeric vector of negative indices (identifying the points to be deleted) or a logical vector of length equal to the number of points in the point pattern `x`. In the latter case, the points (`x$x[i], x$y[i]`) for which `subset[i]=TRUE` will be retained, and the others will be deleted.

Use the function `unmark` to remove marks from a marked point pattern.

Value

A multidimensional point pattern (of class "ppx") in the same domain.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppx](#)

Examples

```
df <- data.frame(x=runif(4),y=runif(4),z=runif(4))
X <- ppx(data=df, coord.type=c("s","s","t"))
X[-2]
```

Description

Extract a subset of a line segment pattern.

Usage

```
## S3 method for class 'psp'
x[i, j, drop, ...]
```

Arguments

x	A two-dimensional line segment pattern. An object of class "psp".
i	Subset index. Either a valid subset index in the usual R sense, indicating which segments should be retained, or a window (an object of class "owin") delineating a subset of the original observation window.
j	Redundant - included for backward compatibility.
drop	Ignored. Required for compatibility with generic function.
...	Ignored.

Details

These functions extract a designated subset of a line segment pattern.

The function `[.psp` is a method for `[` for the class "psp". It extracts a designated subset of a line segment pattern, either by "*thinning*" (retaining/deleting some line segments of a line segment pattern) or "*trimming*" (reducing the window of observation to a smaller subregion and clipping the line segments to this boundary) or both.

The pattern will be "*thinned*" if `subset` is specified. The line segments designated by `subset` will be retained. Here `subset` can be a numeric vector of positive indices (identifying the line segments to be retained), a numeric vector of negative indices (identifying the line segments to be deleted) or a logical vector of length equal to the number of line segments in the line segment pattern `x`. In the latter case, the line segments for which `subset[i]=TRUE` will be retained, and the others will be deleted.

The pattern will be "*trimmed*" if `window` is specified. This should be an object of class `owin` specifying a window of observation to which the line segment pattern `x` will be trimmed. Line segments of `x` lying inside the new window will be retained unchanged. Line segments lying partially inside the new window and partially outside it will be clipped so that they lie entirely inside.

Both "*thinning*" and "*trimming*" can be performed together.

Value

A line segment pattern (of class "psp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`psp.object`, `owin.object`

Examples

```
a <- psp(runif(20),runif(20),runif(20),runif(20), window=owin())
plot(a)
# thinning
id <- sample(c(TRUE, FALSE), 20, replace=TRUE)
b <- a[id]
plot(b, add=TRUE, lwd=3)
# trimming
plot(a)
w <- owin(c(0.1,0.7), c(0.2, 0.8))
b <- a[,w]
plot(b, add=TRUE, col="red")
```

Extract.quad

Subset of Quadrature Scheme

Description

Extract a subset of a quadrature scheme.

Usage

```
## S3 method for class 'quad'
x[...]
```

Arguments

x	A quadrature scheme (object of class "quad").
...	Arguments passed to [.ppp] to determine the subset.

Details

This function extracts a designated subset of a quadrature scheme.

The function [\[.quad\]](#) is a method for [\[](#) for the class "quad". It extracts a designated subset of a quadrature scheme.

The subset to be extracted is determined by the arguments ... which are interpreted by [\[.ppp\]](#). Thus it is possible to take the subset consisting of all quadrature points that lie inside a given region, or a subset of quadrature points identified by numeric indices.

Value

A quadrature scheme (object of class "quad").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[quad.object](#), [\[.ppp\]](#).

Examples

```
data(nztrees)
Q <- quadscheme(nztrees)
W <- owin(c(0,148),c(0,95)) # a subwindow
Q[W]
```

Extract.splitppp

Extract or Replace Sub-Patterns

Description

Extract or replace some of the sub-patterns in a split point pattern.

Usage

```
## S3 method for class 'splitppp'
x[...]
## S3 replacement method for class 'splitppp'
x[...] <- value
```

Arguments

- x An object of class "splitppp", representing a point pattern separated into a list of sub-patterns.
- ... Subset index. Any valid subset index in the usual R sense.
- value Replacement value for the subset. A list of point patterns.

Details

These are subset methods for the class "splitppp".

The argument x should be an object of class "splitppp", representing a point pattern that has been separated into a list of sub-patterns. It is created by [split.ppp](#).

The methods extract or replace a designated subset of the list x, and return an object of class "splitppp".

Value

Another object of class "splitppp".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[split.ppp](#), [plot.splitppp](#), [summary.splitppp](#)

Examples

```
data(amacrine) # multitype point pattern
y <- split(amacrine)
y[1]
y[["off"]]
y[1] <- list(runifpoint(42, amacrine>window))
```

Extract.tess

Extract or Replace Subset of Tessellation

Description

Extract, change or delete a subset of the tiles of a tessellation, to make a new tessellation.

Usage

```
## S3 method for class 'tess'
x[...]
## S3 replacement method for class 'tess'
x[...] <- value
```

Arguments

- x A tessellation (object of class "tess").
- ... One argument that specifies the subset to be extracted or changed. Any valid format for the subset index in a list.
- value Replacement value for the selected tiles of the tessellation. A list of windows (objects of class "owin") or NULL.

Details

A tessellation (object of class "tess", see [tess](#)) is effectively a list of tiles (spatial regions) that cover a spatial region. The subset operator `[.tess` extracts some of these tiles and forms a new tessellation, which of course covers a smaller region than the original.

The replacement operator changes the selected tiles. The replacement value may be either NULL (which causes the selected tiles to be removed from x) or a list of the same length as the selected subset. The entries of value may be windows (objects of class "owin") or NULL to indicate that the corresponding tile should be deleted.

Generally it does not make sense to replace a tile in a tessellation with a completely different tile, because the tiles are expected to fit together. However this facility is sometimes useful for making small adjustments to polygonal tiles.

Value

A tessellation (object of class "tess").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also[tess](#),**Examples**

```
A <- tess(xgrid=0:4, ygrid=0:3)
B <- A[c(1, 3, 7)]
E <- A[-1]
A[c(2, 5, 11)] <- NULL
```

F3est*Empty Space Function of a Three-Dimensional Point Pattern***Description**

Estimates the empty space function $F_3(r)$ from a three-dimensional point pattern.

Usage

```
F3est(X, ..., rmax = NULL, nrval = 128, vside = NULL,
      correction = c("rs", "km", "cs"),
      sphere = c("fudge", "ideal", "digital"))
```

Arguments

<i>X</i>	Three-dimensional point pattern (object of class "pp3").
...	Ignored.
<i>rmax</i>	Optional. Maximum value of argument r for which $F_3(r)$ will be estimated.
<i>nrval</i>	Optional. Number of values of r for which $F_3(r)$ will be estimated. A large value of <i>nrval</i> is required to avoid discretisation effects.
<i>vside</i>	Optional. Side length of the voxels in the discrete approximation.
<i>correction</i>	Optional. Character vector specifying the edge correction(s) to be applied. See Details.
<i>sphere</i>	Optional. Character string specifying how to calculate the theoretical value of $F_3(r)$ for a Poisson process. See Details.

Details

For a stationary point process Φ in three-dimensional space, the empty space function is

$$F_3(r) = P(d(0, \Phi) \leq r)$$

where $d(0, \Phi)$ denotes the distance from a fixed origin 0 to the nearest point of Φ .

The three-dimensional point pattern *X* is assumed to be a partial realisation of a stationary point process Φ . The empty space function of Φ can then be estimated using techniques described in the References.

The box containing the point pattern is discretised into cubic voxels of side length `vside`. The distance function $d(u, \Phi)$ is computed for every voxel centre point u using a three-dimensional version of the distance transform algorithm (Borgefors, 1986). The empirical cumulative distribution function of these values, with appropriate edge corrections, is the estimate of $F_3(r)$.

The available edge corrections are:

- "`rs`": the reduced sample (aka minus sampling, border correction) estimator (Baddeley et al, 1993)
- "`km`": the three-dimensional version of the Kaplan-Meier estimator (Baddeley and Gill, 1997)
- "`cs`": the three-dimensional generalisation of the Chiu-Stoyan or Hanisch estimator (Chiu and Stoyan, 1998).

The result includes a column `theo` giving the theoretical value of $F_3(r)$ for a uniform Poisson process (Complete Spatial Randomness). This value depends on the volume of the sphere of radius r measured in the discretised distance metric. The argument `sphere` determines how this will be calculated.

- If `sphere="ideal"` the calculation will use the volume of an ideal sphere of radius r namely $(4/3)\pi r^3$. This is not recommended because the theoretical values of $F_3(r)$ are inaccurate.
- If `sphere="fudge"` then the volume of the ideal sphere will be multiplied by 0.78, which gives the approximate volume of the sphere in the discretised distance metric.
- If `sphere="digital"` then the volume of the sphere in the discretised distance metric is computed exactly using another distance transform. This takes longer to compute, but is exact.

Value

A function value table (object of class "`fv`") that can be plotted, printed or coerced to a data frame containing the function values.

Warnings

A small value of `vside` and a large value of `nrv` are required for reasonable accuracy.

The default value of `vside` ensures that the total number of voxels is 2^{22} or about 4 million. To change the default number of voxels, see `spatstat.options("nvoxel")`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rana Moyeed.

References

- Baddeley, A.J., Moyeed, R.A., Howard, C.V. and Boyde, A. Analysis of a three-dimensional point pattern with replication. *Applied Statistics* **42** (1993) 641–668.
- Baddeley, A.J. and Gill, R.D. (1997) Kaplan-Meier estimators of interpoint distance distributions for spatial point processes. *Annals of Statistics* **25**, 263–292.
- Borgefors, G. (1986) Distance transformations in digital images. *Computer Vision, Graphics and Image Processing* **34**, 344–371.
- Chiu, S.N. and Stoyan, D. (1998) Estimators of distance distributions for spatial patterns. *Statistica Neerlandica* **52**, 239–246.

See Also

[G3est](#), [K3est](#), [pcf3est](#).

Examples

```
X <- rpoispp(42)
Z <- F3est(X)
if(interactive()) plot(Z)
```

Description

A class "fasp" to represent a "matrix" of functions, amenable to plotting as a matrix of plot panels.

Details

An object of this class is a convenient way of storing (and later plotting, editing, etc) a set of functions $f_{i,j}(r)$ of a real argument r , defined for each possible pair (i, j) of indices $1 \leq i, j \leq n$. We may think of this as a matrix or array of functions $f_{i,j}$.

Function arrays are particularly useful in the analysis of a multitype point pattern (a point pattern in which the points are identified as belonging to separate types). We may want to compute a summary function for the points of type i only, for each of the possible types i . This produces a $1 \times m$ array of functions. Alternatively we may compute a summary function for each possible pair of types (i, j) . This produces an $m \times m$ array of functions.

For multitype point patterns the command [alltypes](#) will compute arrays of summary functions for each possible type or for each possible pair of types. The function [alltypes](#) returns an object of class "fasp".

An object of class "fasp" is a list containing at least the following components:

fns A list of data frames, each representing one of the functions.

which A matrix representing the spatial arrangement of the functions. If `which[i, j] = k` then the function represented by `fns[[k]]` should be plotted in the panel at position (i, j) . If `which[i, j] = NA` then nothing is plotted in that position.

titles A list of character strings, providing suitable plotting titles for the functions.

default.formulae A list of default formulae for plotting each of the functions.

title A character string, giving a default title for the array when it is plotted.

Functions available

There are methods for `plot`, `print` and "`[`" for this class.

The `plot` method displays the entire array of functions. The method [\[.fasp](#) selects a sub-array using the natural indices i, j .

The command [eval.fasp](#) can be used to apply a transformation to each function in the array, and to combine two arrays.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[alltypes](#), [plot.fasp](#), [\[.fasp](#), [eval.fasp](#)

Examples

```
# multitype point pattern
data(amacrine)
GG <- alltypes(amacrine, "G")
plot(GG)

# select the row corresponding to cells of type "on"
Gon <- GG["on", ]
plot(Gon)

# extract the G function for i = "on", j = "off"
Gonoff <- GG["on", "off", drop=TRUE]

# Fisher variance stabilising transformation
GGfish <- eval.fasp(asin(sqrt(GG)))
plot(GGfish)
```

Fest

*Estimate the empty space function F***Description**

Estimates the empty space function $F(r)$ from a point pattern in a window of arbitrary shape.

Usage

```
Fest(X, ..., eps, r=NULL, breaks=NULL, correction=c("rs", "km", "cs"))
```

Arguments

X	The observed point pattern, from which an estimate of $F(r)$ will be computed. An object of class ppp , or data in any format acceptable to as.ppp() .
...	Ignored.
eps	Optional. A positive number. The resolution of the discrete approximation to Euclidean distance (see below). There is a sensible default.
r	Optional. Numeric vector. The values of the argument r at which $F(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r .
breaks	An alternative to the argument r . Not normally invoked by the user. See the Details section.
correction	Optional. The edge correction(s) to be used to estimate $F(r)$. A vector of character strings selected from "none", "rs", "km", "cs" and "best".

Details

The empty space function (also called the “*spherical contact distribution*” or the “*point-to-nearest-event*” distribution) of a stationary point process X is the cumulative distribution function F of the distance from a fixed point in space to the nearest point of X .

An estimate of F derived from a spatial point pattern dataset can be used in exploratory data analysis and formal inference about the pattern (Cressie, 1991; Diggle, 1983; Ripley, 1988). In exploratory analyses, the estimate of F is a useful statistic summarising the sizes of gaps in the pattern. For inferential purposes, the estimate of F is usually compared to the true value of F for a completely random (Poisson) point process, which is

$$F(r) = 1 - e^{-\lambda\pi r^2}$$

where λ is the intensity (expected number of points per unit area). Deviations between the empirical and theoretical F curves may suggest spatial clustering or spatial regularity.

This algorithm estimates the empty space function F from the point pattern X . It assumes that X can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in X) may have arbitrary shape.

The argument X is interpreted as a point pattern object (of class "ppp", see [ppp.object](#)) and can be supplied in any of the formats recognised by [as.ppp](#).

The algorithm uses two discrete approximations which are controlled by the parameter `eps` and by the spacing of values of `r` respectively. (See below for details.) First-time users are strongly advised not to specify these arguments.

The estimation of F is hampered by edge effects arising from the unobservability of points of the random pattern outside the window. An edge correction is needed to reduce bias (Baddeley, 1998; Ripley, 1988). The edge corrections implemented here are the border method or “*reduced sample*” estimator, the spatial Kaplan-Meier estimator (Baddeley and Gill, 1997) and the Chiu-Stoyan estimator (Chiu and Stoyan, 1998).

Our implementation makes essential use of the distance transform algorithm of image processing (Borgefors, 1986). A fine grid of pixels is created in the observation window. The Euclidean distance between two pixels is approximated by the length of the shortest path joining them in the grid, where a path is a sequence of steps between adjacent pixels, and horizontal, vertical and diagonal steps have length 1, 1 and $\sqrt{2}$ respectively in pixel units. If the pixel grid is sufficiently fine then this is an accurate approximation.

The parameter `eps` is the pixel width of the rectangular raster used to compute the distance transform (see below). It must not be too large: the absolute error in distance values due to discretisation is bounded by `eps`.

If `eps` is not specified, the function checks whether the window $X>window$ contains pixel raster information. If so, then `eps` is set equal to the pixel width of the raster; otherwise, `eps` defaults to 1/100 of the width of the observation window.

The argument `r` is the vector of values for the distance r at which $F(r)$ should be evaluated. It is also used to determine the breakpoints (in the sense of [hist](#) for the computation of histograms of distances. The estimators are computed from histogram counts. This introduces a discretisation error which is controlled by the fineness of the breakpoints.

First-time users would be strongly advised not to specify `r`. However, if it is specified, `r` must satisfy `r[1] = 0`, and `max(r)` must be larger than the radius of the largest disc contained in the window. Furthermore, the spacing of successive `r` values must be very fine (ideally not greater than `eps/4`).

The algorithm also returns an estimate of the hazard rate function, $\lambda(r)$, of $F(r)$. The hazard rate is defined by

$$\lambda(r) = -\frac{d}{dr} \log(1 - F(r))$$

The hazard rate of F has been proposed as a useful exploratory statistic (Baddeley and Gill, 1994). The estimate of $\lambda(r)$ given here is a discrete approximation to the hazard rate of the Kaplan-Meier estimator of F . Note that F is absolutely continuous (for any stationary point process X), so the hazard function always exists (Baddeley and Gill, 1997).

The naive empirical distribution of distances from each location in the window to the nearest point of the data pattern, is a biased estimate of F . However this is also returned by the algorithm (if `correction="none"`), as it is sometimes useful in other contexts. Care should be taken not to use the uncorrected empirical F as if it were an unbiased estimator of F .

Value

An object of class "fv", see [fv.object](#), which can be plotted directly using [plot.fv](#).

Essentially a data frame containing up to seven columns:

<code>r</code>	the values of the argument r at which the function $F(r)$ has been estimated
<code>rs</code>	the “reduced sample” or “border correction” estimator of $F(r)$
<code>km</code>	the spatial Kaplan-Meier estimator of $F(r)$
<code>hazard</code>	the hazard rate $\lambda(r)$ of $F(r)$ by the spatial Kaplan-Meier method
<code>cs</code>	the Chiu-Stoyan estimator of $F(r)$
<code>raw</code>	the uncorrected estimate of $F(r)$, i.e. the empirical distribution of the distance from a random point in the window to the nearest point of the data pattern X
<code>theo</code>	the theoretical value of $F(r)$ for a stationary Poisson process of the same estimated intensity.

Warnings

The reduced sample (border method) estimator of F is pointwise approximately unbiased, but need not be a valid distribution function; it may not be a nondecreasing function of r . Its range is always within $[0, 1]$.

The spatial Kaplan-Meier estimator of F is always nondecreasing but its maximum value may be less than 1.

The estimate of $\lambda(r)$ returned by the algorithm is an approximately unbiased estimate for the integral of $\lambda()$ over the corresponding histogram cell. It may exhibit oscillations due to discretisation effects. We recommend modest smoothing, such as kernel smoothing with kernel width equal to the width of a histogram cell.

Note

Sizeable amounts of memory may be needed during the calculation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Baddeley, A.J. Spatial sampling and censoring. In O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*. Chapman and Hall, 1998. Chapter 2, pages 37-78.
- Baddeley, A.J. and Gill, R.D. The empty space hazard of a spatial pattern. Research Report 1994/3, Department of Mathematics, University of Western Australia, May 1994.
- Baddeley, A.J. and Gill, R.D. Kaplan-Meier estimators of interpoint distance distributions for spatial point processes. *Annals of Statistics* **25** (1997) 263-292.
- Borgefors, G. Distance transformations in digital images. *Computer Vision, Graphics and Image Processing* **34** (1986) 344-371.
- Chiu, S.N. and Stoyan, D. (1998) Estimators of distance distributions for spatial patterns. *Statistica Neerlandica* **52**, 239–246.
- Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.
- Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.
- Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.
- Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.

See Also

[Gest](#), [Jest](#), [Kest](#), [km.rs](#), [reduced.sample](#), [kaplan.meier](#)

Examples

```

data(cells)
Fc <- Fest(cells, 0.01)

# Tip: don't use F for the left hand side!
# That's an abbreviation for FALSE

plot(Fc)

# P-P style plot
plot(Fc, cbind(km, theo) ~ theo)

# The empirical F is above the Poisson F
# indicating an inhibited pattern

## Not run:
plot(Fc, . ~ theo)
plot(Fc, asin(sqrt(.)) ~ asin(sqrt(theo)))

## End(Not run)

```

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, 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 and κ it is possible to use [profilepl](#). The maximum likelihood estimator of h is the minimum interpoint distance.

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 κ .

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.
- Fiksel, T. (1984) Estimation of parameterized pair potentials of marked and non-marked Gibbsian point processes. *Elektronische Informationsverarbeitung und Kybernetika* **20**, 270–278.
- Stoyan, D, Kendall, W.S. and Mecke, J. (1987) *Stochastic geometry and its applications*. Wiley.

See Also

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

Examples

```
Fiksel(r=1, hc=0.02, kappa=2)
# prints a sensible description of itself

data(spruces)
X <- unmark(spruces)

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

finpines

Pine saplings in Finland.

Description

The data record the locations of 126 pine saplings in a Finnish forest, their heights and their diameters.

The dataset `finpines` is a marked point pattern containing the locations of the saplings marked by their heights and their diameters.

Sapling locations are given in metres (to six significant digits); heights are in metres (rounded to the nearest 0.1 metre, except in one case to the nearest 0.05 metres); diameters are in centimetres (rounded to the nearest centimetre).

The data were recorded by Professor Seppo Kellomaki, Faculty of Forestry, University of Joensuu, Finland, and subsequently massaged by Professor Antti Penttinen, Department of Statistics, University of Jyväskylä, Finland.

Originally the point locations were observed in polar coordinates with rather poor angular precision. Hence the coordinates are imprecise for large radius because of rounding errors: indeed the alignments can be observed by eye.

The data were manipulated by Prof Penttinen by making small angular perturbations at random. After this transformation, the original data (in a circular plot) were clipped to a square window, for convenience.

Professor Penttinen emphasises that the data were intended only for initial experimentation. They have some strange features. For example, if the height is less than 1.3 metres then the diameter can be uncertain. Also there are some very close pairs of points. Some pairs of trees (namely (58,59), (78,79), (96,97) and (102,103)) violate the requirement that the interpoint distance should be greater than half the sum of their diameters.

These data have subsequently been analysed by Van Lieshout (2004).

Usage

```
data(finpinies)
```

Format

Object of class "ppp" representing the point pattern of sapling locations marked by their heights and diameters. See [ppp.object](#) for details of the format.

Source

Prof Antti Penttinen

References

Van Lieshout, M.N.M. (2004) A J-function for marked point patterns. Research Report PNA-R0404, June 2004. Centrum voor Wiskunde en Informatica (CWI), Amsterdam, 2004.

Examples

```
data(finpinies)
plot(unmark(finpinies), main="Finnish pines: locations")
plot(finpinies, which.marks="height", main="heights")
plot(finpinies, which.marks="diameter", main="diameters")
plot(finpinies, which.marks="diameter",
      main="diameters to scale", markscale=1/200)
```

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)
```

Arguments

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

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.

Value

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

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppm](#), [ppm.object](#), [as.interact](#).

Examples

```
data(betacells)

# unmarked
X <- unmark(betacells)
model <- ppm(X, ~1, PairPiece(seq(10,100,by=10)))
f <- fitin(model)
f
plot(f)

# marked
# fit the stationary multitype Strauss process to 'betacells'
r <- 30.0 * matrix(c(1,2,2,1), nrow=2,ncol=2)
model <- ppm(betacells, ~1, MultiStrauss(c("off","on"), r))
f <- fitin(model)
f
plot(f)
```

fitted.ppmFitted Conditional Intensity for Point Process Model

Description

Given a point process model fitted to a point pattern, compute the fitted conditional intensity 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,
       drop=FALSE, check=TRUE, repair=TRUE)
```

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 the trend ("trend").
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.
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.

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.

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 or (if `type="trend"`) the fitted spatial trend.

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A., Turner, R., Moller, 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

```
data(cells)
str <- ppm(cells, ~x, Strauss(r=0.15))
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)
```

Description

Given a fitted Spatial Logistic Regression model, this function computes the fitted probabilities for each pixel.

Usage

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

Arguments

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

Details

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

The algorithm computes the fitted probabilities of the presence of a random point in each pixel.

Value

A pixel image (object of class "im") containing the fitted probability for each pixel.

Author(s)

Adrian Baddeley <adrian@maths.uwa.edu.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`slrm`, `fitted`

Examples

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

flipxy

Exchange X and Y Coordinates

Description

Exchanges the x and y coordinates in a spatial dataset.

Usage

```
flipxy(X)
## S3 method for class 'owin'
flipxy(X)
## S3 method for class 'ppp'
flipxy(X)
## S3 method for class 'psp'
flipxy(X)
## S3 method for class 'im'
flipxy(X)
```

Arguments

`X` Spatial dataset. An object of class "owin", "ppp", "psp" or "im".

Details

This function swaps the x and y coordinates of a spatial dataset. This could also be performed using the command `affine`, but `flipxy` is faster.

The function `flipxy` is generic, with methods for the classes of objects listed above.

Value

Another object of the same type, representing the result of swapping the x and y coordinates.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[affine](#), [reflect](#), [rotate](#), [shift](#)

Examples

```
data(cells)
X <- flipxy(cells)
```

flu

Influenza Virus Proteins

Description

Replicated spatial point patterns giving the locations of two different virus proteins on the membranes of cells infected with influenza virus.

Usage

```
data(flu)
```

Format

A [hyperframe](#) with 41 rows and four columns:

pattern List of spatial point patterns (objects of class "ppp") with points of two types, identifying the locations of two different proteins on a membrane sheet.

virustype Factor identifying whether the infecting virus was the wild type (wt) or mutant (mut1).

stain Factor identifying whether the membrane sheet was stained for the proteins *M2* and *M1* (stain="M2-M1") or stained for the proteins *M2* and *HA* (stain="M2-HA").

frameid Integer. Serial number of the microscope frame in the original experiment. Frame identifier is not unique across different values of virustype and stain.

The row names of the hyperframe can be used as succinct labels in plots.

Details

The data consist of 41 spatial point patterns, each giving the locations of two different virus proteins on the membranes of cells infected with influenza virus.

Chen et al (2008) conducted the experiment and used spatial analysis to establish evidence for an interaction between the influenza virus proteins M1 and M2 that is important for the study of viral replication.

Canine kidney cells were infected with human influenza, Udorn strain, either the wild type or a mutant which encodes a defective M2 protein. At twelve hours post-infection, membrane sheets were prepared and stained for viral proteins, using two antibodies conjugated to gold particles of two sizes (6 nanometre and 12 nanometre diameter) enabling localisation of two different proteins on each sheet. The 6 nm particles were stained for M2 (ion channel protein), while the 12 nm particles were stained either for M1 (matrix protein) or for HA (hemagglutinin). Membrane sheets were visualised in electron microscopy.

Experimental technique and spatial analysis of the membranes stained for M2 and M1 is reported in Chen et al (2008). Analysis of the membranes stained for M2 and HA is reported in Rossman et al (2010). The M2-HA data shows a stronger association between the two proteins which has also been observed biochemically and functionally (Rossman et al, 2010).

The dataset `flu` is a [hyperframe](#) with one row for each membrane sheet. The column named `pattern` contains the spatial point patterns of gold particle locations, with two types of points (either M1 and M2 or HA and M2). The column named `virustype` is a factor identifying the virus: either wild type `wt` or mutant `mut1`. The column named `stain` is a factor identifying whether the membrane was stained for M1 and M2 (`stain="M2-M1"`) or stained for HA and M2 (`stain="M2-HA"`). The row names of the hyperframe are a succinct summary of the experimental conditions and can be used as labels in plots. See the Examples.

Source

Data generously provided by Dr G.P. Leser and Dr R.A. Lamb. Please cite Chen et al (2008) in any use of these data.

References

Chen, B.J., Leser, G.P., Jackson, D. and Lamb, R.A. (2008) The influenza virus M2 protein cytoplasmic tail interacts with the M1 protein and influences virus assembly at the site of virus budding. *Journal of Virology* **82**, 10059–10070.

Rossman, J.S., Jing, X.H., Leser, G.P. and Lamb, R.A. (2010) Influenza virus M2 protein mediates ESCRT-independent membrane scission *Cell* **142**, 902–913.

Examples

```
data(flu)
flu
Y <- flu$pattern[10]
Y <- flu[10, 1, drop=TRUE]
wildM1 <- with(flu, virustype == "wt" & stain == "M2-M1")
plot(flu[wildM1, 1, drop=TRUE],
     main=c("flu data", "wild type virus, M2-M1 stain"),
     pch=c(3,16), cex=0.4, cols=2:3)
```

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

See Also

ppm, as.owin, coef.ppm, extractAIC.ppm, fitted.ppm, logLik.ppm, model.frame.ppm, model.matrix.ppm, plot.ppm, predict.ppm, residuals.ppm, simulate.ppm, summary.ppm, update.ppm, vcov.ppm.

Examples

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

fryplot*Fry Plot of Point Pattern*

Description

Displays the Fry plot (Patterson plot) of a spatial point pattern.

Usage

```
fryplot(X, ..., width=NULL, from=NULL, to=NULL)
frpoints(X)
```

Arguments

X	A point pattern (object of class "ppp") or something acceptable to as.ppp .
...	Optional arguments to control the appearance of the plot.
width	Optional parameter indicating the width of a box for a zoomed-in view of the Fry plot near the origin.
from, to	Optional. Subset indices specifying which points of X will be considered when forming the vectors (drawn from each point of from, to each point of to.)

Details

The function `fryplot` generates a Fry plot (or Patterson plot); `frpoints` returns the points of the Fry plot as a point pattern dataset.

Fry (1979) and Hanna and Fry (1979) introduced a manual graphical method for investigating features of a spatial point pattern of mineral deposits. A transparent sheet, marked with an origin or centre point, is placed over the point pattern. The transparent sheet is shifted so that the origin lies over one of the data points, and the positions of all the *other* data points are copied onto the transparent sheet. This procedure is repeated for each data point in turn. The resulting plot (the Fry plot) is a pattern of $n(n - 1)$ points, where n is the original number of data points. This procedure was previously proposed by Patterson (1934, 1935) for studying inter-atomic distances in crystals, and is also known as a Patterson plot.

The function `fryplot` generates the Fry/Patterson plot. Standard graphical parameters such as `main`, `pch`, `lwd`, `col`, `bg`, `cex` can be used to control the appearance of the plot. To zoom in (to view only a subset of the Fry plot at higher magnification), use the argument `width` to specify the width of a rectangular field of view centred at the origin, or the standard graphical arguments `xlim` and `ylim` to specify another rectangular field of view. (The actual field of view may be slightly larger, depending on the graphics device.)

The function `frpoints` returns the points of the Fry plot as a point pattern object. There may be a large number of points in this pattern, so this function should be used only if further analysis of the Fry plot is required.

Fry plots are particularly useful for recognising anisotropy in regular point patterns. A void around the origin in the Fry plot suggests regularity (inhibition between points) and the shape of the void gives a clue to anisotropy in the pattern. Fry plots are also useful for detecting periodicity or rounding of the spatial coordinates.

In mathematical terms, the Fry plot of a point pattern `X` is simply a plot of the vectors `X[i] - X[j]` connecting all pairs of distinct points in `X`.

The Fry plot is related to the K function (see [Kest](#)) and the reduced second moment measure (see [Kmeasure](#)). For example, the number of points in the Fry plot lying within a circle of given radius is an unnormalised and uncorrected version of the K function. The Fry plot has a similar appearance to the plot of the reduced second moment measure [Kmeasure](#) when the smoothing parameter sigma is very small.

The Fry plot does not adjust for the effect of the size and shape of the sampling window. The density of points in the Fry plot tapers off near the edges of the plot. This is an edge effect, a consequence of the bounded sampling window. In geological applications this is usually not important, because interest is focused on the behaviour near the origin where edge effects can be ignored. To correct for the edge effect, use [Kmeasure](#) or [Kest](#) or its relatives.

Value

`fryplot` returns NULL. `frypoints` returns a point pattern (object of class "ppp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Fry, N. (1979) Random point distributions and strain measurement in rocks. *Tectonophysics* **60**, 89–105.
- Hanna, S.S. and Fry, N. (1979) A comparison of methods of strain determination in rocks from southwest Dyfed (Pembrokeshire) and adjacent areas. *Journal of Structural Geology* **1**, 155–162.
- Patterson, A.L. (1934) A Fourier series method for the determination of the component of inter-atomic distances in crystals. *Physics Reviews* **46**, 372–376.
- Patterson, A.L. (1935) A direct method for the determination of the components of inter-atomic distances in crystals. *Zeitschrift fuer Krystallographie* **90**, 517–554.

See Also

[Kmeasure](#), [Kest](#)

Examples

```
data(cells)
fryplot(cells)
Y <- frypoints(cells)
data(amacrine)
fryplot(amacrine, width=0.2,
       from=(marks(amacrine) == "on"),
       chars=c(3,16), cols=2:3,
       main="Fry plot centred at an On-cell")
points(0,0)
```

fv

Create a Function Value Table

Description

Advanced Use Only. This low-level function creates an object of class "fv" from raw numerical data.

Usage

```
fv(x, argu = "r", ylab = NULL, valu, fmla = NULL, alim = NULL, labl =
names(x), desc = NULL, unitname = NULL, fname = NULL, yexp = ylab)
```

Arguments

x	A data frame with at least 2 columns containing the values of the function argument and the corresponding values of (one or more versions of) the function.
argu	String. The name of the column of x that contains the values of the function argument.
ylab	Either NULL, or an R language expression representing the mathematical name of the function. See Details.
valu	String. The name of the column of x that should be taken as containing the function values, in cases where a single column is required.
fmla	Either NULL, or a formula specifying the default plotting behaviour. See Details.
alim	Optional. The default range of values of the function argument for which the function will be plotted. Numeric vector of length 2.
labl	Optional. Plot labels for the columns of x. A vector of strings, with one entry for each column of x.
desc	Optional. Descriptions of the columns of x. A vector of strings, with one entry for each column of x.
unitname	Optional. Name of the unit (usually a unit of length) in which the function argument is expressed. Either a single character string, or a vector of two character strings giving the singular and plural forms, respectively.
fname	Optional. The name of the function itself. A character string.
yexp	Optional. Alternative form of ylab more suitable for annotating an axis of the plot. See Details.

Details

This documentation is provided for experienced programmers who want to modify the internal behaviour of **spatstat**. Other users please see [fv.object](#).

The low-level function fv is used to create an object of class "fv" from raw numerical data.

The data frame x contains the numerical data. It should have one column (typically but not necessarily named "r") giving the values of the function argument for which the function has been evaluated; and at least one other column, containing the corresponding values of the function.

Typically there is more than one column of function values. These columns typically give the values of different versions or estimates of the same function, for example, different estimates of the K

function obtained using different edge corrections. However they may also contain the values of related functions such as the derivative or hazard rate.

`argu` specifies the name of the column of `x` that contains the values of the function argument (typically `argu="r"` but this is not compulsory).

`valu` specifies the name of another column that contains the ‘recommended’ estimate of the function. It will be used to provide function values in those situations where a single column of data is required. For example, `envelope` computes its simulation envelopes using the recommended value of the summary function.

`fmla` specifies the default plotting behaviour. It should be a formula, or a string that can be converted to a formula. Variables in the formula are names of columns of `x`. See `plot.fv` for the interpretation of this formula.

`alim` specifies the recommended range of the function argument. This is used in situations where statistical theory or statistical practice indicates that the computed estimates of the function are not trustworthy outside a certain range of values of the function argument. By default, `plot.fv` will restrict the plot to this range.

`fname` is a string giving the name of the function itself. For example, the K function would have `fname="K"`.

`ylab` is a mathematical expression for the function value, used when labelling an axis of the plot, or when printing a description of the function. It should be an R language object. For example the K function’s mathematical name $K(r)$ is rendered by `ylab=substitute(K(r), NULL)`.

If `yexp` is present, then `ylab` will be used only for printing, and `yexp` will be used for annotating axes in a plot. (Otherwise `yexp` defaults to `ylab`). For example the cross-type K function $K_{1,2}(r)$ is rendered by something like `ylab=substitute(Kcross[i,j](r), list(i=1,j=2))` and `yexp=substitute(Kcross[list(i,j)](r), list(i=1,j=2))` to get the most satisfactory behaviour.

`label` is a character vector specifying plot labels for each column of `x`. These labels will appear on the plot axes (in non-default plots), legends and printed output. Entries in `label` may contain the string "%s" which will be replaced by `fname`. For example the border-corrected estimate of the K function has label "%sbord(r)" which becomes "Kbord(r)".

`desc` is a character vector containing intelligible explanations of each column of `x`. Entries in `desc` may contain the string "%s" which will be replaced by `ylab`. For example the border correction estimate of the K function has description "border correction estimate of %s".

Value

An object of class "fv", see `fv.object`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

See `plot.fv` for plotting an "fv" object.

See `as.function.fv` to convert an "fv" object to an R function.

Use `cbind.fv` to combine several "fv" objects. Use `bind.fv` to glue additional columns onto an existing "fv" object.

Undocumented functions for modifying an "fv" object include `fvnames`, `fvnames<-`, `tweak.fv.entry` and `rebadge.fv`.

Examples

```
df <- data.frame(r=seq(0,5,by=0.1))
df <- transform(df, a=pi*r^2, b=3*r^2)
X <- fv(df, "r", substitute(A(r), NULL),
         "a", cbind(a, b) ~ r,
         alim=c(0,4),
         labl=c("r", "%s[true](r)", "%s[approx](r)"),
         desc=c("radius of circle",
                "true area %s",
                "rough area %s"),
         fname="A")
X
```

fv.object

Function Value Table

Description

A class "fv" to support the convenient plotting of several estimates of the same function.

Details

An object of this class is a convenient way of storing and plotting several different estimates of the same function.

It is a data frame with extra attributes indicating the recommended way of plotting the function, and other information.

There are methods for `print` and `plot` for this class.

Objects of class "fv" are returned by `Fest`, `Gest`, `Jest`, and `Kest` along with many other functions.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

Objects of class "fv" are returned by `Fest`, `Gest`, `Jest`, and `Kest` along with many other functions.

See `plot.fv` for plotting an "fv" object.

See `as.function.fv` to convert an "fv" object to an R function.

Use `cbind.fv` to combine several "fv" objects. Use `bind.fv` to glue additional columns onto an existing "fv" object.

Undocumented functions for modifying an "fv" object include `fvnames`, `fvnames<-`, `tweak.fv.entry` and `rebadge.fv`.

Examples

```
data(cells)
K <- Kest(cells)

class(K)

K # prints a sensible summary

plot(K)
```

G3est

Nearest Neighbour Distance Distribution Function of a Three-Dimensional Point Pattern

Description

Estimates the nearest-neighbour distance distribution function $G_3(r)$ from a three-dimensional point pattern.

Usage

```
G3est(X, ..., rmax = NULL, nrval = 128, correction = c("rs", "km", "Hanisch"))
```

Arguments

X	Three-dimensional point pattern (object of class "pp3").
...	Ignored.
rmax	Optional. Maximum value of argument r for which $G_3(r)$ will be estimated.
nrval	Optional. Number of values of r for which $G_3(r)$ will be estimated. A large value of nrval is required to avoid discretisation effects.
correction	Optional. Character vector specifying the edge correction(s) to be applied. See Details.

Details

For a stationary point process Φ in three-dimensional space, the nearest-neighbour function is

$$G_3(r) = P(d^*(x, \Phi) \leq r \mid x \in \Phi)$$

the cumulative distribution function of the distance $d^*(x, \Phi)$ from a typical point x in Φ to its nearest neighbour, i.e. to the nearest *other* point of Φ .

The three-dimensional point pattern X is assumed to be a partial realisation of a stationary point process Φ . The nearest neighbour function of Φ can then be estimated using techniques described in the References. For each data point, the distance to the nearest neighbour is computed. The empirical cumulative distribution function of these values, with appropriate edge corrections, is the estimate of $G_3(r)$.

The available edge corrections are:

- "rs": the reduced sample (aka minus sampling, border correction) estimator (Baddeley et al, 1993)
- "km": the three-dimensional version of the Kaplan-Meier estimator (Baddeley and Gill, 1997)
- "Hanisch": the three-dimensional generalisation of the Hanisch estimator (Hanisch, 1984).

Value

A function value table (object of class "fv") that can be plotted, printed or coerced to a data frame containing the function values.

Warnings

A large value of `nrvl` is required in order to avoid discretisation effects (due to the use of histograms in the calculation).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rana Moyeed.

References

- Baddeley, A.J., Moyeed, R.A., Howard, C.V. and Boyde, A. (1993) Analysis of a three-dimensional point pattern with replication. *Applied Statistics* **42**, 641–668.
- Baddeley, A.J. and Gill, R.D. (1997) Kaplan-Meier estimators of interpoint distance distributions for spatial point processes. *Annals of Statistics* **25**, 263–292.
- Hanisch, K.-H. (1984) Some remarks on estimators of the distribution function of nearest neighbour distance in stationary spatial point patterns. *Mathematische Operationsforschung und Statistik, series Statistics* **15**, 409–412.

See Also

[F3est](#), [K3est](#), [pcf3est](#)

Examples

```
X <- rpoispp3(42)
Z <- G3est(X)
if(interactive()) plot(Z)
```

ganglia

Beta Ganglion Cells in Cat Retina, Old Version

Description

Point pattern of retinal ganglion cells identified as ‘on’ or ‘off’. A marked point pattern.

Usage

```
data(ganglia)
```

Format

An object of class "ppp" representing the point pattern of cell locations. Entries include

x	Cartesian x -coordinate of cell
y	Cartesian y -coordinate of cell
marks	factor with levels off and on indicating “off” and “on” cells

See [ppp.object](#) for details of the format.

Notes

Important: these data are INCORRECT. See below.

The data represent a pattern of beta-type ganglion cells in the retina of a cat recorded in Figure 6(a) of W\"assle et al. (1981).

The pattern was first analysed by W\"assle et al (1981) using nearest neighbour distances. The data used in their analysis are not available.

The present dataset [ganglia](#) was scanned from Figure 6(a) of W\"assle et al (1981) in the early 1990's, but we have no further information. This dataset is the one analysed by Van Lieshout and Baddeley (1999) using multitype J functions, and by Stoyan (1995) using second order methods (pair correlation and mark correlation).

It has now been discovered that these data are **incorrect**. They are not faithful to the scale in Figure 6 of W\"assle et al (1981), and they contain some scanning errors. Hence they should not be used to address the original scientific question. They have been retained only for comparison with other analyses in the statistical literature.

A new, corrected dataset, scanned from the original microscope image, has been provided under the name [betacells](#). Use that dataset for any further study.

Warnings

These data are incorrect. Use the new corrected dataset [betacells](#).

Source

W\"assle et al (1981), data supplied by Marie-Colette van Lieshout and attributed to Peter Diggle

References

- Stoyan, D. (1995) Personal communication.
- Van Lieshout, M.N.M. and Baddeley, A.J. (1999) Indices of dependence between types in multivariate point patterns. *Scandinavian Journal of Statistics* **26**, 511–532.
- W\"assle, H., Boycott, B. B. & Illing, R.-B. (1981). Morphology and mosaic of on- and off-beta cells in the cat retina and some functional considerations. *Proc. Roy. Soc. London Ser. B* **212**, 177–195.

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,
      trend = ~1, interaction = Poisson(),
      rbord = reach(interaction),
      ppmcorrection="border",
      truecoef = NULL, hi.res = NULL)
```

Arguments

<code>object</code>	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").
<code>r</code>	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.
<code>breaks</code>	Optional alternative to <code>r</code> for advanced use.
<code>correction</code>	Edge correction(s) to be employed in calculating the compensator. Options are "border", "Hanisch" and "best".
<code>conditional</code>	Optional. Logical value indicating whether to compute the estimates for the conditional case. See Details.
<code>restrict</code>	Logical value indicating whether to compute the restriction estimator (<code>restrict=TRUE</code>) or the reweighting estimator (<code>restrict=FALSE</code> , the default). Applies only if <code>conditional=TRUE</code> . See Details.
<code>trend, interaction, rbord</code>	Optional. Arguments passed to <code>ppm</code> to fit a point process model to the data, if <code>object</code> is a point pattern. See <code>ppm</code> for details.
<code>...</code>	Extra arguments passed to <code>ppm</code> .
<code>ppmcorrection</code>	The <code>correction</code> argument to <code>ppm</code> .
<code>truecoef</code>	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 <code>hi.res</code> .
<code>hi.res</code>	Optional. List of parameters passed to <code>quadscheme</code> . 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 $G(r)$,

$$\hat{G}(r) = \frac{\sum_i I\{d_i \leq r\}I\{b_i > r\}}{\sum_i I\{b_i > r\}}$$

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

$$\mathbf{C}\hat{G}(r) = \frac{\int \lambda(u, x)I\{b(u) > r\}I\{d(u, x) \leq r\}}{1 + \sum_i I\{b_i > r\}}$$

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

$$D(r) = \sum_i \frac{I\{x_i \in W_{\ominus d_i}\}I\{d_i \leq r\}}{\text{area}(W_{\ominus d_i})}$$

in which $W_{\ominus r}$ denotes the erosion of the window W by a distance r .

hcom: the corresponding model-compensated function

$$\mathbf{C}G(r) = \int_W \frac{\lambda(u, x)I(u \in W_{\ominus d(u)})I(d(u) \leq r)}{\hat{D}(\infty)\text{area}(W_{\ominus d(u)}) + 1}$$

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 Moller (2011). 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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> Ege Rubak and Jesper Moller.

References

Baddeley, A., Rubak, E. and Moller, J. (2011) Score, pseudo-score and residual diagnostics for spatial point process models. *Statistical Science* **26**, 613–646.

See Also

Related functions: [Gest](#), [Gres](#).

Alternative functions: [Kcom](#), [psstA](#), [psstG](#), [psst](#).

Model fitting: [ppm](#).

Examples

```
data(cells)
fit0 <- ppm(cells, ~1) # uniform Poisson
G0 <- Gcom(fit0)
G0
plot(G0)
# uniform Poisson is clearly not correct

# Hanisch estimates only
plot(Gcom(fit0), cbind(han, hcom) ~ r)

fit1 <- ppm(cells, ~1, Strauss(0.08))
plot(Gcom(fit1), cbind(han, hcom) ~ r)

# Try adjusting interaction distance

fit2 <- update(fit1, Strauss(0.10))
plot(Gcom(fit2), cbind(han, hcom) ~ r)

G3 <- Gcom(cells, interaction=Strauss(0.12))
plot(G3, cbind(han, hcom) ~ r)
```

Description

For a multitype point pattern, estimate the distribution of the distance from a point of type i to the nearest point of type j .

Usage

```
Gcross(X, i, j, r=NULL, breaks=NULL, ..., correction=c("rs", "km", "han"))
```

Arguments

X	The observed point pattern, from which an estimate of the cross type distance distribution function $G_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of <code>marks(X)</code> .
j	The type (mark value) of the points in X to which distances are measured. A character string (or something that will be converted to a character string). Defaults to the second level of <code>marks(X)</code> .
r	Optional. Numeric vector. The values of the argument r at which the distribution function $G_{ij}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r .
breaks	An alternative to the argument r . Not normally invoked by the user. See the Details section.
...	Ignored.
correction	Optional. Character string specifying the edge correction(s) to be used. Options are "none", "rs", "km", "hanisch" and "best".

Details

This function *Gcross* and its companions *Gdot* and *Gmulti* are generalisations of the function *Gest* to multitype point patterns.

A multitype point pattern is a spatial pattern of points classified into a finite number of possible “colours” or “types”. In the **spatstat** package, a multitype pattern is represented as a single point pattern object in which the points carry marks, and the mark value attached to each point determines the type of that point.

The argument *X* must be a point pattern (object of class “ppp”) or any data that are acceptable to *as.ppp*. It must be a marked point pattern, and the mark vector *X\$marks* must be a factor. The arguments *i* and *j* will be interpreted as levels of the factor *X\$marks*. (Warning: this means that an integer value *i*=3 will be interpreted as the number 3, **not** the 3rd smallest level).

The “cross-type” (type *i* to type *j*) nearest neighbour distance distribution function of a multitype point process is the cumulative distribution function $G_{ij}(r)$ of the distance from a typical random point of the process with type *i* the nearest point of type *j*.

An estimate of $G_{ij}(r)$ is a useful summary statistic in exploratory data analysis of a multitype point pattern. If the process of type *i* points were independent of the process of type *j* points, then $G_{ij}(r)$ would equal $F_j(r)$, the empty space function of the type *j* points. For a multitype Poisson point process where the type *i* points have intensity λ_i , we have

$$G_{ij}(r) = 1 - e^{-\lambda_j \pi r^2}$$

Deviations between the empirical and theoretical G_{ij} curves may suggest dependence between the points of types *i* and *j*.

This algorithm estimates the distribution function $G_{ij}(r)$ from the point pattern X . It assumes that X can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in X as $X>window$) may have arbitrary shape. Biases due to edge effects are treated in the same manner as in [Gest](#).

The argument r is the vector of values for the distance r at which $G_{ij}(r)$ should be evaluated. It is also used to determine the breakpoints (in the sense of [hist](#)) for the computation of histograms of distances. The reduced-sample and Kaplan-Meier estimators are computed from histogram counts. In the case of the Kaplan-Meier estimator this introduces a discretisation error which is controlled by the fineness of the breakpoints.

First-time users would be strongly advised not to specify r . However, if it is specified, r must satisfy $r[1] = 0$, and $\max(r)$ must be larger than the radius of the largest disc contained in the window. Furthermore, the successive entries of r must be finely spaced.

The algorithm also returns an estimate of the hazard rate function, $\lambda(r)$, of $G_{ij}(r)$. This estimate should be used with caution as $G_{ij}(r)$ is not necessarily differentiable.

The naive empirical distribution of distances from each point of the pattern X to the nearest other point of the pattern, is a biased estimate of G_{ij} . However this is also returned by the algorithm, as it is sometimes useful in other contexts. Care should be taken not to use the uncorrected empirical G_{ij} as if it were an unbiased estimator of G_{ij} .

Value

An object of class "fv" (see [fv.object](#)).

Essentially a data frame containing six numeric columns

r	the values of the argument r at which the function $G_{ij}(r)$ has been estimated
rs	the "reduced sample" or "border correction" estimator of $G_{ij}(r)$
han	the Hanisch-style estimator of $G_{ij}(r)$
km	the spatial Kaplan-Meier estimator of $G_{ij}(r)$
$hazard$	the hazard rate $\lambda(r)$ of $G_{ij}(r)$ by the spatial Kaplan-Meier method
raw	the uncorrected estimate of $G_{ij}(r)$, i.e. the empirical distribution of the distances from each point of type i to the nearest point of type j
$theo$	the theoretical value of $G_{ij}(r)$ for a marked Poisson process with the same estimated intensity (see below).

Warnings

The arguments i and j are always interpreted as levels of the factor Xmarks$. They are converted to character strings if they are not already character strings. The value $i=1$ does **not** refer to the first level of the factor.

The function G_{ij} does not necessarily have a density.

The reduced sample estimator of G_{ij} is pointwise approximately unbiased, but need not be a valid distribution function; it may not be a nondecreasing function of r . Its range is always within $[0, 1]$.

The spatial Kaplan-Meier estimator of G_{ij} is always nondecreasing but its maximum value may be less than 1.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.
- Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.
- Diggle, P. J. (1986). Displaced amacrine cells in the retina of a rabbit : analysis of a bivariate spatial point pattern. *J. Neurosci. Meth.* **18**, 115–125.
- Harkness, R.D and Isham, V. (1983) A bivariate spatial point pattern of ants' nests. *Applied Statistics* **32**, 293–303
- Lotwick, H. W. and Silverman, B. W. (1982). Methods for analysing spatial processes of several types of points. *J. Royal Statist. Soc. Ser. B* **44**, 406–413.
- Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.
- Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.
- Van Lieshout, M.N.M. and Baddeley, A.J. (1999) Indices of dependence between types in multivariate point patterns. *Scandinavian Journal of Statistics* **26**, 511–532.

See Also

[Gdot](#), [Gest](#), [Gmulti](#)

Examples

```

data(betacells)
# cat retina data
G01 <- Gcross(betacells)

# equivalent to:
## Not run:
G01 <- Gcross(betacells, "off", "on")

## End(Not run)

plot(G01)

# empty space function of 'on' points
## Not run:
F1 <- Fest(split(betacells)$on, r = G01$r, eps=10.0)
lines(F1$r, F1$km, lty=3)

## End(Not run)

# synthetic example
pp <- runifpoispp(30)
pp <- pp %mark% factor(sample(0:1, pp$n, replace=TRUE))
G <- Gcross(pp, "0", "1") # note: "0" not 0

```

Gdot*Multitype Nearest Neighbour Distance Function (i-to-any)*

Description

For a multitype point pattern, estimate the distribution of the distance from a point of type i to the nearest other point of any type.

Usage

```
Gdot(X, i, r=NULL, breaks=NULL, ..., correction=c("km", "rs", "han"))
```

Arguments

X	The observed point pattern, from which an estimate of the distance distribution function $G_{i\bullet}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of marks(X).
r	Optional. Numeric vector. The values of the argument r at which the distribution function $G_{i\bullet}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r .
breaks	An alternative to the argument r. Not normally invoked by the user. See the Details section.
...	Ignored.
correction	Optional. Character string specifying the edge correction(s) to be used. Options are "none", "rs", "km", "hanisch" and "best".

Details

This function `Gdot` and its companions `Gcross` and `Gmulti` are generalisations of the function `Gest` to multitype point patterns.

A multitype point pattern is a spatial pattern of points classified into a finite number of possible “colours” or “types”. In the **spatstat** package, a multitype pattern is represented as a single point pattern object in which the points carry marks, and the mark value attached to each point determines the type of that point.

The argument `X` must be a point pattern (object of class “`ppp`”) or any data that are acceptable to `as.ppp`. It must be a marked point pattern, and the mark vector `X$marks` must be a factor. The argument will be interpreted as a level of the factor `X$marks`. (Warning: this means that an integer value `i=3` will be interpreted as the number 3, **not** the 3rd smallest level.)

The “dot-type” (type i to any type) nearest neighbour distance distribution function of a multitype point process is the cumulative distribution function $G_{i\bullet}(r)$ of the distance from a typical random point of the process with type i the nearest other point of the process, regardless of type.

An estimate of $G_{i\bullet}(r)$ is a useful summary statistic in exploratory data analysis of a multitype point pattern. If the type i points were independent of all other points, then $G_{i\bullet}(r)$ would equal $G_{ii}(r)$,

the nearest neighbour distance distribution function of the type i points alone. For a multitype Poisson point process with total intensity λ , we have

$$G_{i\bullet}(r) = 1 - e^{-\lambda\pi r^2}$$

Deviations between the empirical and theoretical $G_{i\bullet}$ curves may suggest dependence of the type i points on the other points.

This algorithm estimates the distribution function $G_{i\bullet}(r)$ from the point pattern X . It assumes that X can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in X as $X>window$) may have arbitrary shape. Biases due to edge effects are treated in the same manner as in [Gest](#).

The argument r is the vector of values for the distance r at which $G_{i\bullet}(r)$ should be evaluated. It is also used to determine the breakpoints (in the sense of [hist](#)) for the computation of histograms of distances. The reduced-sample and Kaplan-Meier estimators are computed from histogram counts. In the case of the Kaplan-Meier estimator this introduces a discretisation error which is controlled by the fineness of the breakpoints.

First-time users would be strongly advised not to specify r . However, if it is specified, r must satisfy $r[1] = 0$, and $\max(r)$ must be larger than the radius of the largest disc contained in the window. Furthermore, the successive entries of r must be finely spaced.

The algorithm also returns an estimate of the hazard rate function, $\lambda(r)$, of $G_{i\bullet}(r)$. This estimate should be used with caution as $G_{i\bullet}(r)$ is not necessarily differentiable.

The naive empirical distribution of distances from each point of the pattern X to the nearest other point of the pattern, is a biased estimate of $G_{i\bullet}$. However this is also returned by the algorithm, as it is sometimes useful in other contexts. Care should be taken not to use the uncorrected empirical $G_{i\bullet}$ as if it were an unbiased estimator of $G_{i\bullet}$.

Value

An object of class "fv" (see [fv.object](#)).

Essentially a data frame containing six numeric columns

r	the values of the argument r at which the function $G_{i\bullet}(r)$ has been estimated
rs	the "reduced sample" or "border correction" estimator of $G_{i\bullet}(r)$
han	the Hanisch-style estimator of $G_{i\bullet}(r)$
km	the spatial Kaplan-Meier estimator of $G_{i\bullet}(r)$
$hazard$	the hazard rate $\lambda(r)$ of $G_{i\bullet}(r)$ by the spatial Kaplan-Meier method
raw	the uncorrected estimate of $G_{i\bullet}(r)$, i.e. the empirical distribution of the distances from each point of type i to the nearest other point of any type.
$theo$	the theoretical value of $G_{i\bullet}(r)$ for a marked Poisson process with the same estimated intensity (see below).

Warnings

The argument i is interpreted as a level of the factor Xmarks$. It is converted to a character string if it is not already a character string. The value $i=1$ does **not** refer to the first level of the factor.

The function $G_{i\bullet}$ does not necessarily have a density.

The reduced sample estimator of $G_{i\bullet}$ is pointwise approximately unbiased, but need not be a valid distribution function; it may not be a nondecreasing function of r . Its range is always within $[0, 1]$.

The spatial Kaplan-Meier estimator of $G_{i\bullet}$ is always nondecreasing but its maximum value may be less than 1.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

References

- Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.
- Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.
- Diggle, P. J. (1986). Displaced amacrine cells in the retina of a rabbit : analysis of a bivariate spatial point pattern. *J. Neurosci. Meth.* **18**, 115–125.
- Harkness, R.D and Isham, V. (1983) A bivariate spatial point pattern of ants' nests. *Applied Statistics* **32**, 293–303
- Lotwick, H. W. and Silverman, B. W. (1982). Methods for analysing spatial processes of several types of points. *J. Royal Statist. Soc. Ser. B* **44**, 406–413.
- Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.
- Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.
- Van Lieshout, M.N.M. and Baddeley, A.J. (1999) Indices of dependence between types in multivariate point patterns. *Scandinavian Journal of Statistics* **26**, 511–532.

See Also

[Gcross](#), [Gest](#), [Gmulti](#)

Examples

```
data(betacells)
# cat retina data
G0. <- Gdot(betacells, "off")
plot(G0.)

# synthetic example
pp <- runifpoispp(30)
pp <- pp %mark% factor(sample(0:1, npoints(pp), replace=TRUE))
G <- Gdot(pp, "0")
G <- Gdot(pp, 0) # equivalent
```

Description

Estimates the nearest neighbour distance distribution function $G(r)$ from a point pattern in a window of arbitrary shape.

Usage

```
Gest(X, r=NULL, breaks=NULL, ..., correction=c("rs", "km", "han"))
```

Arguments

X	The observed point pattern, from which an estimate of $G(r)$ will be computed. An object of class <code>ppp</code> , or data in any format acceptable to <code>as.ppp()</code> .
r	Optional. Numeric vector. The values of the argument r at which $G(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r .
breaks	An alternative to the argument <code>r</code> . Not normally invoked by the user. See the Details section.
...	Ignored.
correction	Optional. The edge correction(s) to be used to estimate $G(r)$. A vector of character strings selected from "none", "rs", "km", "Hanisch" and "best".

Details

The nearest neighbour distance distribution function (also called the “*event-to-event*” or “*inter-event*” distribution) of a point process X is the cumulative distribution function G of the distance from a typical random point of X to the nearest other point of X .

An estimate of G derived from a spatial point pattern dataset can be used in exploratory data analysis and formal inference about the pattern (Cressie, 1991; Diggle, 1983; Ripley, 1988). In exploratory analyses, the estimate of G is a useful statistic summarising one aspect of the “clustering” of points. For inferential purposes, the estimate of G is usually compared to the true value of G for a completely random (Poisson) point process, which is

$$G(r) = 1 - e^{-\lambda\pi r^2}$$

where λ is the intensity (expected number of points per unit area). Deviations between the empirical and theoretical G curves may suggest spatial clustering or spatial regularity.

This algorithm estimates the nearest neighbour distance distribution function G from the point pattern `X`. It assumes that `X` can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in `X` as `X>window`) may have arbitrary shape.

The argument `X` is interpreted as a point pattern object (of class "ppp", see `ppp.object`) and can be supplied in any of the formats recognised by `as.ppp()`.

The estimation of G is hampered by edge effects arising from the unobservability of points of the random pattern outside the window. An edge correction is needed to reduce bias (Baddeley, 1998; Ripley, 1988). The edge corrections implemented here are the border method or “*reduced sample*” estimator, the spatial Kaplan-Meier estimator (Baddeley and Gill, 1997) and the Hanisch estimator (Hanisch, 1984).

The argument `r` is the vector of values for the distance r at which $G(r)$ should be evaluated. It is also used to determine the breakpoints (in the sense of `hist`) for the computation of histograms of distances. The estimators are computed from histogram counts. This introduces a discretisation error which is controlled by the fineness of the breakpoints.

First-time users would be strongly advised not to specify `r`. However, if it is specified, `r` must satisfy `r[1] = 0`, and `max(r)` must be larger than the radius of the largest disc contained in the window. Furthermore, the successive entries of `r` must be finely spaced.

The algorithm also returns an estimate of the hazard rate function, $\lambda(r)$, of $G(r)$. The hazard rate is defined as the derivative

$$\lambda(r) = -\frac{d}{dr} \log(1 - G(r))$$

This estimate should be used with caution as G is not necessarily differentiable.

The naive empirical distribution of distances from each point of the pattern X to the nearest other point of the pattern, is a biased estimate of G . However it is sometimes useful. It can be returned by the algorithm, by selecting `correction="none"`. Care should be taken not to use the uncorrected empirical G as if it were an unbiased estimator of G .

To simply compute the nearest neighbour distance for each point in the pattern, use `nndist`. To determine which point is the nearest neighbour of a given point, use `nnwhich`.

Value

An object of class "fv", see `fv.object`, which can be plotted directly using `plot.fv`.

Essentially a data frame containing some or all of the following columns:

<code>r</code>	the values of the argument r at which the function $G(r)$ has been estimated
<code>rs</code>	the "reduced sample" or "border correction" estimator of $G(r)$
<code>km</code>	the spatial Kaplan-Meier estimator of $G(r)$
<code>hazard</code>	the hazard rate $\lambda(r)$ of $G(r)$ by the spatial Kaplan-Meier method
<code>raw</code>	the uncorrected estimate of $G(r)$, i.e. the empirical distribution of the distances from each point in the pattern X to the nearest other point of the pattern
<code>han</code>	the Hanisch correction estimator of $G(r)$
<code>theo</code>	the theoretical value of $G(r)$ for a stationary Poisson process of the same estimated intensity.

Warnings

The function G does not necessarily have a density. Any valid c.d.f. may appear as the nearest neighbour distance distribution function of a stationary point process.

The reduced sample estimator of G is pointwise approximately unbiased, but need not be a valid distribution function; it may not be a nondecreasing function of r . Its range is always within $[0, 1]$.

The spatial Kaplan-Meier estimator of G is always nondecreasing but its maximum value may be less than 1.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Baddeley, A.J. Spatial sampling and censoring. In O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*. Chapman and Hall, 1998. Chapter 2, pages 37-78.
- Baddeley, A.J. and Gill, R.D. Kaplan-Meier estimators of interpoint distance distributions for spatial point processes. *Annals of Statistics* **25** (1997) 263-292.
- Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.
- Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.
- Hanisch, K.-H. (1984) Some remarks on estimators of the distribution function of nearest-neighbour distance in stationary spatial point patterns. *Mathematische Operationsforschung und Statistik, series Statistics* **15**, 409–412.

- Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.
- Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.

See Also

[nndist](#), [nnwhich](#), [Fest](#), [Jest](#), [Kest](#), [km.rs](#), [reduced.sample](#), [kaplan.meier](#)

Examples

```
data(cells)
G <- Gest(cells)
plot(G)

# P-P style plot
plot(G, cbind(km, theo) ~ theo)

# the empirical G is below the Poisson G,
# indicating an inhibited pattern

## Not run:
plot(G, . ~ r)
plot(G, . ~ theo)
plot(G, asin(sqrt(.)) ~ asin(sqrt(theo)))

## End(Not run)
```

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 positive 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 . 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`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

Geyer, C.J. (1999) Likelihood Inference for Spatial Point Processes. Chapter 3 in O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. Van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*, Chapman and Hall / CRC, Monographs on Statistics and Applied Probability, number 80. Pages 79–140.

See Also

[ppm](#), [pairwise.family](#), [ppm.object](#), [Strauss](#), [SatPiece](#)

Examples

```
data(cells)
ppm(cells, ~1, Geyer(r=0.07, sat=2))
# fit the stationary saturation process to 'cells'
```

GfoxFoxall's Distance Functions

Description

Given a point pattern X and a spatial object Y , compute estimates of Foxall's G and J functions.

Usage

```
Gfox(X, Y, r = NULL, breaks = NULL, correction = c("km", "rs", "han"), ...)  
Jfox(X, Y, r = NULL, breaks = NULL, correction = c("km", "rs", "han"), ...)
```

Arguments

X	A point pattern (object of class "ppp") from which distances will be measured.
Y	An object of class "ppp", "psp" or "owin" to which distances will be measured.
r	Optional. Numeric vector. The values of the argument r at which $G_{\text{fox}}(r)$ or $J_{\text{fox}}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r .
<code>breaks</code>	An alternative to the argument r . Not normally invoked by the user.
<code>correction</code>	Optional. The edge correction(s) to be used to estimate $G_{\text{fox}}(r)$ or $J_{\text{fox}}(r)$. A vector of character strings selected from "none", "rs", "km", "cs" and "best".
...	Extra arguments affecting the discretisation of distances. These arguments are ignored by Gfox, but Jfox passes them to Hest to determine the discretisation of the spatial domain.

Details

Given a point pattern X and another spatial object Y , these functions compute two nonparametric measures of association between X and Y , introduced by Foxall (Foxall and Baddeley, 2002).

Let the random variable R be the distance from a typical point of X to the object Y . Foxall's G -function is the cumulative distribution function of R :

$$G(r) = P(R \leq r)$$

Let the random variable S be the distance from a *fixed* point in space to the object Y . The cumulative distribution function of S is the (unconditional) spherical contact distribution function

$$H(r) = P(S \leq r)$$

which is computed by [Hest](#).

Foxall's J -function is the ratio

$$J(r) = \frac{1 - G(r)}{1 - H(r)}$$

For further interpretation, see Foxall and Baddeley (2002).

Accuracy of Jfox depends on the pixel resolution, which is controlled by the arguments `eps`, `dimyx` and `xy` passed to [as.mask](#). For example, use `eps=0.1` to specify square pixels of side 0.1 units, and `dimyx=256` to specify a 256 by 256 grid of pixels.

Value

A function value table (object of class "fv") which can be printed, plotted, or converted to a data frame of values.

Author(s)

Rob Foxall and Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

References

Foxall, R. and Baddeley, A. (2002) Nonparametric measures of association between a spatial point process and a random set, with geological applications. *Applied Statistics* **51**, 165–182.

See Also

[Gest](#), [Hest](#), [Jest](#), [Fest](#)

Examples

```
data(copper)
X <- copper$SouthPoints
Y <- copper$SouthLines
G <- Gfox(X,Y)
J <- Jfox(X,Y, correction="km")

## Not run:
J <- Jfox(X,Y, correction="km", eps=0.25)

## End(Not run)
```

Description

For a marked point pattern, estimate the distribution of the distance from a typical point in subset I to the nearest point of subset J.

Usage

```
Gmulti(X, I, J, r=NULL, breaks=NULL, ...,
       disjoint=NULL, correction=c("rs", "km", "han"))
```

Arguments

- X The observed point pattern, from which an estimate of the multitype distance distribution function $G_{IJ}(r)$ will be computed. It must be a marked point pattern. See under Details.
- I Subset of points of X from which distances are measured.
- J Subset of points in X to which distances are measured.

r	Optional. Numeric vector. The values of the argument r at which the distribution function $G_{IJ}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r .
breaks	An alternative to the argument r . Not normally invoked by the user. See the Details section.
...	Ignored.
disjoint	Optional flag indicating whether the subsets I and J are disjoint. If missing, this value will be computed by inspecting the vectors I and J.
correction	Optional. Character string specifying the edge correction(s) to be used. Options are "none", "rs", "km", "hanisch" and "best".

Details

The function **Gmulti** generalises **Gest** (for unmarked point patterns) and **Gdot** and **Gcross** (for multitype point patterns) to arbitrary marked point patterns.

Suppose X_I, X_J are subsets, possibly overlapping, of a marked point process. This function computes an estimate of the cumulative distribution function $G_{IJ}(r)$ of the distance from a typical point of X_I to the nearest distinct point of X_J .

The argument **X** must be a point pattern (object of class "ppp") or any data that are acceptable to **as.ppp**.

The arguments **I** and **J** specify two subsets of the point pattern. They may be any type of subset indices, for example, logical vectors of length equal to **npoints(X)**, or integer vectors with entries in the range 1 to **npoints(X)**, or negative integer vectors.

Alternatively, **I** and **J** may be **functions** that will be applied to the point pattern **X** to obtain index vectors. If **I** is a function, then evaluating **I(X)** should yield a valid subset index. This option is useful when generating simulation envelopes using **envelope**.

This algorithm estimates the distribution function $G_{IJ}(r)$ from the point pattern **X**. It assumes that **X** can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in **X** as **X>window**) may have arbitrary shape. Biases due to edge effects are treated in the same manner as in **Gest**.

The argument **r** is the vector of values for the distance r at which $G_{IJ}(r)$ should be evaluated. It is also used to determine the breakpoints (in the sense of **hist**) for the computation of histograms of distances. The reduced-sample and Kaplan-Meier estimators are computed from histogram counts. In the case of the Kaplan-Meier estimator this introduces a discretisation error which is controlled by the fineness of the breakpoints.

First-time users would be strongly advised not to specify **r**. However, if it is specified, **r** must satisfy **r[1] = 0**, and **max(r)** must be larger than the radius of the largest disc contained in the window. Furthermore, the successive entries of **r** must be finely spaced.

The algorithm also returns an estimate of the hazard rate function, $\lambda(r)$, of $G_{IJ}(r)$. This estimate should be used with caution as $G_{IJ}(r)$ is not necessarily differentiable.

The naive empirical distribution of distances from each point of the pattern **X** to the nearest other point of the pattern, is a biased estimate of G_{IJ} . However this is also returned by the algorithm, as it is sometimes useful in other contexts. Care should be taken not to use the uncorrected empirical G_{IJ} as if it were an unbiased estimator of G_{IJ} .

Value

An object of class "fv" (see [fv.object](#)).

Essentially a data frame containing six numeric columns

r	the values of the argument r at which the function $G_{IJ}(r)$ has been estimated
rs	the "reduced sample" or "border correction" estimator of $G_{IJ}(r)$
han	the Hanisch-style estimator of $G_{IJ}(r)$
km	the spatial Kaplan-Meier estimator of $G_{IJ}(r)$
hazard	the hazard rate $\lambda(r)$ of $G_{IJ}(r)$ by the spatial Kaplan-Meier method
raw	the uncorrected estimate of $G_{IJ}(r)$, i.e. the empirical distribution of the distances from each point of type i to the nearest point of type j
theo	the theoretical value of $G_{IJ}(r)$ for a marked Poisson process with the same estimated intensity

Warnings

The function G_{IJ} does not necessarily have a density.

The reduced sample estimator of G_{IJ} is pointwise approximately unbiased, but need not be a valid distribution function; it may not be a nondecreasing function of r . Its range is always within $[0, 1]$.

The spatial Kaplan-Meier estimator of G_{IJ} is always nondecreasing but its maximum value may be less than 1.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.
- Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.
- Diggle, P. J. (1986). Displaced amacrine cells in the retina of a rabbit : analysis of a bivariate spatial point pattern. *J. Neurosci. Meth.* **18**, 115–125.
- Harkness, R.D and Isham, V. (1983) A bivariate spatial point pattern of ants' nests. *Applied Statistics* **32**, 293–303
- Lotwick, H. W. and Silverman, B. W. (1982). Methods for analysing spatial processes of several types of points. *J. Royal Statist. Soc. Ser. B* **44**, 406–413.
- Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.
- Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.
- Van Lieshout, M.N.M. and Baddeley, A.J. (1999) Indices of dependence between types in multivariate point patterns. *Scandinavian Journal of Statistics* **26**, 511–532.

See Also

[Gcross](#), [Gdot](#), [Gest](#)

Examples

```
data(longleaf)
# Longleaf Pine data: marks represent diameter

Gm <- Gmulti(longleaf, longleaf$marks <= 15, longleaf$marks >= 25)
plot(Gm)
```

gorillas

Gorilla Nesting Sites

Description

Locations of nesting sites of gorillas, and associated covariates, in a National Park in Cameroon.

Usage

```
data(gorillas)
```

Format

`gorillas` is a marked point pattern (object of class "ppp") representing nest site locations.

`gorillas.extra` is a named list of 7 pixel images (objects of class "im") containing spatial covariates. It also belongs to the class "listof".

All spatial coordinates are in metres. The coordinate reference system is WGS_84_UTM_Zone_32N.

Details

These data come from a study of gorillas in the Kagwene Gorilla Sanctuary, Cameroon, by the Wildlife Conservation Society Takamanda-Mone Landscape Project (WCS-TMLP). A detailed description and analysis of the data is reported in Funwi-Gabga and Mateu (2012).

The dataset `gorillas` is a marked point pattern (object of class "ppp") giving the spatial locations of 647 nesting sites of gorilla groups observed in the sanctuary over time. Locations are given as UTM (Zone 32N) coordinates in metres. The observation window is the boundary of the sanctuary, represented as a polygon. Marks attached to the points are:

group Identifier of the gorilla group that constructed the nest site: a categorical variable with values `major` or `minor`.

season Season in which data were collected: categorical, either `rainy` or `dry`.

date Day of observation. A value of class "Date".

The accompanying dataset `gorillas.extra` contains spatial covariate information. It is a named list containing seven pixel images (objects of class "im") giving the values of seven covariates over the study region. It also belongs to the class "listof" so that it can be plotted. The component images are:

aspect Compass direction of the terrain slope. Categorical, with levels N, NE, E, SE, S, SW, W and NW.

elevation Digital elevation of terrain, in metres.

heat Heat Load Index at each point on the surface (Beer's aspect), discretised. Categorical with values Warmest (Beer's aspect between 0 and 0.999), Moderate (Beer's aspect between 1 and 1.999), Coolest (Beer's aspect equals 2).

slopeangle Terrain slope, in degrees.

slopetype Type of slope. Categorical, with values Valley, Toe (toe slope), Flat, Midslope, Upper and Ridge.

vegetation Vegetation or cover type. Categorical, with values Disturbed (highly disturbed forest), Colonising (colonising forest), Grassland (savannah), Primary (primary forest), Secondary (secondary forest), and Transition (transitional vegetation).

waterdist Euclidean distance from nearest water body, in metres.

For further information see Funwi-Gabga and Mateu (2012).

Source

Field data collector: Wildlife Conservation Society Takamanda-Mone Landscape Project (WCS-TMLP). Please acknowledge WCS-TMLP in any use of these data.

Data kindly provided by Funwi-Gabga Neba, Data Coordinator of A.P.E.S. Database Project, Department of Primatology, Max Planck Institute for Evolutionary Anthropology, Leipzig, Germany.

The collaboration of Prof Jorge Mateu, Universitat Jaume I, Castellon, Spain is gratefully acknowledged.

References

Funwi-Gabga, N. (2008) *A pastoralist survey and fire impact assessment in the Kagwene Gorilla Sanctuary, Cameroon*. M.Sc. thesis, Geology and Environmental Science, University of Buea.

Funwi-Gabga, N. and Mateu, J. (2012) Understanding the nesting spatial behaviour of gorillas in the Kagwene Sanctuary, Cameroon. *Stochastic Environmental Research and Risk Assessment*, in press.

Examples

```
summary(gorillas)
plot(gorillas)
plot(gorillas.extra)
```

gpc2owin

Convert Polygonal Region into Different Format

Description

Conversion between the different representations of a polygonal region in the packages **spatstat** and **gpclib**.

Usage

```
gpc2owin(x)
owin2gpc(x)
```

Arguments

- | | |
|---|---|
| x | Object representing a polygonal region. An object of class "owin" in the spatstat package (for owin2gpc) or an object of class "gpc.poly" in the gpclib package (for gpc2owin). |
|---|---|

Details

The packages **spatstat** and **gpclib** have slightly different internal formats for representing a polygonal region in the two-dimensional plane. In **gpclib** a polygonal region is an object of class "gpc.poly", while in **spatstat** it is an object of class "owin" and of type "polygonal".

These two functions convert the two formats: owin2gpc converts an "owin" to a "gpc.poly", while gpc2owin does the reverse.

Conversion of a "gpc.poly" to an "owin" can also be performed by calling [as.owin](#).

Value

An object of class "owin" in the **spatstat** package (for gpc2owin) or an object of class "gpc.poly" in the **gpclib** package (for owin2gpc).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[as.owin](#)

Examples

```
data(letterR)
if(spatstat.options("gpclib") && require(gpclib)) {
  R <- owin2gpc(letterR)
  L <- gpc2owin(R)
} else cat("gpclib is not available\n")
```

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

<code>object</code>	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 .
<code>...</code>	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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> Ege Rubak and Jesper Moller.

References

Baddeley, A., Rubak, E. and Moller, J. (2011) Score, pseudo-score and residual diagnostics for spatial point process models. *Statistical Science* **26**, 613–646.

See Also

Related functions: `Gcom`, `Gest`.

Alternative functions: `Kres`, `psstA`, `psstG`, `psst`.

Model-fitting: `ppm`.

Examples

```
data(cells)
fit0 <- ppm(cells, ~1) # uniform Poisson
G0 <- Gres(fit0)
plot(G0)
# Hanisch correction estimate
plot(G0, hres ~ r)
# uniform Poisson is clearly not correct

fit1 <- ppm(cells, ~1, Strauss(0.08))
plot(Gres(fit1), hres ~ r)
# fit looks approximately OK; try adjusting interaction distance

plot(Gres(cells, interaction=Strauss(0.12)))

# How to make envelopes
## Not run:
E <- envelope(fit1, Gres, interaction=as.interact(fit1), nsim=39)
plot(E)

## End(Not run)
```

```
# For computational efficiency
Gc <- Gcom(fit1)
G1 <- Gres(Gc)
```

gridcentres

Rectangular grid of points

Description

Generates a rectangular grid of points in a window

Usage

```
gridcentres(window, nx, ny)
```

Arguments

window	A window. An object of class owin , or data in any format acceptable to as.owin() .
nx	Number of points in each row of the rectangular grid.
ny	Number of points in each column of the rectangular grid.

Details

This function creates a rectangular grid of points in the window.

The bounding rectangle of the window is divided into a regular $nx \times ny$ grid of rectangular tiles. The function returns the x, y coordinates of the centres of these tiles.

Note that some of these grid points may lie outside the window, if `window` is not of type "rectangle". The function [inside.owin](#) can be used to select those grid points which do lie inside the window. See the examples.

This function is useful in creating dummy points for quadrature schemes (see [quadscheme](#)) and for other miscellaneous purposes.

Value

A list with two components `x` and `y`, which are numeric vectors giving the coordinates of the points of the rectangular grid.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[quad.object](#), [quadscheme](#), [inside.owin](#), [stratrand](#)

Examples

```
w <- unit.square()
xy <- gridcentres(w, 10,15)
## Not run:
plot(w)
points(xy)

## End(Not run)

bdry <- list(x=c(0.1,0.3,0.7,0.4,0.2),
              y=c(0.1,0.1,0.5,0.7,0.3))
w <- owin(c(0,1), c(0,1), poly=bdry)
xy <- gridcentres(w, 30, 30)
ok <- inside.owin(xy$x, xy$y, w)
## Not run:
plot(w)
points(xy$x[ok], xy$y[ok])

## End(Not run)
```

gridweights

Compute Quadrature Weights Based on Grid Counts

Description

Computes quadrature weights for a given set of points, using the “counting weights” for a grid of rectangular tiles.

Usage

```
gridweights(X, ntile, ..., window=NULL, verbose=FALSE, npix=NULL, areas=NULL)
```

Arguments

X	Data defining a point pattern.
ntile	Number of tiles in each row and column of the rectangular grid. An integer vector of length 1 or 2.
...	Ignored.
window	Default window for the point pattern
verbose	Logical flag. If TRUE, information will be printed about the computation of the grid weights.
npix	Dimensions of pixel grid to use when computing a digital approximation to the tile areas.
areas	Vector of areas of the tiles, if they are already known.

Details

This function computes a set of quadrature weights for a given pattern of points (typically comprising both “data” and ‘dummy’ points). See [quad.object](#) for an explanation of quadrature weights and quadrature schemes.

The weights are computed by the “counting weights” rule based on a regular grid of rectangular tiles. First X and (optionally) $window$ are converted into a point pattern object. Then the bounding rectangle of the window of the point pattern is divided into a regular $ntile[1] \times ntile[2]$ grid of rectangular tiles. The weight attached to a point of X is the area of the tile in which it lies, divided by the number of points of X lying in that tile.

For non-rectangular windows the tile areas are currently calculated by approximating the window as a binary mask. The accuracy of this approximation is controlled by $npix$, which becomes the argument $dimyx$ of [as.mask](#).

Value

Vector of nonnegative weights for each point in X .

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[quad.object](#), [dirichlet.weights](#)

Examples

```
Q <- quadscheme(runifpoispp(10))
X <- as.ppp(Q) # data and dummy points together
w <- gridweights(X, 10)
w <- gridweights(X, c(10, 10))
```

hamster

Aherne's hamster tumour data

Description

Point pattern of cell nuclei in hamster kidney, each nucleus classified as either ‘dividing’ or ‘pyknotic’. A multitype point pattern.

Usage

`data(hamster)`

Format

An object of class “`ppp`” representing the point pattern of cell locations. Entries include

<code>x</code>	Cartesian x -coordinate of cell
<code>y</code>	Cartesian y -coordinate of cell
<code>marks</code>	factor with levels “dividing” and “pyknotic”.

See [ppp.object](#) for details of the format.

Notes

These data were presented and analysed by Diggle (1983, section 7.3).

The data give the positions of the centres of the nuclei of certain cells in a histological section of tissue from a laboratory-induced metastasising lymphoma in the kidney of a hamster.

The nuclei are classified as either "pyknotic" (corresponding to dying cells) or "dividing" (corresponding to cells arrested in metaphase, i.e. in the act of dividing). The background void is occupied by unrecorded, interphase cells in relatively large numbers.

The sampling window is a square, originally about 0.25 mm square in real units, which has been rescaled to the unit square.

Source

Dr W. A. Aherne, Department of Pathology, University of Newcastle-upon-Tyne, UK. Data supplied by Prof. Peter Diggle

References

Diggle, P.J. (1983) *Statistical analysis of spatial point patterns*. Academic Press.

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)`

Arguments

`hc` The hard core distance

Details

A hard core process with hard core distance $h < r$ 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

$$f(x_1, \dots, x_n) = \alpha \beta^{n(x)}$$

where x_1, \dots, 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.

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.
 Ripley, B.D. (1981) *Spatial statistics*. John Wiley and Sons.

See Also

[Strauss](#), [StraussHard](#), [MultiHard](#), [ppm](#), [pairwise.family](#), [ppm.object](#)

Examples

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

data(cells)

## Not run:
ppm(cells, ~1, Hardcore(0.05))
# fit the stationary hard core process to 'cells'

## End(Not run)

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

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`ppm`

Examples

```
data(longleaf)
X <- unmark(longleaf)
# inhomogeneous point pattern

# 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 * (fit.3$maxlogpl - fit.h$maxlogpl)
x <- X$x
y <- X$y
df <- ncol(polynomial(x,y,3)) - ncol(harmonic(x,y,3))
pval <- 1 - pchisq(lrts, df=df)
```

harmonise.im

*Make Pixel Images Compatible***Description**

Convert several pixel images to a common pixel raster.

Usage

```
harmonise.im(...)
harmonize.im(...)
```

Arguments

...	Any number of pixel images (objects of class "im") or data which can be converted to pixel images by as.im .
-----	--

Details

This function makes any number of pixel images compatible, by converting them all to a common pixel grid.

At least one of the arguments ... must be a pixel image. Some arguments may be windows (objects of class "owin"), functions (`function(x,y)`) or numerical constants. These will be converted to images using [as.im](#).

The common pixel grid is determined by inspecting all the pixel images in the argument list, computing the bounding box of all the images, then finding the image with the highest spatial resolution, and extending its pixel grid to cover the bounding box.

The return value is a list with entries corresponding to the input arguments. If the arguments were named (name=value) then the return value also carries these names.

If you just want to determine the appropriate pixel resolution, without converting the images, use [commonGrid](#).

Value

A list, of length equal to the number of arguments ..., whose entries are pixel images.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[commonGrid](#), [compatible.im](#), [as.im](#)

Examples

```
A <- setcov(square(1))
B <- function(x,y) { x }
G <- density(runifpoint(42))
harmonise.im(X=A, Y=B, Z=G)
```

heather *Diggle's Heather Data*

Description

The spatial mosaic of vegetation of the heather plant (*Calluna vulgaris*) recorded in a 10 by 20 metre sampling plot in Sweden.

Usage

```
data(heather)
```

Format

A list with three entries, representing the same data at different spatial resolutions:

coarse	original heather data, 100 by 200 pixels
medium	current heather data, 256 by 512 pixels
fine	finest resolution data, 778 by 1570 pixels

Each of these entries is an object of class "owin" containing a binary pixel mask.

Notes on data

These data record the spatial mosaic of vegetation of the heather plant (*Calluna vulgaris*) in a 10 by 20 metre sampling plot near Jdraas, Sweden. They were recorded and first analysed by Diggle(1981).

The dataset heather contains three different versions of the data that have been analysed by different writers over the decades.

coarse: Data as originally digitised by Diggle in 1983 at 100 by 200 pixels resolution (i.e. 10 pixels = 1 metre).

These data were entered by hand in the form of a run-length encoding (original file no longer available) and translated by a program into a 100 by 200 pixel binary image.

There are known to be some errors in the image which arise from errors in counting the run-length so that occasionally there will be an unexpected 'spike' on one single column.

fine: A fine scale digitisation of the original map, prepared by CWI (Centre for Computer Science, Amsterdam, Netherlands) in 1994.

The original hand-drawn map was scanned by Adrian Baddeley, and processed by Chris Jonker, Henk Heijmans and Adrian Baddeley to yield a clean binary image of 778 by 1570 pixels resolution.

medium: The version of the heather data currently supplied on Professor Diggle's website. This is a 256 by 512 pixel image. The method used to create this image is not stated.

History of analysis of data

The data were recorded, presented and analysed by Diggle (1983). He proposed a Boolean model consisting of discs of random size with centres generated by of a Poisson point process.

Renshaw and Ford (1983) reported that spectral analysis of the data suggested the presence of strong row and column effects. However, this may have been attributable to errors in the run-length encoding of the original data.

Hall (1985) and Hall (1988, pp 301-318) took a bootstrap approach.

Ripley (1988, pp. 121-122, 131-135] used opening and closing functions to argue that a Boolean model of discs is inappropriate.

Cressie (1991, pp. 763-770) tried a more general Boolean model.

Source

Peter Diggle

References

- Cressie, N.A.C. (1991) *Statistics for Spatial Data*. John Wiley and Sons, New York.
- Diggle, P.J. (1981) Binary mosaics and the spatial pattern of heather. *Biometrics* **37**, 531-539.
- Hall, P. (1985) Resampling a coverage pattern. *Stochastic Processes and their Applications* **20** 231-246.
- Hall, P. (1988) *An introduction to the theory of coverage processes*. John Wiley and Sons, New York.
- Renshaw, E. and Ford, E.D. (1983) The interpretation of process from pattern using two-dimensional spectral analysis: Methods and problems of interpretation. *Applied Statistics* **32** 51-63.
- Ripley, B.D. (1988) *Statistical Inference for Spatial Processes*. Cambridge University Press.

Hest

Spherical Contact Distribution Function

Description

Estimates the spherical contact distribution function of a random set.

Usage

```
Hest(X, r=NULL, breaks=NULL, ...,
      correction=c("km", "rs", "han"),
      conditional=TRUE)
```

Arguments

- | | |
|-------------|--|
| X | The observed random set. An object of class "ppp", "psp" or "owin". |
| r | Optional. Vector of values for the argument r at which $H(r)$ should be evaluated. Users are advised <i>not</i> to specify this argument; there is a sensible default. |
| breaks | Optional. An alternative to the argument r . Not normally invoked by the user. |
| ... | Arguments passed to <code>as.mask</code> to control the discretisation. |
| correction | Optional. The edge correction(s) to be used to estimate $H(r)$. A vector of character strings selected from "none", "rs", "km", "han" and "best". |
| conditional | Logical value indicating whether to compute the conditional or unconditional distribution. See Details. |

Details

The spherical contact distribution function of a stationary random set X is the cumulative distribution function H of the distance from a fixed point in space to the nearest point of X , given that the point lies outside X . That is, $H(r)$ equals the probability that X lies closer than r units away from the fixed point x , given that X does not cover x .

Let $D = d(x, X)$ be the shortest distance from an arbitrary point x to the set X . Then the spherical contact distribution function is

$$H(r) = P(D \leq r \mid D > 0)$$

For a point process, the spherical contact distribution function is the same as the empty space function F discussed in [Fest](#).

The argument X may be a point pattern (object of class "ppp"), a line segment pattern (object of class "psp") or a window (object of class "owin"). It is assumed to be a realisation of a stationary random set.

The algorithm first calls [distmap](#) to compute the distance transform of X , then computes the Kaplan-Meier and reduced-sample estimates of the cumulative distribution following Hansen et al (1999). If `conditional=TRUE` (the default) the algorithm returns an estimate of the spherical contact function $H(r)$ as defined above. If `conditional=FALSE`, it instead returns an estimate of the cumulative distribution function $H^*(r) = P(D \leq r)$ which includes a jump at $r = 0$ if X has nonzero area.

Accuracy depends on the pixel resolution, which is controlled by the arguments `eps`, `dimyx` and `xy` passed to [as.mask](#). For example, use `eps=0.1` to specify square pixels of side 0.1 units, and `dimyx=256` to specify a 256 by 256 grid of pixels.

Value

An object of class "fv", see [fv.object](#), which can be plotted directly using [plot.fv](#).

Essentially a data frame containing up to six columns:

<code>r</code>	the values of the argument r at which the function $H(r)$ has been estimated
<code>rs</code>	the “reduced sample” or “border correction” estimator of $H(r)$
<code>km</code>	the spatial Kaplan-Meier estimator of $H(r)$
<code>hazard</code>	the hazard rate $\lambda(r)$ of $H(r)$ by the spatial Kaplan-Meier method
<code>han</code>	the spatial Hanisch-Chiu-Stoyan estimator of $H(r)$
<code>raw</code>	the uncorrected estimate of $H(r)$, i.e. the empirical distribution of the distance from a fixed point in the window to the nearest point of X

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A.J. Spatial sampling and censoring. In O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*. Chapman and Hall, 1998. Chapter 2, pages 37-78.

Baddeley, A.J. and Gill, R.D. The empty space hazard of a spatial pattern. Research Report 1994/3, Department of Mathematics, University of Western Australia, May 1994.

Hansen, M.B., Baddeley, A.J. and Gill, R.D. First contact distributions for spatial patterns: regularity and estimation. *Advances in Applied Probability* **31** (1999) 15-33.

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

Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.

See Also

[Fest](#)

Examples

```
X <- runifpoint(42)
H <- Hest(X)
Y <- rpoisline(10)
H <- Hest(Y)
H <- Hest(Y, dimyx=256)
data(heather)
H <- Hest(heather$coarse)
H <- Hest(heather$coarse, conditional=FALSE)
```

hist.im

Histogram of Pixel Values in an Image

Description

Computes and displays a histogram of the pixel values in a pixel image. The `hist` method for class "`im`".

Usage

```
## S3 method for class 'im'
hist(x, ..., probability=FALSE)
```

Arguments

<code>x</code>	A pixel image (object of class " <code>im</code> ").
<code>...</code>	Arguments passed to hist.default or barplot .
<code>probability</code>	Logical. If TRUE, the histogram will be normalised to give probabilities or probability densities.

Details

This function computes and (by default) displays a histogram of the pixel values in the image `x`.

An object of class "`im`" describes a pixel image. See [im.object](#)) for details of this class.

The function `hist.im` is a method for the generic function `hist` for the class "`im`".

Any arguments in `...` are passed to [hist.default](#) (for numeric valued images) or [barplot](#) (for factor or logical images). For example, such arguments control the axes, and may be used to suppress the plotting.

Value

For numeric-valued images, an object of class "histogram" as returned by `hist.default`. This object can be plotted.

For factor-valued or logical images, an object of class "barplotdata", which can be plotted. This is a list with components called `counts` (contingency table of counts of the numbers of pixels taking each possible value), `probs` (corresponding relative frequencies) and `mids` (graphical *x*-coordinates of the midpoints of the bars in the barplot).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`hist`, `hist.default`, `barplot`, `im.object`, `summary.im`.

Examples

```
X <- as.im(function(x,y) {x^2}, unit.square())
hist(X)
hist(cut(X,3))
```

humberside

Humberside Data on Childhood Leukaemia and Lymphoma

Description

Spatial locations of cases of childhood leukaemia and lymphoma, and randomly-selected controls, in North Humberside. A marked point pattern.

Usage

```
data(humberside)
```

Format

The dataset `humberside` is an object of class "ppp" representing a marked point pattern. Entries include

<code>x</code>	Cartesian <i>x</i> -coordinate of home address
<code>y</code>	Cartesian <i>y</i> -coordinate of home address
<code>marks</code>	factor with levels <code>case</code> and <code>control</code> indicating whether this is a disease case or a control.

See `ppp.object` for details of the format.

The dataset `humberside.convex` is an object of the same format, representing the same point pattern data, but contained in a larger, 5-sided convex polygon.

Notes

Cuzick and Edwards (1990) first presented and analysed these data.

The data record 62 cases of childhood leukaemia and lymphoma diagnosed in the North Humberside region of England between 1974 and 1986, together with 141 controls selected at random from the birth register for the same period.

The data are represented as a marked point pattern, with the points giving the spatial location of each individual's home address (actually, the centroid for the postal code) and the marks identifying cases and controls.

Coordinates are expressed in units of 100 metres, and the resolution is 100 metres. At this resolution, there are some duplicated points.

Two versions of the dataset are supplied, both containing the same point coordinates, but using different windows. The dataset `humberside` has a polygonal window with 102 edges which closely approximates the Humberside region, while `humberside.convex` has a convex 5-sided polygonal window originally used by Diggle and Chetwynd (1991) and shown in Figure 1 of that paper. (This pentagon has been modified slightly from the original data, by shifting two vertices horizontally by 1 unit, so that the pentagon contains all the data points.)

Source

Dr Ray Cartwright and Dr Freda Alexander. Published and analysed in Cuzick and Edwards (1990), see Table 1. Pentagonal boundary from Diggle and Chetwynd (1991), Figure 1. Point coordinates and pentagonal boundary supplied by Andrew Lawson. Detailed region boundary was digitised by Adrian Baddeley, 2005, from a reprint of Cuzick and Edwards (1990).

References

- J. Cuzick and R. Edwards (1990) Spatial clustering for inhomogeneous populations. *Journal of the Royal Statistical Society, series B*, **52** (1990) 73-104.
- P.J. Diggle and A.G. Chetwynd (1991) Second-order analysis of spatial clustering for inhomogeneous populations. *Biometrics* 47 (1991) 1155-1163.

Examples

```
data(humberside)
plot(humberside)
plot(humberside.convex>window, add=TRUE, lty=2)
```

hyperframe

Hyper Data Frame

Description

Create a hyperframe: a two-dimensional array in which each column consists of values of the same atomic type (like the columns of a data frame) or objects of the same class.

Usage

```
hyperframe(...,
           row.names=NULL, check.rows=FALSE, check.names=TRUE,
           stringsAsFactors=default.stringsAsFactors())
```

Arguments

- ... Arguments of the form `value` or `tag=value`. Each value is either an atomic vector, or a list of objects of the same class, or a single atomic value, or a single object. Each value will become a column of the array. The tag determines the name of the column. See Details.
- `row.names, check.rows, check.names, stringsAsFactors`
Arguments passed to `data.frame` controlling the names of the rows, whether to check that rows are consistent, whether to check validity of the column names, and whether to convert character columns to factors.

Details

A hyperframe is like a data frame, except that its entries can be objects of any kind.

A hyperframe is a two-dimensional array in which each column consists of values of one atomic type (as in a data frame) or consists of objects of one class.

The arguments ... are any number of arguments of the form `value` or `tag=value`. Each value will become a column of the array. The tag determines the name of the column.

Each value can be either

- an atomic vector or factor (i.e. numeric vector, integer vector, character vector, logical vector, complex vector or factor)
- a list of objects which are all of the same class
- one atomic value, which will be replicated to make an atomic vector or factor
- one object, which will be replicated to make a list of objects.

All columns (vectors, factors and lists) must be of the same length, if their length is greater than 1.

Value

An object of class "hyperframe".

Methods for Hyperframes

There are methods for `print`, `plot`, `summary`, `with`, `[`, `[<,$, $<-`, `names`, `as.data.frame` as.list, `cbind` and `rbind` for the class of hyperframes. There is also `is.hyperframe` and `as.hyperframe`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`as.hyperframe`, `as.hyperframe.ppx`, `plot.hyperframe`, `with.hyperframe`, `as.data.frame.hyperframe`, `cbind.hyperframe`, `rbind.hyperframe`

Examples

```
# equivalent to a data frame
hyperframe(X=1:10, Y=3)

# list of functions
hyperframe(f=list(sin, cos, tan))

# table of functions and matching expressions
hyperframe(f=list(sin, cos, tan),
           e=list(expression(sin(x)), expression(cos(x)), expression(tan(x)))))

hyperframe(X=1:10, Y=letters[1:10], Z=factor(letters[1:10]),
           stringsAsFactors=FALSE)

lambda <- runif(4, min=50, max=100)
X <- lapply(as.list(lambda), function(x) { rpoispp(x) })
h <- hyperframe(lambda=lambda, X=X)
h

h$lambda2 <- lambda^2
h[, "lambda3"] <- lambda^3
h[, "Y"] <- X
```

identify.ppp

Identify Points in a Point Pattern

Description

If a point pattern is plotted in the graphics window, this function will find the point of the pattern which is nearest to the mouse position, and print its mark value (or its serial number if there is no mark).

Usage

```
## S3 method for class 'ppp'
identify(x, ...)
```

Arguments

<code>x</code>	A point pattern (object of class "ppp").
<code>...</code>	Arguments passed to identify.default .

Details

This is a method for the generic function [identify](#) for point pattern objects.

The point pattern `x` should first be plotted using [plot.ppp](#). Then `identify(x)` reads the position of the graphics pointer each time the left mouse button is pressed. It then finds the point of the pattern `x` closest to the mouse position. If this closest point is sufficiently close to the mouse pointer, its index (and its mark if any) will be returned as part of the value of the call.

Each time a point of the pattern is identified, text will be displayed next to the point, showing its serial number (if `x` is unmarked) or its mark value (if `x` is marked).

Value

If x is unmarked, the result is a vector containing the serial numbers of the points in the pattern x that were identified. If x is marked, the result is a 2-column matrix, the first column containing the serial numbers and the second containing the marks for these points.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[identify](#), [clickppp](#)

identify.psp	<i>Identify Segments in a Line Segment Pattern</i>
--------------	--

Description

If a line segment pattern is plotted in the graphics window, this function will find the segment which is nearest to the mouse position, and print its serial number.

Usage

```
## S3 method for class 'psp'
identify(x, ..., labels=seq_len(nsegments(x)), n=nsegments(x), plot=TRUE)
```

Arguments

x	A line segment pattern (object of class "psp").
...	Ignored.
labels	Labels associated with the segments, to be plotted when the segments are identified. A character vector or numeric vector of length equal to the number of segments in x .
n	Maximum number of segments to be identified.
plot	Logical. Whether to plot the labels when a segment is identified.

Details

This is a method for the generic function [identify](#) for line segment pattern objects.

The line segment pattern x should first be plotted using [plot.psp](#). Then `identify(x)` reads the position of the graphics pointer each time the left mouse button is pressed. It then finds the segment in the pattern x that is closest to the mouse position. This segment's index will be returned as part of the value of the call.

Each time a segment is identified, text will be displayed next to the point, showing its serial number (or the relevant entry of `labels`).

Value

Vector containing the serial numbers of the segments in the pattern x that were identified.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[identify](#), [identify.ppp](#).

idw

Inverse-distance weighted smoothing of observations at irregular points

Description

Performs spatial smoothing of numeric values observed at a set of irregular locations using inverse-distance weighting.

Usage

`idw(X, power=2, at="pixels", ...)`

Arguments

- | | |
|--------------------|---|
| <code>X</code> | A marked point pattern (object of class "ppp"). |
| <code>power</code> | Numeric. Power of distance used in the weighting. |
| <code>at</code> | String specifying whether to compute the intensity values at a grid of pixel locations (<code>at="pixels"</code>) or only at the points of <code>X</code> (<code>at="points"</code>). |
| <code>...</code> | Arguments passed to as.mask to control the pixel resolution of the result. |

Details

This function performs spatial smoothing of numeric values observed at a set of irregular locations. Smoothing is performed by inverse distance weighting. If the observed values are v_1, \dots, v_n at locations x_1, \dots, x_n respectively, then the smoothed value at a location u is

$$g(u) = \frac{\sum_i w_i v_i}{\sum_i w_i}$$

where the weights are the inverse p -th powers of distance,

$$w_i = \frac{1}{d(u, x_i)^p}$$

where $d(u, x_i) = ||u - x_i||$ is the Euclidean distance from u to x_i .

The argument `X` must be a marked point pattern (object of class "ppp", see [ppp.object](#)). The points of the pattern are taken to be the observation locations x_i , and the marks of the pattern are taken to be the numeric values v_i observed at these locations.

The marks are allowed to be a data frame. Then the smoothing procedure is applied to each column of marks.

If `at="pixels"` (the default), the smoothed mark value is calculated at a grid of pixels, and the result is a pixel image. The arguments ... control the pixel resolution. See [as.mask](#).

If `at="points"`, the smoothed mark values are calculated at the data points only, using a leave-one-out rule (the mark value at a data point is excluded when calculating the smoothed value for that point).

An alternative to inverse-distance weighting is kernel smoothing, which is performed by [smooth.ppp](#).

Value

If X has a single column of marks:

- If `at="pixels"` (the default), the result is a pixel image (object of class "im"). Pixel values are values of the interpolated function.
- If `at="points"`, the result is a numeric vector of length equal to the number of points in X. Entries are values of the interpolated function at the points of X.

If X has a data frame of marks:

- If `at="pixels"` (the default), the result is a named list of pixel images (object of class "im"). There is one image for each column of marks. This list also belongs to the class `listof`, for which there is a plot method.
- If `at="points"`, the result is a data frame with one row for each point of X, and one column for each column of marks. Entries are values of the interpolated function at the points of X.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[density.ppp](#), [ppp.object](#), [im.object](#).

See [smooth.ppp](#) for kernel smoothing.

To perform interpolation, see also the `akima` package.

Examples

```
# data frame of marks: trees marked by diameter and height
data(finpine)
plot(idw(finpine))
idw(finpine, at="points")[1:5,]
```

Iest

Estimate the I-function

Description

Estimates the summary function $I(r)$ for a multitype point pattern.

Usage

```
Iest(X, ..., eps=NULL, r=NULL, breaks=NULL, correction=NULL)
```

Arguments

X	The observed point pattern, from which an estimate of $I(r)$ will be computed. An object of class "ppp", or data in any format acceptable to as.ppp() .
...	Ignored.
eps	the resolution of the discrete approximation to Euclidean distance (see below). There is a sensible default.
r	Optional. Numeric vector of values for the argument r at which $I(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
breaks	An alternative to the argument r. Not normally invoked by the user. See Details section.
correction	Optional. Vector of character strings specifying the edge correction(s) to be used by Jest .

Details

The I function summarises the dependence between types in a multitype point process (Van Lieshout and Baddeley, 1999) It is based on the concept of the J function for an unmarked point process (Van Lieshout and Baddeley, 1996). See [Jest](#) for information about the J function.

The I function is defined as

$$I(r) = \sum_{i=1}^m p_i J_{ii}(r) - J_{\bullet\bullet}(r)$$

where $J_{\bullet\bullet}$ is the J function for the entire point process ignoring the marks, while J_{ii} is the J function for the process consisting of points of type i only, and p_i is the proportion of points which are of type i .

The I function is designed to measure dependence between points of different types, even if the points are not Poisson. Let X be a stationary multitype point process, and write X_i for the process of points of type i . If the processes X_i are independent of each other, then the I -function is identically equal to 0. Deviations $I(r) < 1$ or $I(r) > 1$ typically indicate negative and positive association, respectively, between types. See Van Lieshout and Baddeley (1999) for further information.

An estimate of I derived from a multitype spatial point pattern dataset can be used in exploratory data analysis and formal inference about the pattern. The estimate of $I(r)$ is compared against the constant function 0. Deviations $I(r) < 1$ or $I(r) > 1$ may suggest negative and positive association, respectively.

This algorithm estimates the I -function from the multitype point pattern X. It assumes that X can be treated as a realisation of a stationary (spatially homogeneous) random spatial marked point process in the plane, observed through a bounded window.

The argument X is interpreted as a point pattern object (of class "ppp", see [ppp.object](#)) and can be supplied in any of the formats recognised by [as.ppp\(\)](#). It must be a multitype point pattern (it must have a marks vector which is a factor).

The function [Jest](#) is called to compute estimates of the J functions in the formula above. In fact three different estimates are computed using different edge corrections. See [Jest](#) for information.

Value

An object of class "fv", see [fv.object](#), which can be plotted directly using [plot.fv](#).

Essentially a data frame containing

r	the vector of values of the argument r at which the function I has been estimated
rs	the “reduced sample” or “border correction” estimator of $I(r)$ computed from the border-corrected estimates of J functions
km	the spatial Kaplan-Meier estimator of $I(r)$ computed from the Kaplan-Meier estimates of J functions
han	the Hanisch-style estimator of $I(r)$ computed from the Hanisch-style estimates of J functions
un	the uncorrected estimate of $I(r)$ computed from the uncorrected estimates of J
theo	the theoretical value of $I(r)$ for a stationary Poisson process: identically equal to 0

Note

Sizeable amounts of memory may be needed during the calculation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Van Lieshout, M.N.M. and Baddeley, A.J. (1996) A nonparametric measure of spatial interaction in point patterns. *Statistica Neerlandica* **50**, 344–361.
- Van Lieshout, M.N.M. and Baddeley, A.J. (1999) Indices of dependence between types in multivariate point patterns. *Scandinavian Journal of Statistics* **26**, 511–532.

See Also

[Jest](#)

Examples

```
data(amacrine)
Ic <- Iest(amacrine)
plot(Ic, main="Amacrine Cells data")
# values are below I= 0, suggesting negative association
# between 'on' and 'off' cells.
```

Description

Creates an object of class "im" representing a two-dimensional pixel image.

Usage

```
im(mat, xcol=seq_len(ncol(mat)), yrow=seq_len(nrow(mat)),
  xrange=NULL, yrange=NULL,
  unitname=NULL)
```

Arguments

<code>mat</code>	matrix or vector containing the pixel values of the image.
<code>xcol</code>	vector of <code>x</code> coordinates for the pixel grid
<code>yrow</code>	vector of <code>y</code> coordinates for the pixel grid
<code>xrange, yrange</code>	Optional. Vectors of length 2 giving the <code>x</code> and <code>y</code> limits of the enclosing rectangle. (Ignored if <code>xcol, yrow</code> are present.)
<code>unitname</code>	Optional. Name of unit of length. Either a single character string, or a vector of two character strings giving the singular and plural forms, respectively.

Details

This function creates an object of class "`im`" representing a two-dimensional pixel image. See [im.object](#) for details of this class.

The matrix `mat` contains the ‘greyscale’ values for a rectangular grid of pixels. Note carefully that the entry `mat[i, j]` gives the pixel value at the location `(xcol[j], yrow[i])`. That is, the `row` index of the matrix `mat` corresponds to increasing `y` coordinate, while the column index of `mat` corresponds to increasing `x` coordinate. Thus `yrow` has one entry for each row of `mat` and `xcol` has one entry for each column of `mat`. Under the usual convention in R, a correct display of the image would be obtained by transposing the matrix, e.g. `image.default(xcol, yrow, t(mat))`, if you wanted to do it by hand.

The entries of `mat` may be numeric (real or integer), complex, logical, character, or factor values. If `mat` is not a matrix, it will be converted into a matrix with `nrow(mat) = length(yrow)` and `ncol(mat) = length(xcol)`.

To make a factor-valued image, note that R has a quirky way of handling matrices with factor-valued entries. The command `matrix` cannot be used directly, because it destroys factor information. To make a factor-valued image, do one of the following:

- Create a factor containing the pixel values, say `mat <- factor(.....)`, and then assign matrix dimensions to it by `dim(mat) <- c(nr, nc)` where `nr, nc` are the numbers of rows and columns. The resulting object `mat` is both a factor and a vector.
- Supply `mat` as a one-dimensional factor and specify the arguments `xcol` and `yrow` to determine the dimensions of the image.
- Use the functions `cut.im` or `eval.im` to make factor-valued images from other images).

For a description of the methods available for pixel image objects, see [im.object](#).

To convert other kinds of data to a pixel image (for example, functions or windows), use [as.im](#).

Warnings

The internal representation of images is likely to change in future releases of `spatstat`. The safe way to extract pixel values from an image object is to use [as.matrix.im](#) or [\[.im](#).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[im.object](#), [as.im](#), [as.matrix.im](#), [\[.im](#), [eval.im](#)

Examples

```

vec <- rnorm(1200)
mat <- matrix(vec, nrow=30, ncol=40)
whitenoise <- im(mat)
whitenoise <- im(mat, xrange=c(0,1), yrange=c(0,1))
whitenoise <- im(mat, xcol=seq(0,1,length=40), yrow=seq(0,1,length=30))
whitenoise <- im(vec, xcol=seq(0,1,length=40), yrow=seq(0,1,length=30))
plot(whitenoise)

# Factor-valued images:
f <- factor(letters[1:12])
dim(f) <- c(3,4)
Z <- im(f)

# Factor image from other image:
cutwhite <- cut(whitenoise, 3)
plot(cutwhite)

# Factor image from raw data
cutmat <- cut(mat, 3)
dim(cutmat) <- c(30,40)
cutwhite <- im(cutmat)
plot(cutwhite)

```

im.object

Class of Images

Description

A class "im" to represent a two-dimensional pixel image.

Details

An object of this class represents a two-dimensional pixel image. It specifies

- the dimensions of the rectangular array of pixels
- x and y coordinates for the pixels
- a numeric value ("grey value") at each pixel

If X is an object of type im, it contains the following elements:

v	matrix of values
dim	dimensions of matrix v
xrange	range of x coordinates of image window
yrange	range of y coordinates of image window
xstep	width of one pixel
ystep	height of one pixel
xcol	vector of x coordinates of centres of pixels
yrow	vector of y coordinates of centres of pixels

Users are strongly advised not to manipulate these entries directly.

Objects of class "im" may be created by the functions `im` and `as.im`. Image objects are also returned by various functions including `distmap`, `Kmeasure`, `setcov`, `eval.im` and `cut.im`.

Image objects may be displayed using the methods `plot.im`, `image.im`, `persp.im` and `contour.im`. There are also methods `print.im` for printing information about an image, `summary.im` for summarising an image, `mean.im` for calculating the average pixel value, `hist.im` for plotting a histogram of pixel values, `quantile.im` for calculating quantiles of pixel values, and `cut.im` for dividing the range of pixel values into categories.

Pixel values in an image may be extracted using the subset operator `[.im]`. To extract all pixel values from an image object, use `as.matrix.im`. The levels of a factor-valued image can be extracted and changed with `levels` and `levels<-`.

Calculations involving one or more images (for example, squaring all the pixel values in an image, converting numbers to factor levels, or subtracting one image from another) can often be done easily using `eval.im`. To find all pixels satisfying a certain constraint, use `solutionset`.

Note carefully that the entry `v[i, j]` gives the pixel value at the location `(xcol[j], yrow[i])`. That is, the `row` index of the matrix `v` corresponds to increasing `y` coordinate, while the column index of `mat` corresponds to increasing `x` coordinate. Thus `yrow` has one entry for each row of `v` and `xcol` has one entry for each column of `v`. Under the usual convention in R, a correct display of the image would be obtained by transposing the matrix, e.g. `image.default(xcol, yrow, t(v))`, if you wanted to do it by hand.

Warnings

The internal representation of images is likely to change in future releases of `spatstat`. Do not address the entries in an image directly. To extract all pixel values from an image object, use `as.matrix.im`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`im`, `as.im`, `plot.im`, `persp.im`, `eval.im`, `[.im`

imcov

Spatial Covariance of a Pixel Image

Description

Computes the unnormalised spatial covariance function of a pixel image.

Usage

`imcov(X, Y=X)`

Arguments

- | | |
|----------------|---------------------------------------|
| <code>X</code> | A pixel image (object of class "im"). |
| <code>Y</code> | Optional. Another pixel image. |

Details

The (uncentred, unnormalised) *spatial covariance function* of a pixel image X in the plane is the function $C(v)$ defined for each vector v as

$$C(v) = \int X(u)X(u - v) du$$

where the integral is over all spatial locations u , and where $X(u)$ denotes the pixel value at location u .

This command computes a discretised approximation to the spatial covariance function, using the Fast Fourier Transform. The return value is another pixel image (object of class "im") whose greyscale values are values of the spatial covariance function.

If the argument Y is present, then `imcov(X, Y)` computes the set *cross-covariance* function $C(u)$ defined as

$$C(v) = \int X(u)Y(u - v) du.$$

Note that `imcov(X, Y)` is equivalent to `convolve.im(X, Y, reflectY=TRUE)`.

Value

A pixel image (an object of class "im") representing the spatial covariance function of X , or the cross-covariance of X and Y .

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`setcov`, `convolve.im`, `owin`, `as.owin`, `erosion`

Examples

```
X <- as.im(square(1))
v <- imcov(X)
plot(v)
```

`incircle`

Find Largest Circle Inside Window

Description

Find the largest circle contained in a given window.

Usage

```
incircle(W)
```

Arguments

W	A window (object of class "owin").
---	------------------------------------

Details

Given a window W of any type and shape, this function determines the largest circle that is contained inside W .

For non-rectangular windows, the incircle is computed approximately by finding the maximum of the distance map (see [distmap](#)) of the complement of the window.

Value

A list with entries x, y, r giving the location (x, y) and radius r of the incircle.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[centroid.owin](#)

Examples

```
W <- square(1)
Wc <- incircle(W)
plot(W)
plot(disc(Wc$r, c(Wc$x, Wc$y)), add=TRUE)

data(letterR)
plot(letterR)
Rc <- incircle(letterR)
plot(disc(Rc$r, c(Rc$x, Rc$y)), add=TRUE)

W <- as.mask(letterR)
plot(W)
Rc <- incircle(W)
plot(disc(Rc$r, c(Rc$x, Rc$y)), add=TRUE)
```

Description

Define the coordinates of one or more straight lines in the plane

Usage

```
inflne(a = NULL, b = NULL, h = NULL, v = NULL, p = NULL, theta = NULL)
## S3 method for class 'inflne'
print(x, ...)
## S3 method for class 'inflne'
plot(x, ...)
```

Arguments

a,b	Numeric vectors of equal length giving the intercepts a and slopes b of the lines. Incompatible with h,v,p,theta
h	Numeric vector giving the positions of horizontal lines when they cross the y axis. Incompatible with a,b,v,p,theta
v	Numeric vector giving the positions of vertical lines when they cross the x axis. Incompatible with a,b,h,p,theta
p, theta	Numeric vectors of equal length giving the polar coordinates of the line. Incompatible with a,b,h,v
x	An object of class "inflne"
...	Extra arguments passed to <code>print</code> for printing or <code>abline</code> for plotting

Details

The class `inflne` is a convenient way to handle infinite straight lines in the plane.

The position of a line can be specified in several ways:

- its intercept a and slope b in the equation $y = a + bx$ can be used unless the line is vertical.
- for vertical lines we can use the position v where the line crosses the y axis
- for horizontal lines we can use the position h where the line crosses the x axis
- the polar coordinates p and θ can be used for any line. The line equation is

$$y \cos \theta + x \sin \theta = p$$

The command `inflne` will accept line coordinates in any of these formats. The arguments a,b,h,v have the same interpretation as they do in the line-plotting function `abline`.

The command `inflne` converts between different coordinate systems (e.g. from a,b to p,theta) and returns an object of class "inflne" that contains a representation of the lines in each appropriate coordinate system. This object can be printed and plotted.

Value

The value of `inflne` is an object of class "inflne" which is basically a data frame with columns a,b,h,v,p,theta. Each row of the data frame represents one line. Entries may be NA if a coordinate is not applicable to a particular line.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

Examples

```
inflne(a=10:13,b=1)
inflne(p=1:3, theta=pi/4)
plot(c(-1,1),c(-1,1),type="n",xlab="",ylab="", asp=1)
plot(inflne(p=0.4, theta=seq(0,pi,length=20)))
```

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=list())
```

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 (2011).

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 plotted (by `plot.influence.ppm`), or converted to a marked point pattern by `as.ppp` (see `as.ppp.influence.ppm`).

Value

An object of class "influence.ppm" that can be plotted by `plot.influence.ppm`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

References

Baddeley, A. and Chang, Y.M. and Song, Y. (2011) Leverage and influence diagnostics for spatial point process models. *Scandinavian Journal of Statistics*, in press.

See Also

[leverage.ppm](#), [dfbetas.ppm](#), [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](#).

Anyway, `inforder.family` is an object of class "isf" containing a function `inforder.family$eval` for evaluating the sufficient statistics of a Gibbs point process model taking an exponential family form.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

References

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

See Also

[AreaInter](#) to create the area interaction process structure.

Other families: [pairwise.family](#), [pairsat.family](#), [ord.family](#).

inside.owin*Test Whether Points Are Inside A Window***Description**

Test whether points lie inside or outside a given window.

Usage

```
inside.owin(x, y, w)
```

Arguments

- x Vector of x coordinates of points to be tested. (Alternatively, a point pattern object providing both x and y coordinates.)
- y Vector of y coordinates of points to be tested.
- w A window. This should be an object of class [owin](#), or can be given in any format acceptable to [as.owin\(\)](#).

Details

This function tests whether each of the points $(x[i], y[i])$ lies inside or outside the window w and returns TRUE if it is inside.

The boundary of the window is treated as being inside.

If w is of type "rectangle" or "polygonal", the algorithm uses analytic geometry (the discrete Stokes theorem). Computation time is linear in the number of points and (for polygonal windows) in the number of vertices of the boundary polygon. Boundary cases are correct to single precision accuracy.

If w is of type "mask" then the pixel closest to $(x[i], y[i])$ is tested. The results may be incorrect for points lying within one pixel diameter of the window boundary.

Normally x and y must be numeric vectors of equal length (length zero is allowed) containing the coordinates of points. Alternatively x can be a point pattern (object of class "ppp") while y is missing; then the coordinates of the point pattern are extracted.

Value

Logical vector whose i th entry is TRUE if the corresponding point $(x[i], y[i])$ is inside w .

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[owin.object](#), [as.owin](#)

Examples

```

# hexagonal window
k <- 6
theta <- 2 * pi * (0:(k-1))/k
co <- cos(theta)
si <- sin(theta)
mas <- owin(c(-1,1), c(-1,1), poly=list(x=co, y=si))
## Not run:
plot(mas)

## End(Not run)

# random points in rectangle
x <- runif(30,min=-1, max=1)
y <- runif(30,min=-1, max=1)

ok <- inside.owin(x, y, mas)

## Not run:
points(x[ok], y[ok])
points(x[!ok], y[!ok], pch="x")

## End(Not run)

```

integral.im

Integral of a Pixel Image

Description

Computes the integral of a pixel image.

Usage

```
integral.im(x, ...)
```

Arguments

- | | |
|-----|--|
| x | A pixel image (object of class "im") with pixel values that can be treated as numeric or complex values. |
| ... | Ignored. |

Details

This function treats the pixel image x as a function of the spatial coordinates, and computes its integral. The integral is calculated by summing the pixel values and multiplying by the area of one pixel.

The pixel values of x may be numeric, integer, logical or complex. They cannot be factor or character values.

The logical values TRUE and FALSE are converted to 1 and 0 respectively, so that the integral of a logical image is the total area of the TRUE pixels, in the same units as `unitname(x)`.

For more complicated integration tasks such as computing the integral of an image over a specified subset, use `eval.im` to construct an integrand or `[.im` to extract a subset of the image.

Value

A single numeric or complex value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[eval.im](#), [\[.im](#)

Examples

```
# approximate integral of f(x,y) dx dy
f <- function(x,y){3*x^2 + 2*y}
Z <- as.im(f, square(1))
integral.im(Z)
# correct answer is 2

data(cells)
D <- density(cells)
integral.im(D)
# should be approximately equal to number of points = 42

# integrate over the subset [0.1,0.9] x [0.2,0.8]
W <- owin(c(0.1,0.9), c(0.2,0.8))
DW <- D[W, drop=FALSE]
integral.im(DW)
```

intensity

Intensity of a Dataset or a Model

Description

Generic function for computing the intensity of a spatial dataset or spatial point process model.

Usage

`intensity(X, ...)`

Arguments

- | | |
|----------------|--|
| <code>X</code> | A spatial dataset or a spatial point process model. |
| ... | Further arguments depending on the class of <code>X</code> . |

Details

This is a generic function for computing the intensity of a spatial dataset or spatial point process model. There are methods for point patterns (objects of class "ppp") and fitted point process models (objects of class "ppm").

The empirical intensity of a dataset is the average density (the average amount of 'stuff' per unit area or volume). The empirical intensity of a point pattern is computed by the method [intensity.ppp](#).

The theoretical intensity of a stochastic model is the expected density (expected amount of 'stuff' per unit area or volume). The theoretical intensity of a fitted point process model is computed by the method [intensity.ppm](#).

Value

Usually a numeric value or vector.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[intensity.ppp](#), [intensity.ppm](#).

[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, ...)
```

Arguments

X	A fitted point process model (object of class "ppm").
...	Arguments passed to predict.ppm in some cases. See Details.

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 another Gibbs point process model, the intensity is computed approximately using the Poisson-saddlepoint approximation (Baddeley and Nair, 2012).

Value

A numeric value (if the model is stationary) or a pixel image.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>, Gopal Nair, and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A. and Nair, G. (2012) Fast approximation of the intensity of Gibbs point processes. *Electronic Journal of Statistics* **6** 1155–1169.

See Also

[intensity](#), [intensity.ppp](#)

Examples

```
fitP <- ppm(swedishpines, ~1, Poisson())
intensity(fitP)
fitS <- ppm(swedishpines, ~1, Strauss(9))
intensity(fitS)
```

[intensity.ppp](#)

Empirical Intensity of Point Pattern

Description

Computes the average number of points per unit area in a point pattern dataset.

Usage

```
## S3 method for class 'ppp'
intensity(X, ...)
```

Arguments

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

Details

This is a method for the generic function [intensity](#). It computes the empirical intensity of a point pattern (object of class "ppp"), i.e. the average density of points per unit area.

If the point pattern is multitype, the intensities of the different types are computed separately.

Value

A numeric value (giving the intensity) or numeric vector (giving the intensity for each possible type).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[intensity](#), [intensity.ppm](#)

Examples

```
intensity(cells)
```

interp.im

Interpolate a Pixel Image

Description

Interpolates the values of a pixel image at any desired location in the frame.

Usage

```
interp.im(Z, x, y)
```

Arguments

Z Pixel image (object of class "im") with numeric or integer values.
x,y Vectors of Cartesian coordinates.

Details

A value at each location ($x[i], y[i]$) will be interpolated using the pixel values of Z at the four surrounding pixel centres, by simple bilinear interpolation.

At the boundary (where ($x[i], y[i]$) is not surrounded by four pixel centres) the value at the nearest pixel is taken.

Value

Vector of interpolated values, with NA for points that lie outside the domain of the image.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

Examples

```

opa <- par(mfrow=c(1,2))
# coarse image
V <- as.im(function(x,y) { x^2 + y }, owin(), dimyx=10)
plot(V, main="coarse image", col=terrain.colors(256))

# lookup value at location (0.5,0.5)
V[list(x=0.5,y=0.5)]
# interpolated value at location (0.5,0.5)
interp.im(V, 0.5, 0.5)
# true value is 0.75

# how to obtain an interpolated image at a desired resolution
U <- as.im(interp.im, W=owin(), Z=V, dimyx=256)
plot(U, main="interpolated image", col=terrain.colors(256))
par(opa)

```

intersect.owin

Intersection, Union or Set Subtraction of Two Windows

Description

Yields the intersection, union or set subtraction of two windows.

Usage

```

intersect.owin(A, B, ..., fatal=TRUE)
union.owin(A,B, ...)
setminus.owin(A,B, ...)

```

Arguments

A	A window object (see Details).
B	A window object.
...	Optional arguments passed to as.mask to control the discretisation, if required.
fatal	Logical. Determines what happens if the intersection is empty.

Details

The function `intersect.owin` computes the intersection between the two windows A and B, while `union.owin` computes their union. The function `setminus.owin` computes the intersection of A with the complement of B.

The arguments A and B must be window objects (either objects of class "owin", or data that can be coerced to this class by [as.owin](#)).

If the intersection is empty, then if `fatal=FALSE` the result is NULL, while if `fatal=TRUE` an error occurs.

The intersection or union of more than two windows can also be computed. For `intersect.owin` and `union.owin` the arguments ... can include additional window objects.

Value

A window (object of class "owin").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[is.subset.owin](#), [overlap.owin](#), [bounding.box](#), [owin.object](#)

Examples

```
# rectangles
u <- unit.square()
v <- owin(c(0.5,3.5), c(0.4,2.5))
# polygon
data(letterR)
# mask
m <- as.mask(letterR)

# two rectangles
intersect.owin(u, v)
union.owin(u,v)
setminus.owin(u,v)

# polygon and rectangle
intersect.owin(letterR, v)
union.owin(letterR,v)
setminus.owin(letterR,v)

# mask and rectangle
intersect.owin(m, v)
union.owin(m,v)
setminus.owin(m,v)

# mask and polygon
p <- rotate(v, 0.2)
intersect.owin(m, p)
union.owin(m,p)
setminus.owin(m,p)

# two polygons
A <- letterR
B <- rotate(letterR, 0.2)
plot(bounding.box(A,B), main="intersection")
w <- intersect.owin(A, B)
plot(w, add=TRUE, col="lightblue")
plot(A, add=TRUE)
plot(B, add=TRUE)

plot(bounding.box(A,B), main="union")
w <- union.owin(A,B)
plot(w, add=TRUE, col="lightblue")
plot(A, add=TRUE)
```

```

plot(B, add=TRUE)

plot(bounding.box(A,B), main="set minus")
w <- setminus.owin(A,B)
plot(w, add=TRUE, col="lightblue")
plot(A, add=TRUE)
plot(B, add=TRUE)

# intersection and union of three windows
C <- shift(B, c(0.2, 0.3))
plot(union.owin(A,B,C))
plot(intersect.owin(A,B,C))

```

intersect.tess*Intersection of Two Tessellations***Description**

Yields the intersection of two tessellations, or the intersection of a tessellation with a window.

Usage

```
intersect.tess(X, Y, ...)
```

Arguments

- | | |
|------|--|
| X, Y | Two tessellations (objects of class "tess"), or windows (objects of class "tess"), or other data that can be converted to tessellations by as.tess . |
| ... | Optional arguments passed to as.mask to control the discretisation, if required. |

Details

A tessellation is a collection of disjoint spatial regions (called *tiles*) that fit together to form a larger spatial region. See [tess](#).

If X and Y are not tessellations, they are first converted into tessellations by [as.tess](#).

The function `intersect.tess` then computes the intersection between the two tessellations. This is another tessellation, each of whose tiles is the intersection of a tile from X and a tile from Y.

One possible use of this function is to slice a window W into subwindows determined by a tessellation. See the Examples.

Value

A tessellation (object of class "tess").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[tess](#), [as.tess](#), [intersect.owin](#)

Examples

```
opa <- par(mfrow=c(1, 3))
# polygon
data(letterR)
plot(letterR)
# tessellation of rectangles
X <- tess(xgrid=seq(2, 4, length=10), ygrid=seq(0, 3.5, length=8))
plot(X)
plot(intersect.tess(X, letterR))

A <- runifpoint(10)
B <- runifpoint(10)
plot(DA <- dirichlet(A))
plot(DB <- dirichlet(B))
plot(intersect.tess(DA, DB))

par(opa)
```

iplot*Point and Click Interface for Displaying a Point Pattern*

Description

Plot a two-dimensional spatial point pattern with interactive (point-and-click) control over the plot.

Usage

```
iplot(x, xname)
```

Arguments

- | | |
|--------------------|--|
| <code>x</code> | The spatial point pattern to be plotted. An object of class "ppp". |
| <code>xname</code> | Optional. Character string to use as the title of the dataset. |

Details

This function generates a plot of a spatial point pattern dataset (object of class "ppp") and allows interactive control over the appearance of the plot using a point-and-click interface.

A new popup window is launched. The point pattern `x` is displayed in the left half of the window using `plot.ppp` with the default values of all the plot parameters. The right side of the window contains buttons and sliders allowing the user to change the header text, the plot symbols, the scale used to represent numeric marks, and so on. For a multitype point pattern, the user can also switch between viewing all points in a single display, and splitting the points into separate patterns according to type.

Value

NULL.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[istat](#), [ppp.object](#), [plot.ppp](#)

Examples

```
if(interactive()) {
  data(cells)
  iplot(cells)
}
```

ippm

*Optimise Irregular Trend Parameters in Point Process Model***Description**

Experimental extension to [ppm](#). Find optimal values of the irregular trend parameters in a point process model using Fisher scoring algorithm.

Usage

```
ippm(..., iScore=NULL,
      start=list(),
      covfunargs=start,
      maxiter=20, tol=1e-4, progress=TRUE, stepfactor=1,
      dbug=FALSE)
```

Arguments

...	Arguments passed to ppm to fit the point process model.
iScore	A named list of R functions that compute the partial derivatives of <code>logf</code> 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 <i>all</i> irregular parameters required by the covariates in the model. Must include all the parameters named in <code>start</code> .
maxiter	Integer. Maximum number of iterations of Fisher scoring algorithm.
tol	Numeric value or vector. The algorithm stops when the difference between two successive estimates of the irregular parameter is less than <code>tol</code> .
progress	Logical. Whether to print progress reports.
stepfactor	Numeric value between 0 and 1 indicating that the change in the parameter between successive iterations is only a specified fraction of the step computed by the Newton-Raphson algorithm.
dbug	Logical. Whether to print debugging output.

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 Fisher scoring method.

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.

To perform Fisher scoring we also need the derivative of $\log f(u, \gamma)$ with respect to γ . We call this the *irregular score*. 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` must be a list (with one entry for each entry of `start`) of functions with arguments x, y, \dots , 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 \dots 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").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

`ppm`

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),
      tol=0.01, nd=nd)

```

is.convex

Test Whether a Window is Convex

Description

Determines whether a window is convex.

Usage

```
is.convex(x)
```

Arguments

x	Window (object of class "owin").
---	----------------------------------

Details

If x is a rectangle, the result is TRUE.

If x is polygonal, the result is TRUE if x consists of a single polygon and this polygon is equal to the minimal convex hull of its vertices computed by [chull](#).

If x is a mask, the algorithm first extracts all boundary pixels of x using [vertices](#). Then it computes the (polygonal) convex hull K of the boundary pixels. The result is TRUE if every boundary pixel lies within one pixel diameter of an edge of K .

Value

Logical value, equal to TRUE if x is convex.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[owin](#), [convexhull.xy](#), [vertices](#)

is.empty

Test Whether An Object Is Empty

Description

Checks whether the argument is an empty window, an empty point pattern, etc.

Usage

```
is.empty(x)
## S3 method for class 'owin'
is.empty(x)
## S3 method for class 'ppp'
is.empty(x)
## S3 method for class 'psp'
is.empty(x)
## Default S3 method:
is.empty(x)
```

Arguments

x A window (object of class "owin"), a point pattern (object of class "ppp"), or a line segment pattern (object of class "psp").

Details

This function tests whether the object x represents an empty spatial object, such as an empty window, a point pattern with zero points, or a line segment pattern with zero line segments.

An empty window can be obtained as the output of [intersect.owin](#), [erosion](#), [opening](#), [complement.owin](#) and some other operations.

An empty point pattern or line segment pattern can be obtained as the result of simulation.

Value

Logical value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

is.im*Test Whether An Object Is A Pixel Image***Description**

Tests whether its argument is a pixel image (object of class "im").

Usage

```
is.im(x)
```

Arguments

x	Any object.
----------	-------------

Details

This function tests whether the argument **x** is a pixel image object of class "im". For details of this class, see [im.object](#).

The object is determined to be an image if it inherits from class "im".

Value

TRUE if **x** is a pixel image, otherwise FALSE.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

is.marked*Test Whether Marks Are Present***Description**

Generic function to test whether a given object (usually a point pattern or something related to a point pattern) has "marks" attached to the points.

Usage

```
is.marked(x, ...)
```

Arguments

x	Object to be inspected
...	Other arguments.

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.

Other objects related to point patterns, such as point process models, may involve marked points.

This function tests whether the object X contains or involves marked points. It is generic; methods are provided for point patterns (objects of class "ppp") and point process models (objects of class "ppm").

Value

Logical value, equal to TRUE if X is marked.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[is.marked.ppp](#), [is.marked.ppm](#)

[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). 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 marked point pattern dataset.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[is.marked](#), [is.marked.ppp](#)

Examples

```
data(lansing)
# Multitype point pattern --- trees marked by species

fit1 <- ppm(lansing, ~ marks, Poisson())
is.marked(fit1)
# TRUE

fit2 <- ppm(lansing, ~ 1, Poisson())
is.marked(fit2)
# TRUE

data(cells)
# Unmarked point pattern

fit3 <- ppm(cells, ~ 1, Poisson())
is.marked(fit3)
# FALSE
```

Description

Tests whether a point pattern has “marks” attached to the points.

Usage

```
## S3 method for class 'ppp'
is.marked(X, na.action="warn", ...)
```

Arguments

X	Point pattern (object of class "ppp")
na.action	String indicating what to do if NA values are encountered amongst the marks. Options are "warn", "fatal" and "ignore".
...	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.

This function tests whether the point pattern X contains or involves marked points. It is a method for the generic function `is.marked`.

The argument `na.action` determines what action will be taken if the point pattern has a vector of marks but some or all of the marks are NA. Options are "fatal" to cause a fatal error; "warn" to issue a warning and then return TRUE; and "ignore" to take no action except returning TRUE.

Value

Logical value, equal to TRUE if X is a marked point pattern.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`is.marked`, `is.marked.ppm`

Examples

```
data(cells)
is.marked(cells) #FALSE
data(longleaf)
is.marked(longleaf) #TRUE
```

`is.multitype`

Test whether Object is Multitype

Description

Generic function to test whether a given object (usually a point pattern or something related to a point pattern) has “marks” attached to the points which classify the points into several types.

Usage

`is.multitype(X, ...)`

Arguments

- X Object to be inspected
 ... Other arguments.

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. Other objects related to point patterns, such as point process models, may involve marked points.

This function tests whether the object X contains or involves marked points, **and** that the marks are a factor.

For example, the `amacrine` dataset is multitype (there are two types of cells, on and off), but the `longleaf` dataset is *not* multitype (the marks are real numbers).

This function is generic; methods are provided for point patterns (objects of class "ppp") and point process models (objects of class "ppm").

Value

Logical value, equal to TRUE if X is multitype.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[is.multitype.ppp](#), [is.multitype.ppm](#)

[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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[is.multitype](#), [is.multitype.ppp](#)

Examples

```
data(lansing)
# Multitype point pattern --- trees marked by species

fit1 <- ppm(lansing, ~ marks, Poisson())
is.multitype(fit1)
# TRUE

fit2 <- ppm(lansing, ~ 1, Poisson())
is.multitype(fit2)
# TRUE

data(cells)
# Unmarked point pattern

fit3 <- ppm(cells, ~ 1, Poisson())
is.multitype(fit3)
# FALSE
```

is.multitype.ppp*Test Whether A Point Pattern is Multitype***Description**

Tests whether a point pattern has “marks” attached to the points which classify the points into several types.

Usage

```
## S3 method for class 'ppp'
is.multitype(X, na.action="warn", ...)
```

Arguments

X	Point pattern (object of class "ppp")
na.action	String indicating what to do if NA values are encountered amongst the marks. Options are "warn", "fatal" and "ignore".
...	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.

This function tests whether the point pattern X contains or involves marked points, **and** that the marks are a factor. It is a method for the generic function [is.multitype](#).

For example, the [amacrine](#) dataset is multitype (there are two types of cells, on and off), but the [longleaf](#) dataset is *not* multitype (the marks are real numbers).

The argument na.action determines what action will be taken if the point pattern has a vector of marks but some or all of the marks are NA. Options are "fatal" to cause a fatal error; "warn" to issue a warning and then return TRUE; and "ignore" to take no action except returning TRUE.

Value

Logical value, equal to TRUE if X is a multitype point pattern.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[is.multitype](#), [is.multitype.ppm](#)

Examples

```
data(cells)
is.multitype(cells) #FALSE - no marks
data(longleaf)
is.multitype(longleaf) #FALSE - real valued marks
data(amacrine)
is.multitype(amacrine) #TRUE
```

is.owin

Test Whether An Object Is A Window

Description

Checks whether its argument is a window (object of class "owin").

Usage

```
is.owin(x)
```

Arguments

x Any object.

Details

This function tests whether the object x is a window object of class "owin". See [owin.object](#) for details of this class.

The result is determined to be TRUE if x inherits from "owin", i.e. if x has "owin" amongst its classes.

Value

TRUE if x is a point pattern, otherwise FALSE.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

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").

Usage

```
is.ppm(x)
```

Arguments

x	Any object.
---	-------------

Details

This function tests whether the object x is a fitted point process model object of class "ppm". See [ppm.object](#) for details of this class.

Value

TRUE if x has "ppm" amongst its classes, otherwise FALSE.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

is.ppp*Test Whether An Object Is A Point Pattern***Description**

Checks whether its argument is a point pattern (object of class "ppp").

Usage

```
is.ppp(x)
```

Arguments

x	Any object.
---	-------------

Details

This function tests whether the object x is a point pattern object of class "ppp". See [ppm.object](#) for details of this class.

The result is determined to be TRUE if x inherits from "ppp", i.e. if x has "ppp" amongst its classes.

Value

TRUE if x is a point pattern, otherwise FALSE.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

is.rectangle

Determine Type of Window

Description

Determine whether a window is a rectangle, a polygonal region, or a binary mask.

Usage

```
is.rectangle(w)
is.polygonal(w)
is.mask(w)
```

Arguments

w Window to be inspected. An object of class "owin".

Details

These simple functions determine whether a window w (object of class "owin") is a rectangle (`is.rectangle(w) = TRUE`), a domain with polygonal boundary (`is.polygonal(w) = TRUE`), or a binary pixel mask (`is.mask(w) = TRUE`).

Value

Logical value, equal to TRUE if w is a window of the specified type.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[owin](#)

is.stationary*Recognise Stationary and Poisson Point Process Models*

Description

Given a point process model that has been fitted to data, determine whether the model is a stationary point process, and whether it is a Poisson point process.

Usage

```
is.stationary(x)
## 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 'rmhmodel'
is.stationary(x)
is.poisson(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 'rmhmodel'
is.poisson(x)
## S3 method for class 'interact'
is.poisson(x)
```

Arguments

x A fitted spatial point process model (object of class "ppm", "kppm" or "slrm") or 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.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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[is.marked](#) to determine whether a model is a marked point process.
[summary.ppm](#) for detailed information.
Model-fitting functions [ppm](#), [kppm](#), [slrm](#).

Examples

```
data(cells)
data(redwood)

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)
```

is.subset.owin

Determine Whether One Window is Contained In Another

Description

Tests whether window A is a subset of window B.

Usage

```
is.subset.owin(A, B)
```

Arguments

- | | |
|---|--------------------------------|
| A | A window object (see Details). |
| B | A window object (see Details). |

Details

This function tests whether the window A is a subset of the window B.

The arguments A and B must be window objects (either objects of class "owin", or data that can be coerced to this class by [as.owin](#)).

Various algorithms are used, depending on the geometrical type of the two windows.

Note that if B is not rectangular, the algorithm proceeds by discretising A, converting it to a pixel mask using [as.mask](#). In this case the resulting answer is only "approximately correct". The accuracy of the approximation can be controlled: see [as.mask](#).

Value

Logical scalar; TRUE if A is a sub-window of B, otherwise FALSE.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

Examples

```
w1 <- as.owin(c(0,1,0,1))
w2 <- as.owin(c(-1,2,-1,2))
is.subset.owin(w1,w2) # Returns TRUE.
is.subset.owin(w2,w1) # Returns FALSE.
```

Description

Compute various summary functions for a point pattern using a point-and-click interface.

Usage

```
istat(x, xname)
```

Arguments

- | | |
|-------|---|
| x | The spatial point pattern to be analysed. An object of class "ppp". |
| xname | Optional. Character string to use as the title of the dataset. |

Details

This command launches an interactive (point-and-click) interface which offers a choice of spatial summary functions that can be applied to the point pattern x.

The selected summary function is computed for the point pattern x and plotted in a popup window.

The selection of functions includes [Kest](#), [Lest](#), [pcf](#), [Fest](#), [Gest](#) and [Jest](#). For the function [pcf](#) it is possible to control the bandwidth parameter bw.

There is also an option to show simulation envelopes of the summary function.

Value

NULL.

Note

Before adjusting the bandwidth parameter bw, it is advisable to select *No simulation envelopes* to save a lot of computation time.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[iplot](#)

Examples

```
if(interactive()) {  
  data(swedishpines)  
  istat(swedishpines)  
}
```

japanesepines

Japanese Pines Point Pattern

Description

The data give the locations of Japanese black pine saplings in a square sampling region in a natural forest. The observations were originally collected by Numata (1961).

These data are used as a standard example in the textbook of Diggle (2003); see pages 1, 14, 19, 22, 24, 56–57 and 61.

Usage

```
data(japanesepines)
```

Format

An object of class "ppp" representing the point pattern of tree locations in a 5.7 x 5.7 metre square, rescaled to the unit square and rounded to two decimal places.

See [ppp.object](#) for details of the format of a point pattern object.

Source

Diggle (2003), obtained from Numata (1961)

References

- Diggle, P.J. (2003) *Statistical Analysis of Spatial Point Patterns*. Arnold Publishers.
- Numata, M. (1961) Forest vegetation in the vicinity of Choshi. Coastal flora and vegetation at Choshi, Chiba Prefecture. IV. *Bulletin of Choshi Marine Laboratory, Chiba University* **3**, 28–48 (in Japanese).

Description

For a multitype point pattern, estimate the multitype J function summarising the interpoint dependence between points of type i and of type j .

Usage

```
Jcross(X, i, j, eps=NULL, r=NULL, breaks=NULL, ..., correction=NULL)
```

Arguments

X	The observed point pattern, from which an estimate of the multitype J function $J_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of <code>marks(X)</code> .
j	The type (mark value) of the points in X to which distances are measured. A character string (or something that will be converted to a character string). Defaults to the second level of <code>marks(X)</code> .
<code>eps</code>	A positive number. The resolution of the discrete approximation to Euclidean distance (see below). There is a sensible default.
<code>r</code>	Optional. Numeric vector. The values of the argument r at which the function $J_{ij}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r .
<code>breaks</code>	An alternative to the argument <code>r</code> . Not normally invoked by the user. See the Details section.
<code>...</code>	Ignored.
<code>correction</code>	Optional. Character string specifying the edge correction(s) to be used. Options are "none", "rs", "km", "Hanisch" and "best".

Details

This function `Jcross` and its companions `Jdot` and `Jmulti` are generalisations of the function `Jest` to multitype point patterns.

A multitype point pattern is a spatial pattern of points classified into a finite number of possible “colours” or “types”. In the `spatstat` package, a multitype pattern is represented as a single point

pattern object in which the points carry marks, and the mark value attached to each point determines the type of that point.

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to [as.ppp](#). It must be a marked point pattern, and the mark vector Xmarks$ must be a factor. The argument i will be interpreted as a level of the factor Xmarks$. (Warning: this means that an integer value $i=3$ will be interpreted as the number 3, **not** the 3rd smallest level).

The “type i to type j ” multitype J function of a stationary multitype point process X was introduced by Van Lieshout and Baddeley (1999). It is defined by

$$J_{ij}(r) = \frac{1 - G_{ij}(r)}{1 - F_j(r)}$$

where $G_{ij}(r)$ is the distribution function of the distance from a type i point to the nearest point of type j , and $F_j(r)$ is the distribution function of the distance from a fixed point in space to the nearest point of type j in the pattern.

An estimate of $J_{ij}(r)$ is a useful summary statistic in exploratory data analysis of a multitype point pattern. If the subprocess of type i points is independent of the subprocess of points of type j , then $J_{ij}(r) \equiv 1$. Hence deviations of the empirical estimate of J_{ij} from the value 1 may suggest dependence between types.

This algorithm estimates $J_{ij}(r)$ from the point pattern X . It assumes that X can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in X as $X>window$) may have arbitrary shape. Biases due to edge effects are treated in the same manner as in [Jest](#), using the Kaplan-Meier and border corrections. The main work is done by [Gmulti](#) and [Fest](#).

The argument r is the vector of values for the distance r at which $J_{ij}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must exceed the radius of the largest disc contained in the window.

Value

An object of class "fv" (see [fv.object](#)).

Essentially a data frame containing six numeric columns

J	the recommended estimator of $J_{ij}(r)$, currently the Kaplan-Meier estimator.
r	the values of the argument r at which the function $J_{ij}(r)$ has been estimated
km	the Kaplan-Meier estimator of $J_{ij}(r)$
rs	the “reduced sample” or “border correction” estimator of $J_{ij}(r)$
han	the Hanisch-style estimator of $J_{ij}(r)$
un	the “uncorrected” estimator of $J_{ij}(r)$ formed by taking the ratio of uncorrected empirical estimators of $1 - G_{ij}(r)$ and $1 - F_j(r)$, see Gdot and Fest .
$theo$	the theoretical value of $J_{ij}(r)$ for a marked Poisson process, namely 1.

The result also has two attributes "G" and "F" which are respectively the outputs of [Gcross](#) and [Fest](#) for the point pattern.

Warnings

The arguments i and j are always interpreted as levels of the factor Xmarks$. They are converted to character strings if they are not already character strings. The value $i=1$ does **not** refer to the first level of the factor.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Van Lieshout, M.N.M. and Baddeley, A.J. (1996) A nonparametric measure of spatial interaction in point patterns. *Statistica Neerlandica* **50**, 344–361.
- Van Lieshout, M.N.M. and Baddeley, A.J. (1999) Indices of dependence between types in multivariate point patterns. *Scandinavian Journal of Statistics* **26**, 511–532.

See Also

[Jdot](#), [Jest](#), [Jmulti](#)

Examples

```
# Lansing woods data: 6 types of trees
data(lansing)

Jhm <- Jcross(lansing, "hickory", "maple")
# diagnostic plot for independence between hickories and maples
plot(Jhm)

# synthetic example with two types "a" and "b"
pp <- runifpoint(30) %mark% factor(sample(c("a", "b"), 30, replace=TRUE))
J <- Jcross(pp)
```

Jdot

Multitype J Function (i-to-any)

Description

For a multitype point pattern, estimate the multitype J function summarising the interpoint dependence between the type i points and the points of any type.

Usage

```
Jdot(X, i, eps=NULL, r=NULL, breaks=NULL, ..., correction=NULL)
```

Arguments

- | | |
|-----|---|
| X | The observed point pattern, from which an estimate of the multitype J function $J_{i\bullet}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details. |
| i | The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of <code>marks(X)</code> . |
| eps | A positive number. The resolution of the discrete approximation to Euclidean distance (see below). There is a sensible default. |

r	numeric vector. The values of the argument r at which the function $J_{i\bullet}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r .
breaks	An alternative to the argument r . Not normally invoked by the user. See the Details section.
...	Ignored.
correction	Optional. Character string specifying the edge correction(s) to be used. Options are "none", "rs", "km", "Hanisch" and "best".

Details

This function `Jdot` and its companions `Jcross` and `Jmulti` are generalisations of the function `Jest` to multitype point patterns.

A multitype point pattern is a spatial pattern of points classified into a finite number of possible “colours” or “types”. In the `spatstat` package, a multitype pattern is represented as a single point pattern object in which the points carry marks, and the mark value attached to each point determines the type of that point.

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to `as.ppp`. It must be a marked point pattern, and the mark vector Xmarks$ must be a factor. The argument i will be interpreted as a level of the factor Xmarks$. (Warning: this means that an integer value $i=3$ will be interpreted as the number 3, **not** the 3rd smallest level.)

The “type i to any type” multitype J function of a stationary multitype point process X was introduced by Van Lieshout and Baddeley (1999). It is defined by

$$J_{i\bullet}(r) = \frac{1 - G_{i\bullet}(r)}{1 - F_\bullet(r)}$$

where $G_{i\bullet}(r)$ is the distribution function of the distance from a type i point to the nearest other point of the pattern, and $F_\bullet(r)$ is the distribution function of the distance from a fixed point in space to the nearest point of the pattern.

An estimate of $J_{i\bullet}(r)$ is a useful summary statistic in exploratory data analysis of a multitype point pattern. If the pattern is a marked Poisson point process, then $J_{i\bullet}(r) \equiv 1$. If the subprocess of type i points is independent of the subprocess of points of all types not equal to i , then $J_{i\bullet}(r)$ equals $J_{ii}(r)$, the ordinary J function (see `Jest` and Van Lieshout and Baddeley (1996)) of the points of type i . Hence deviations from zero of the empirical estimate of $J_{i\bullet} - J_{ii}$ may suggest dependence between types.

This algorithm estimates $J_{i\bullet}(r)$ from the point pattern X . It assumes that X can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in X as $X>window$) may have arbitrary shape. Biases due to edge effects are treated in the same manner as in `Jest`, using the Kaplan-Meier and border corrections. The main work is done by `Gmulti` and `Fest`.

The argument r is the vector of values for the distance r at which $J_{i\bullet}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must exceed the radius of the largest disc contained in the window.

Value

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

Essentially a data frame containing six numeric columns

J the recommended estimator of $J_{i\bullet}(r)$, currently the Kaplan-Meier estimator.

r	the values of the argument r at which the function $J_{i\bullet}(r)$ has been estimated
km	the Kaplan-Meier estimator of $J_{i\bullet}(r)$
rs	the “reduced sample” or “border correction” estimator of $J_{i\bullet}(r)$
han	the Hanisch-style estimator of $J_{i\bullet}(r)$
un	the “uncorrected” estimator of $J_{i\bullet}(r)$ formed by taking the ratio of uncorrected empirical estimators of $1 - G_{i\bullet}(r)$ and $1 - F_{\bullet}(r)$, see Gdot and Fest .
theo	the theoretical value of $J_{i\bullet}(r)$ for a marked Poisson process, namely 1.

The result also has two attributes "G" and "F" which are respectively the outputs of [Gdot](#) and [Fest](#) for the point pattern.

Warnings

The argument *i* is interpreted as a level of the factor *X\$marks*. It is converted to a character string if it is not already a character string. The value *i*=1 does **not** refer to the first level of the factor.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Van Lieshout, M.N.M. and Baddeley, A.J. (1996) A nonparametric measure of spatial interaction in point patterns. *Statistica Neerlandica* **50**, 344–361.
 Van Lieshout, M.N.M. and Baddeley, A.J. (1999) Indices of dependence between types in multivariate point patterns. *Scandinavian Journal of Statistics* **26**, 511–532.

See Also

[Jcross](#), [Jest](#), [Jmulti](#)

Examples

```
# Lansing woods data: 6 types of trees
data(lansing)

Jh. <- Jdot(lansing, "hickory")
plot(Jh.)
# diagnostic plot for independence between hickories and other trees
Jhh <- Jest(lansing[lansing$marks == "hickory", ])
plot(Jhh, add=TRUE, legendpos="bottom")

## Not run:
# synthetic example with two marks "a" and "b"
pp <- runifpoint(30) %mark% factor(sample(c("a", "b"), 30, replace=TRUE))
J <- Jdot(pp, "a")

## End(Not run)
```

Jest	<i>Estimate the J-function</i>
------	--------------------------------

Description

Estimates the summary function $J(r)$ for a point pattern in a window of arbitrary shape.

Usage

```
Jest(X, ..., eps=NULL, r=NULL, breaks=NULL, correction=NULL)
```

Arguments

<code>X</code>	The observed point pattern, from which an estimate of $J(r)$ will be computed. An object of class "ppp", or data in any format acceptable to as.ppp() .
<code>...</code>	Ignored.
<code>eps</code>	the resolution of the discrete approximation to Euclidean distance (see below). There is a sensible default.
<code>r</code>	vector of values for the argument r at which $J(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on <code>r</code> .
<code>breaks</code>	An alternative to the argument <code>r</code> . Not normally invoked by the user. See Details section.
<code>correction</code>	Optional. Character string specifying the choice of edge correction(s) in Fest and Gest .

Details

The J function (Van Lieshout and Baddeley, 1996) of a stationary point process is defined as

$$J(r) = \frac{1 - G(r)}{1 - F(r)}$$

where $G(r)$ is the nearest neighbour distance distribution function of the point process (see [Gest](#)) and $F(r)$ is its empty space function (see [Fest](#)).

For a completely random (uniform Poisson) point process, the J -function is identically equal to 1. Deviations $J(r) < 1$ or $J(r) > 1$ typically indicate spatial clustering or spatial regularity, respectively. The J -function is one of the few characteristics that can be computed explicitly for a wide range of point processes. See Van Lieshout and Baddeley (1996), Baddeley et al (2000), Thonnes and Van Lieshout (1999) for further information.

An estimate of J derived from a spatial point pattern dataset can be used in exploratory data analysis and formal inference about the pattern. The estimate of $J(r)$ is compared against the constant function 1. Deviations $J(r) < 1$ or $J(r) > 1$ may suggest spatial clustering or spatial regularity, respectively.

This algorithm estimates the J -function from the point pattern `X`. It assumes that `X` can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in `X` as `X>window`) may have arbitrary shape.

The argument X is interpreted as a point pattern object (of class "ppp", see [ppp.object](#)) and can be supplied in any of the formats recognised by [as.ppp\(\)](#).

The functions [Fest](#) and [Gest](#) are called to compute estimates of $F(r)$ and $G(r)$ respectively. These estimates are then combined by simply taking the ratio $J(r) = (1 - G(r))/(1 - F(r))$.

In fact three different estimates are computed using different edge corrections (Baddeley, 1998). The Kaplan-Meier estimate (returned as km) is the ratio $J = (1-G)/(1-F)$ of the Kaplan-Meier estimates of $1 - F$ and $1 - G$ computed by [Fest](#) and [Gest](#) respectively. The reduced-sample or border corrected estimate (returned as rs) is the same ratio $J = (1-G)/(1-F)$ of the border corrected estimates. These estimators are slightly biased for J , since they are ratios of approximately unbiased estimators. The logarithm of the Kaplan-Meier estimate is unbiased for $\log J$.

The uncorrected estimate (returned as un) is the ratio $J = (1-G)/(1-F)$ of the uncorrected ("raw") estimates of the survival functions of F and G , which are the empirical distribution functions of the empty space distances [Fest\(X, ...\)\\$raw](#) and of the nearest neighbour distances [Gest\(X, ...\)\\$raw](#). The uncorrected estimates of F and G are severely biased. However the uncorrected estimate of J is approximately unbiased (if the process is close to Poisson); it is insensitive to edge effects, and should be used when edge effects are severe (see Baddeley et al, 2000).

The algorithm for [Fest](#) uses two discrete approximations which are controlled by the parameter `eps` and by the spacing of values of `r` respectively. See [Fest](#) for details. First-time users are strongly advised not to specify these arguments.

Note that the value returned by [Jest](#) includes the output of [Fest](#) and [Gest](#) as attributes (see the last example below). If the user is intending to compute the F , G and J functions for the point pattern, it is only necessary to call [Jest](#).

Value

An object of class "fv", see [fv.object](#), which can be plotted directly using [plot.fv](#).

Essentially a data frame containing

<code>r</code>	the vector of values of the argument r at which the function J has been estimated
<code>J</code>	the recommended estimate of $J(r)$, which is the Kaplan-Meier estimate km
<code>rs</code>	the "reduced sample" or "border correction" estimator of $J(r)$ computed from the border-corrected estimates of F and G
<code>km</code>	the spatial Kaplan-Meier estimator of $J(r)$ computed from the Kaplan-Meier estimates of F and G
<code>han</code>	the Hanisch-style estimator of $J(r)$ computed from the Hanisch estimate of G and the Chiu-Stoyan estimate of F
<code>un</code>	the uncorrected estimate of $J(r)$ computed from the uncorrected estimates of F and G
<code>theo</code>	the theoretical value of $J(r)$ for a stationary Poisson process: identically equal to 1

The data frame also has **attributes**

<code>F</code>	the output of Fest for this point pattern, containing three estimates of the empty space function $F(r)$ and an estimate of its hazard function
<code>G</code>	the output of Gest for this point pattern, containing three estimates of the nearest neighbour distance distribution function $G(r)$ and an estimate of its hazard function

Note

Sizeable amounts of memory may be needed during the calculation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Baddeley, A.J. Spatial sampling and censoring. In O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*. Chapman and Hall, 1998. Chapter 2, pages 37–78.
- Baddeley, A.J. and Gill, R.D. The empty space hazard of a spatial pattern. Research Report 1994/3, Department of Mathematics, University of Western Australia, May 1994.
- Baddeley, A.J. and Gill, R.D. Kaplan-Meier estimators of interpoint distance distributions for spatial point processes. *Annals of Statistics* **25** (1997) 263–292.
- Baddeley, A., Kerscher, M., Schladitz, K. and Scott, B.T. Estimating the J function without edge correction. *Statistica Neerlandica* **54** (2000) 315–328.
- Borgefors, G. Distance transformations in digital images. *Computer Vision, Graphics and Image Processing* **34** (1986) 344–371.
- Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.
- Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.
- Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.
- Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.
- Thonnes, E. and Van Lieshout, M.N.M. A comparative study on the power of Van Lieshout and Baddeley's J -function. *Biometrical Journal* **41** (1999) 721–734.
- Van Lieshout, M.N.M. and Baddeley, A.J. A nonparametric measure of spatial interaction in point patterns. *Statistica Neerlandica* **50** (1996) 344–361.

See Also

[Fest](#), [Gest](#), [Kest](#), [km.rs](#), [reduced.sample](#), [kaplan.meier](#)

Examples

```
data(cells)
J <- Jest(cells, 0.01)
plot(J, main="cells data")
# values are far above J = 1, indicating regular pattern

data(redwood)
J <- Jest(redwood, 0.01, legendpos="center")
plot(J, main="redwood data")
# values are below J = 1, indicating clustered pattern
```

Jmulti*Marked J Function***Description**

For a marked point pattern, estimate the multitype J function summarising dependence between the points in subset I and those in subset J .

Usage

```
Jmulti(X, I, J, eps=NULL, r=NULL, breaks=NULL, ..., disjoint=NULL,
correction=NULL)
```

Arguments

X	The observed point pattern, from which an estimate of the multitype distance distribution function $J_{IJ}(r)$ will be computed. It must be a marked point pattern. See under Details.
I	Subset of points of X from which distances are measured. See Details.
J	Subset of points in X to which distances are measured. See Details.
eps	A positive number. The pixel resolution of the discrete approximation to Euclidean distance (see Jest). There is a sensible default.
r	numeric vector. The values of the argument r at which the distribution function $J_{IJ}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r .
breaks	An alternative to the argument r . Not normally invoked by the user. See the Details section.
...	Ignored.
disjoint	Optional flag indicating whether the subsets I and J are disjoint. If missing, this value will be computed by inspecting the vectors I and J.
correction	Optional. Character string specifying the edge correction(s) to be used. Options are "none", "rs", "km", "Hanisch" and "best".

Details

The function **Jmulti** generalises [Jest](#) (for unmarked point patterns) and [Jdot](#) and [Jcross](#) (for multitype point patterns) to arbitrary marked point patterns.

Suppose X_I, X_J are subsets, possibly overlapping, of a marked point process. Define

$$J_{IJ}(r) = \frac{1 - G_{IJ}(r)}{1 - F_J(r)}$$

where $F_J(r)$ is the cumulative distribution function of the distance from a fixed location to the nearest point of X_J , and $G_{IJ}(r)$ is the distribution function of the distance from a typical point of X_I to the nearest distinct point of X_J .

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to [as.ppp](#).

The arguments I and J specify two subsets of the point pattern. They may be any type of subset indices, for example, logical vectors of length equal to `npoints(X)`, or integer vectors with entries in the range 1 to `npoints(X)`, or negative integer vectors.

Alternatively, I and J may be **functions** that will be applied to the point pattern X to obtain index vectors. If I is a function, then evaluating $I(X)$ should yield a valid subset index. This option is useful when generating simulation envelopes using `envelope`.

It is assumed that X can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in X as $X>window$) may have arbitrary shape. Biases due to edge effects are treated in the same manner as in `Jest`.

The argument r is the vector of values for the distance r at which $J_{IJ}(r)$ should be evaluated. It is also used to determine the breakpoints (in the sense of `hist`) for the computation of histograms of distances. The reduced-sample and Kaplan-Meier estimators are computed from histogram counts. In the case of the Kaplan-Meier estimator this introduces a discretisation error which is controlled by the fineness of the breakpoints.

First-time users would be strongly advised not to specify r . However, if it is specified, r must satisfy $r[1] = 0$, and $\max(r)$ must be larger than the radius of the largest disc contained in the window. Furthermore, the successive entries of r must be finely spaced.

Value

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

Essentially a data frame containing six numeric columns

r	the values of the argument r at which the function $J_{IJ}(r)$ has been estimated
rs	the "reduced sample" or "border correction" estimator of $J_{IJ}(r)$
km	the spatial Kaplan-Meier estimator of $J_{IJ}(r)$
han	the Hanisch-style estimator of $J_{IJ}(r)$
un	the uncorrected estimate of $J_{IJ}(r)$, formed by taking the ratio of uncorrected empirical estimators of $1 - G_{IJ}(r)$ and $1 - F_J(r)$, see <code>Gdot</code> and <code>Fest</code> .
$theo$	the theoretical value of $J_{IJ}(r)$ for a marked Poisson process with the same estimated intensity, namely 1.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

Van Lieshout, M.N.M. and Baddeley, A.J. (1999) Indices of dependence between types in multivariate point patterns. *Scandinavian Journal of Statistics* **26**, 511–532.

See Also

`Jcross`, `Jdot`, `Jest`

Examples

```
data(longleaf)
# Longleaf Pine data: marks represent diameter

Jm <- Jmulti(longleaf, marks(longleaf) <= 15, marks(longleaf) >= 25)
plot(Jm)
```

K3est

K-function of a Three-Dimensional Point Pattern

Description

Estimates the K -function from a three-dimensional point pattern.

Usage

```
K3est(X, ..., rmax = NULL, nrval = 128, correction = c("translation", "isotropic"))
```

Arguments

<code>X</code>	Three-dimensional point pattern (object of class "pp3").
<code>...</code>	Ignored.
<code>rmax</code>	Optional. Maximum value of argument r for which $K_3(r)$ will be estimated.
<code>nrval</code>	Optional. Number of values of r for which $K_3(r)$ will be estimated. A large value of <code>nrval</code> is required to avoid discretisation effects.
<code>correction</code>	Optional. Character vector specifying the edge correction(s) to be applied. See Details.

Details

For a stationary point process Φ in three-dimensional space, the three-dimensional K function is

$$K_3(r) = \frac{1}{\lambda} E(N(\Phi, x, r) \mid x \in \Phi)$$

where λ is the intensity of the process (the expected number of points per unit volume) and $N(\Phi, x, r)$ is the number of points of Φ , other than x itself, which fall within a distance r of x . This is the three-dimensional generalisation of Ripley's K function for two-dimensional point processes (Ripley, 1977).

The three-dimensional point pattern X is assumed to be a partial realisation of a stationary point process Φ . The distance between each pair of distinct points is computed. The empirical cumulative distribution function of these values, with appropriate edge corrections, is renormalised to give the estimate of $K_3(r)$.

The available edge corrections are:

- "translation": the Ohser translation correction estimator (Ohser, 1983; Baddeley et al, 1993)
- "isotropic": the three-dimensional counterpart of Ripley's isotropic edge correction (Ripley, 1977; Baddeley et al, 1993).

Value

A function value table (object of class "fv") that can be plotted, printed or coerced to a data frame containing the function values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rana Moyeed.

References

- Baddeley, A.J., Moyeed, R.A., Howard, C.V. and Boyde, A. (1993) Analysis of a three-dimensional point pattern with replication. *Applied Statistics* **42**, 641–668.
- Ohser, J. (1983) On estimators for the reduced second moment measure of point processes. *Mathematische Operationsforschung und Statistik, series Statistics*, **14**, 63 – 71.
- Ripley, B.D. (1977) Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, Series B*, **39**, 172 – 212.

See Also

[F3est](#), [G3est](#), [pcf3est](#)

Examples

```
X <- rpoispp3(42)
Z <- K3est(X)
if(interactive()) plot(Z)
```

kaplan.meier

Kaplan-Meier Estimator using Histogram Data

Description

Compute the Kaplan-Meier estimator of a survival time distribution function, from histogram data

Usage

```
kaplan.meier(obs, nco, breaks, upperobs=0)
```

Arguments

- | | |
|----------|---|
| obs | vector of n integers giving the histogram of all observations (censored or uncensored survival times) |
| nco | vector of n integers giving the histogram of uncensored observations (those survival times that are less than or equal to the censoring time) |
| breaks | Vector of $n + 1$ breakpoints which were used to form both histograms. |
| upperobs | Number of observations beyond the rightmost breakpoint, if any. |

Details

This function is needed mainly for internal use in **spatstat**, but may be useful in other applications where you want to form the Kaplan-Meier estimator from a huge dataset.

Suppose T_i are the survival times of individuals $i = 1, \dots, M$ with unknown distribution function $F(t)$ which we wish to estimate. Suppose these times are right-censored by random censoring times C_i . Thus the observations consist of right-censored survival times $\tilde{T}_i = \min(T_i, C_i)$ and non-censoring indicators $D_i = 1\{T_i \leq C_i\}$ for each i .

If the number of observations M is large, it is efficient to use histograms. Form the histogram `obs` of all observed times \tilde{T}_i . That is, `obs[k]` counts the number of values \tilde{T}_i in the interval `(breaks[k], breaks[k+1])` for $k > 1$ and `[breaks[1], breaks[2]]` for $k = 1$. Also form the histogram `nco` of all uncensored times, i.e. those \tilde{T}_i such that $D_i = 1$. These two histograms are the arguments passed to `kaplan.meier`.

The vectors `km` and `lambda` returned by `kaplan.meier` are (histogram approximations to) the Kaplan-Meier estimator of $F(t)$ and its hazard rate $\lambda(t)$. Specifically, `km[k]` is an estimate of $F(breaks[k+1])$, and `lambda[k]` is an estimate of the average of $\lambda(t)$ over the interval `(breaks[k], breaks[k+1])`.

The histogram breaks must include 0. If the histogram breaks do not span the range of the observations, it is important to count how many survival times \tilde{T}_i exceed the rightmost breakpoint, and give this as the value `upperobs`.

Value

A list with two elements:

<code>km</code>	Kaplan-Meier estimate of the survival time c.d.f. $F(t)$
<code>lambda</code>	corresponding Nelson-Aalen estimate of the hazard rate $\lambda(t)$

These are numeric vectors of length n .

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[reduced.sample](#), [km.rs](#)

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,
      trend = ~1, interaction = Poisson(), rbord = reach(interaction),
      compute.var = TRUE,
      truecoef = NULL, hi.res = NULL)
```

Arguments

<code>object</code>	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").
<code>r</code>	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.
<code>breaks</code>	Optional alternative to <code>r</code> for advanced use.
<code>...</code>	Ignored.
<code>correction</code>	Optional vector of character strings specifying the edge correction(s) to be used. See Kest for options.
<code>conditional</code>	Optional. Logical value indicating whether to compute the estimates for the conditional case. See Details.
<code>restrict</code>	Logical value indicating whether to compute the restriction estimator (<code>restrict=TRUE</code>) or the reweighting estimator (<code>restrict=FALSE</code> , the default). Applies only if <code>conditional=TRUE</code> . See Details.
<code>trend, interaction, rbord</code>	Optional. Arguments passed to ppm to fit a point process model to the data, if <code>object</code> is a point pattern. See ppm for details.
<code>compute.var</code>	Logical value indicating whether to compute the Poincare variance bound for the residual K function (calculation is only implemented for the isotropic correction).
<code>truecoef</code>	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 <code>hi.res</code> .
<code>hi.res</code>	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:

<code>correction</code>	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

$$\hat{K}(r) = \sum_i \sum_{j < i} e(x_i, x_j, r, x) I\{d(x_i, x_j) \leq r\}$$

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

$$C \tilde{K}(r) = \int_W \lambda(u, x) \sum_j e(u, x_j, r, x \cup u) I\{d(u, x_j) \leq r\}$$

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 Moller (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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> Ege Rubak and Jesper Moller.

References

Baddeley, A., Rubak, E. and Moller, J. (2011) Score, pseudo-score and residual diagnostics for spatial point process models. *Statistical Science* **26**, 613–646.

See Also

Related functions: [Kres](#), [Kest](#).

Alternative functions: [Gcom](#), [psstG](#), [psstA](#), [psst](#).

Point process models: [ppm](#).

Examples

```
fit0 <- ppm(cells, ~1) # uniform Poisson

if(interactive()) {
  plot(Kcom(fit0))
# compare the isotropic-correction estimates
  plot(Kcom(fit0), cbind(iso, icom) ~ r)
# uniform Poisson is clearly not correct
}

fit1 <- ppm(cells, ~1, Strauss(0.08))

K1 <- Kcom(fit1)
K1
if(interactive()) {
  plot(K1)
  plot(K1, cbind(iso, icom) ~ r)
  plot(K1, cbind(trans, tcom) ~ r)
# how to plot the difference between nonparametric estimates and compensators
  plot(K1, iso - icom ~ r)
# fit looks approximately OK; try adjusting interaction distance
}
fit2 <- ppm(cells, ~1, Strauss(0.12))

K2 <- Kcom(fit2)
if(interactive()) {
  plot(K2)
  plot(K2, cbind(iso, icom) ~ r)
  plot(K2, iso - icom ~ r)
}
```

Kcross

*Multitype K Function (Cross-type)***Description**

For a multitype point pattern, estimate the multitype K function which counts the expected number of points of type j within a given distance of a point of type i .

Usage

```
Kcross(X, i, j, r=NULL, breaks=NULL, correction, ..., ratio=FALSE)
```

Arguments

X	The observed point pattern, from which an estimate of the cross type K function $K_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of <code>marks(X)</code> .
j	The type (mark value) of the points in X to which distances are measured. A character string (or something that will be converted to a character string). Defaults to the second level of <code>marks(X)</code> .
r	numeric vector. The values of the argument r at which the distribution function $K_{ij}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r .
<code>breaks</code>	An alternative to the argument r . Not normally invoked by the user. See the Details section.
<code>correction</code>	A character vector containing any selection of the options "border", "bord.modif", "isotropic", "Ripley", "translate", "none" or "best". It specifies the edge correction(s) to be applied.
\dots	Ignored.
<code>ratio</code>	Logical. If <code>TRUE</code> , the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.

Details

This function `Kcross` and its companions `Kdot` and `Kmulti` are generalisations of the function `Kest` to multitype point patterns.

A multitype point pattern is a spatial pattern of points classified into a finite number of possible “colours” or “types”. In the `spatstat` package, a multitype pattern is represented as a single point pattern object in which the points carry marks, and the mark value attached to each point determines the type of that point.

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to `as.ppp`. It must be a marked point pattern, and the mark vector Xmarks$ must be a factor.

The arguments i and j will be interpreted as levels of the factor Xmarks$. If i and j are missing, they default to the first and second level of the marks factor, respectively.

The “cross-type” (type i to type j) K function of a stationary multitype point process X is defined so that $\lambda_j K_{ij}(r)$ equals the expected number of additional random points of type j within a distance r of a typical point of type i in the process X . Here λ_j is the intensity of the type j points, i.e. the expected number of points of type j per unit area. The function K_{ij} is determined by the second order moment properties of X .

An estimate of $K_{ij}(r)$ is a useful summary statistic in exploratory data analysis of a multitype point pattern. If the process of type i points were independent of the process of type j points, then $K_{ij}(r)$ would equal πr^2 . Deviations between the empirical K_{ij} curve and the theoretical curve πr^2 may suggest dependence between the points of types i and j .

This algorithm estimates the distribution function $K_{ij}(r)$ from the point pattern X . It assumes that X can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in X as $X>window$) may have arbitrary shape. Biases due to edge effects are treated in the same manner as in [Kest](#), using the border correction.

The argument r is the vector of values for the distance r at which $K_{ij}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must not exceed the radius of the largest disc contained in the window.

The pair correlation function can also be applied to the result of [Kcross](#); see [pcf](#).

Value

An object of class "fv" (see [fv.object](#)).

Essentially a data frame containing numeric columns

r	the values of the argument r at which the function $K_{ij}(r)$ has been estimated
theo	the theoretical value of $K_{ij}(r)$ for a marked Poisson process, namely πr^2

together with a column or columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $K_{ij}(r)$ obtained by the edge corrections named.

If $\text{ratio}=\text{TRUE}$ then the return value also has two attributes called "numerator" and "denominator" which are "fv" objects containing the numerators and denominators of each estimate of $K(r)$.

Warnings

The arguments i and j are always interpreted as levels of the factor Xmarks$. They are converted to character strings if they are not already character strings. The value $i=1$ does **not** refer to the first level of the factor.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.
- Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.
- Harkness, R.D and Isham, V. (1983) A bivariate spatial point pattern of ants' nests. *Applied Statistics* **32**, 293–303

Lotwick, H. W. and Silverman, B. W. (1982). Methods for analysing spatial processes of several types of points. *J. Royal Statist. Soc. Ser. B* **44**, 406–413.

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

Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.

See Also

`Kdot`, `Kest`, `Kmulti`, `pcf`

Examples

```
data(betacells)
# cat retina data
K01 <- Kcross(betacells, "off", "on")
plot(K01)

## Not run:
K10 <- Kcross(betacells, "on", "off")

# synthetic example: point pattern with marks 0 and 1
pp <- runifpoispp(50)
pp <- pp %mark% factor(sample(0:1, npoints(pp), replace=TRUE))
K <- Kcross(pp, "0", "1")
K <- Kcross(pp, 0, 1) # equivalent

## End(Not run)
```

Kcross.inhom

Inhomogeneous Cross K Function

Description

For a multitype point pattern, estimate the inhomogeneous version of the cross K function, which counts the expected number of points of type j within a given distance of a point of type i , adjusted for spatially varying intensity.

Usage

```
Kcross.inhom(X, i, j, lambdaI=NULL, lambdaJ=NULL, ..., r=NULL, breaks=NULL,
            correction = c("border", "isotropic", "Ripley", "translate"),
            sigma=NULL, varcov=NULL,
            lambdaIJ=NULL)
```

Arguments

- | | |
|----------------|--|
| <code>X</code> | The observed point pattern, from which an estimate of the inhomogeneous cross type K function $K_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details. |
| <code>i</code> | The type (mark value) of the points in <code>X</code> from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of <code>marks(X)</code> . |

j	The type (mark value) of the points in X to which distances are measured. A character string (or something that will be converted to a character string). Defaults to the second level of marks(X).
lambdaI	Optional. Values of the estimated intensity of the sub-process of points of type i. Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the type i points in X, or a function(x,y) which can be evaluated to give the intensity value at any location.
lambdaJ	Optional. Values of the estimated intensity of the sub-process of points of type j. Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the type j points in X, or a function(x,y) which can be evaluated to give the intensity value at any location.
r	Optional. Numeric vector giving the values of the argument r at which the cross K function $K_{ij}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r.
breaks	Optional. An alternative to the argument r. Not normally invoked by the user. See the Details section.
correction	A character vector containing any selection of the options "border", "bord.modif", "isotropic", "Ripley", "translate", "none" or "best". It specifies the edge correction(s) to be applied.
...	Ignored.
sigma	Standard deviation of isotropic Gaussian smoothing kernel, used in computing leave-one-out kernel estimates of lambdaI, lambdaJ if they are omitted.
varcov	Variance-covariance matrix of anisotropic Gaussian kernel, used in computing leave-one-out kernel estimates of lambdaI, lambdaJ if they are omitted. Incompatible with sigma.
lambdaIJ	Optional. A matrix containing estimates of the product of the intensities lambdaI and lambdaJ for each pair of points of types i and j respectively.

Details

This is a generalisation of the function [Kcross](#) to include an adjustment for spatially inhomogeneous intensity, in a manner similar to the function [Kinhom](#).

The inhomogeneous cross-type K function is described by Moller and Waagepetersen (2003, pages 48-49 and 51-53).

Briefly, given a multitype point process, suppose the sub-process of points of type j has intensity function $\lambda_j(u)$ at spatial locations u . Suppose we place a mass of $1/\lambda_j(\zeta)$ at each point ζ of type j . Then the expected total mass per unit area is 1. The inhomogeneous “cross-type” K function $K_{ij}^{\text{inhom}}(r)$ equals the expected total mass within a radius r of a point of the process of type i .

If the process of type i points were independent of the process of type j points, then $K_{ij}^{\text{inhom}}(r)$ would equal πr^2 . Deviations between the empirical K_{ij} curve and the theoretical curve πr^2 suggest dependence between the points of types i and j .

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to [as.ppp](#). It must be a marked point pattern, and the mark vector X\$marks must be a factor.

The arguments i and j will be interpreted as levels of the factor X\$marks. (Warning: this means that an integer value i=3 will be interpreted as the number 3, **not** the 3rd smallest level). If i and j are missing, they default to the first and second level of the marks factor, respectively.

The argument lambdaI supplies the values of the intensity of the sub-process of points of type i. It may be either

a pixel image (object of class "im") which gives the values of the type i intensity at all locations in the window containing X;

a numeric vector containing the values of the type i intensity evaluated only at the data points of type i. The length of this vector must equal the number of type i points in X.

a function which can be evaluated to give values of the intensity at any locations.

omitted: if `lambdaI` is omitted then it will be estimated using a leave-one-out kernel smoother.

If `lambdaI` is omitted, then it will be estimated using a ‘leave-one-out’ kernel smoother, as described in Baddeley, Moller and Waagepetersen (2000). The estimate of `lambdaI` for a given point is computed by removing the point from the point pattern, applying kernel smoothing to the remaining points using `density.ppp`, and evaluating the smoothed intensity at the point in question. The smoothing kernel bandwidth is controlled by the arguments `sigma` and `varcov`, which are passed to `density.ppp` along with any extra arguments.

Similarly `lambdaJ` should contain estimated values of the intensity of the sub-process of points of type j. It may be either a pixel image, a function, a numeric vector, or omitted.

The optional argument `lambdaIJ` is for advanced use only. It is a matrix containing estimated values of the products of these two intensities for each pair of data points of types i and j respectively.

The argument `r` is the vector of values for the distance r at which $K_{ij}(r)$ should be evaluated. The values of `r` must be increasing nonnegative numbers and the maximum `r` value must exceed the radius of the largest disc contained in the window.

The argument `correction` chooses the edge correction as explained e.g. in `Kest`.

The pair correlation function can also be applied to the result of `Kcross.inhom`; see `pcf`.

Value

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

Essentially a data frame containing numeric columns

`r` the values of the argument r at which the function $K_{ij}(r)$ has been estimated

`theo` the theoretical value of $K_{ij}(r)$ for a marked Poisson process, namely πr^2

together with a column or columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $K_{ij}(r)$ obtained by the edge corrections named.

Warnings

The arguments `i` and `j` are always interpreted as levels of the factor `X$marks`. They are converted to character strings if they are not already character strings. The value `i=1` does **not** refer to the first level of the factor.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

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

See Also

[Kcross](#), [Kinhom](#), [Kdot.inhom](#), [Kmulti.inhom](#), [pcf](#)

Examples

```
# Lansing Woods data
data(lansing)
lansing <- lansing[seq(1,lansing$n, by=10)]
ma <- split(lansing)$maple
wh <- split(lansing)$whiteoak

# method (1): estimate intensities by nonparametric smoothing
lambdaM <- density.ppp(ma, sigma=0.15, at="points")
lambdaW <- density.ppp(wh, sigma=0.15, at="points")
K <- Kcross.inhom(lansing, "whiteoak", "maple", lambdaW, lambdaM)

# method (2): leave-one-out
K <- Kcross.inhom(lansing, "whiteoak", "maple", sigma=0.15)

# method (3): fit parametric intensity model
fit <- ppm(lansing, ~marks * polynom(x,y,2))
# evaluate fitted intensities at data points
# (these are the intensities of the sub-processes of each type)
inten <- fitted(fit, dataonly=TRUE)
# split according to types of points
lambda <- split(inten, lansing$marks)
K <- Kcross.inhom(lansing, "whiteoak", "maple",
                   lambda$whiteoak, lambda$maple)

# synthetic example: type A points have intensity 50,
#                      type B points have intensity 100 * x
lamB <- as.im(function(x,y){50 + 100 * x}, owin())
X <- superimpose(A=runifpoispp(50), B=rpoispp(lamB))
K <- Kcross.inhom(X, "A", "B",
                   lambdaI=as.im(50, X>window), lambdaJ=lamB)
```

Kdot

Multitype K Function (*i-to-any*)

Description

For a multitype point pattern, estimate the multitype K function which counts the expected number of other points of the process within a given distance of a point of type i .

Usage

```
Kdot(X, i, r=NULL, breaks=NULL, correction, ..., ratio=FALSE)
```

Arguments

- | | |
|-----|---|
| X | The observed point pattern, from which an estimate of the multitype K function $K_{i\bullet}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details. |
|-----|---|

i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of marks(X).
r	numeric vector. The values of the argument r at which the distribution function $K_{i\bullet}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r .
breaks	An alternative to the argument r. Not normally invoked by the user. See the Details section.
correction	A character vector containing any selection of the options "border", "bord.modif", "isotropic", "Ripley", "translate", "none" or "best". It specifies the edge correction(s) to be applied.
...	Ignored.
ratio	Logical. If TRUE, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.

Details

This function `Kdot` and its companions `Kcross` and `Kmulti` are generalisations of the function `Kest` to multitype point patterns.

A multitype point pattern is a spatial pattern of points classified into a finite number of possible “colours” or “types”. In the `spatstat` package, a multitype pattern is represented as a single point pattern object in which the points carry marks, and the mark value attached to each point determines the type of that point.

The argument `X` must be a point pattern (object of class "ppp") or any data that are acceptable to `as.ppp`. It must be a marked point pattern, and the mark vector `X$marks` must be a factor.

The argument `i` will be interpreted as a level of the factor `X$marks`. If `i` is missing, it defaults to the first level of the marks factor, `i = levels(X$marks)[1]`.

The “type i to any type” multitype K function of a stationary multitype point process X is defined so that $\lambda K_{i\bullet}(r)$ equals the expected number of additional random points within a distance r of a typical point of type i in the process X . Here λ is the intensity of the process, i.e. the expected number of points of X per unit area. The function $K_{i\bullet}$ is determined by the second order moment properties of X .

An estimate of $K_{i\bullet}(r)$ is a useful summary statistic in exploratory data analysis of a multitype point pattern. If the subprocess of type i points were independent of the subprocess of points of all types not equal to i , then $K_{i\bullet}(r)$ would equal πr^2 . Deviations between the empirical $K_{i\bullet}$ curve and the theoretical curve πr^2 may suggest dependence between types.

This algorithm estimates the distribution function $K_{i\bullet}(r)$ from the point pattern `X`. It assumes that `X` can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in `X` as `X>window`) may have arbitrary shape. Biases due to edge effects are treated in the same manner as in `Kest`, using the border correction.

The argument `r` is the vector of values for the distance r at which $K_{i\bullet}(r)$ should be evaluated. The values of `r` must be increasing nonnegative numbers and the maximum `r` value must exceed the radius of the largest disc contained in the window.

The pair correlation function can also be applied to the result of `Kdot`; see `pcf`.

Value

An object of class "fv" (see [fv.object](#)).

Essentially a data frame containing numeric columns

r	the values of the argument r at which the function $K_{i\bullet}(r)$ has been estimated
theo	the theoretical value of $K_{i\bullet}(r)$ for a marked Poisson process, namely πr^2

together with a column or columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $K_{i\bullet}(r)$ obtained by the edge corrections named.

If `ratio=TRUE` then the return value also has two attributes called "numerator" and "denominator" which are "fv" objects containing the numerators and denominators of each estimate of $K(r)$.

Warnings

The argument `i` is interpreted as a level of the factor `X$marks`. It is converted to a character string if it is not already a character string. The value `i=1` does **not** refer to the first level of the factor.

The reduced sample estimator of $K_{i\bullet}$ is pointwise approximately unbiased, but need not be a valid distribution function; it may not be a nondecreasing function of r . Its range is always within $[0, 1]$.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.
- Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.
- Harkness, R.D and Isham, V. (1983) A bivariate spatial point pattern of ants' nests. *Applied Statistics* **32**, 293–303
- Lotwick, H. W. and Silverman, B. W. (1982). Methods for analysing spatial processes of several types of points. *J. Royal Statist. Soc. Ser. B* **44**, 406–413.
- Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.
- Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.

See Also

[Kdot](#), [Kest](#), [Kmulti](#), [pcf](#)

Examples

```
# Lansing woods data: 6 types of trees
data(lansing)

## Not run:
Kh. <- Kdot(lansing, "hickory")

## End(Not run)

# diagnostic plot for independence between hickories and other trees
```

```

plot(Kh.)

## Not run:
# synthetic example with two marks "a" and "b"
pp <- runifpoispp(50)
pp <- pp %mark% factor(sample(c("a","b"), npoints(pp), replace=TRUE))
K <- Kdot(pp, "a")

## End(Not run)

```

Kdot.inhom*Inhomogeneous Multitype K Dot Function***Description**

For a multitype point pattern, estimate the inhomogeneous version of the dot K function, which counts the expected number of points of any type within a given distance of a point of type i , adjusted for spatially varying intensity.

Usage

```
Kdot.inhom(X, i, lambdaI=NULL, lambdadot=NULL, ..., r=NULL, breaks=NULL,
           correction = c("border", "isotropic", "Ripley", "translate"),
           sigma=NULL, varcov=NULL, lambdaIdot=NULL)
```

Arguments

<i>X</i>	The observed point pattern, from which an estimate of the inhomogeneous cross type K function $K_{i\bullet}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
<i>i</i>	The type (mark value) of the points in <i>X</i> from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of <code>marks(X)</code> .
<i>lambdaI</i>	Optional. Values of the estimated intensity of the sub-process of points of type <i>i</i> . Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the type <i>i</i> points in <i>X</i> , or a function(<i>x,y</i>) which can be evaluated to give the intensity value at any location.
<i>lambdadot</i>	Optional. Values of the estimated intensity of the entire point process, Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the points in <i>X</i> , or a function(<i>x,y</i>) which can be evaluated to give the intensity value at any location.
<i>...</i>	Ignored.
<i>r</i>	Optional. Numeric vector giving the values of the argument <i>r</i> at which the cross K function $K_{ij}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on <i>r</i> .
<i>breaks</i>	Optional. An alternative to the argument <i>r</i> . Not normally invoked by the user. See the Details section.

correction	A character vector containing any selection of the options "border", "bord.modif", "isotropic", "Ripley", "translate", "none" or "best". It specifies the edge correction(s) to be applied.
sigma	Standard deviation of isotropic Gaussian smoothing kernel, used in computing leave-one-out kernel estimates of <code>lambdaI</code> , <code>lambdadot</code> if they are omitted.
varcov	Variance-covariance matrix of anisotropic Gaussian kernel, used in computing leave-one-out kernel estimates of <code>lambdaI</code> , <code>lambdadot</code> if they are omitted. Incompatible with <code>sigma</code> .
<code>lambdaIdot</code>	Optional. A matrix containing estimates of the product of the intensities <code>lambdaI</code> and <code>lambdadot</code> for each pair of points, the first point of type <i>i</i> and the second of any type.

Details

This is a generalisation of the function `Kdot` to include an adjustment for spatially inhomogeneous intensity, in a manner similar to the function `Kinhom`.

Briefly, given a multitype point process, consider the points without their types, and suppose this unmarked point process has intensity function $\lambda(u)$ at spatial locations u . Suppose we place a mass of $1/\lambda(\zeta)$ at each point ζ of the process. Then the expected total mass per unit area is 1. The inhomogeneous “dot-type” K function $K_{i\bullet}^{\text{inhom}}(r)$ equals the expected total mass within a radius r of a point of the process of type *i*, discounting this point itself.

If the process of type *i* points were independent of the points of other types, then $K_{i\bullet}^{\text{inhom}}(r)$ would equal πr^2 . Deviations between the empirical $K_{i\bullet}$ curve and the theoretical curve πr^2 suggest dependence between the points of types *i* and *j* for $j \neq i$.

The argument `X` must be a point pattern (object of class "ppp") or any data that are acceptable to `as.ppp`. It must be a marked point pattern, and the mark vector `X$marks` must be a factor.

The argument `i` will be interpreted as a level of the factor `X$marks`. (Warning: this means that an integer value `i=3` will be interpreted as the number 3, **not** the 3rd smallest level). If `i` is missing, it defaults to the first level of the marks factor, `i = levels(X$marks)[1]`.

The argument `lambdaI` supplies the values of the intensity of the sub-process of points of type *i*. It may be either

a **pixel image** (object of class "im") which gives the values of the type *i* intensity at all locations in the window containing `X`;

a **numeric vector** containing the values of the type *i* intensity evaluated only at the data points of type *i*. The length of this vector must equal the number of type *i* points in `X`.

a **function** of the form `function(x,y)` which can be evaluated to give values of the intensity at any locations.

omitted: if `lambdaI` is omitted then it will be estimated using a leave-one-out kernel smoother.

If `lambdaI` is omitted, then it will be estimated using a ‘leave-one-out’ kernel smoother, as described in Baddeley, Moller and Waagepetersen (2000). The estimate of `lambdaI` for a given point is computed by removing the point from the point pattern, applying kernel smoothing to the remaining points using `density.ppp`, and evaluating the smoothed intensity at the point in question. The smoothing kernel bandwidth is controlled by the arguments `sigma` and `varcov`, which are passed to `density.ppp` along with any extra arguments.

Similarly the argument `lambdadot` should contain estimated values of the intensity of the entire point process. It may be either a pixel image, a numeric vector of length equal to the number of points in `X`, a function, or omitted.

For advanced use only, the optional argument `lambdaIdot` is a matrix containing estimated values of the products of these two intensities for each pair of points, the first point of type `i` and the second of any type.

The argument `r` is the vector of values for the distance r at which $K_{i\bullet}(r)$ should be evaluated. The values of r must be increasing nonnegative numbers and the maximum r value must exceed the radius of the largest disc contained in the window.

The argument `correction` chooses the edge correction as explained e.g. in [Kest](#).

The pair correlation function can also be applied to the result of [Kcross.inhom](#); see [pcf](#).

Value

An object of class "fv" (see [fv.object](#)).

Essentially a data frame containing numeric columns

<code>r</code>	the values of the argument r at which the function $K_{i\bullet}(r)$ has been estimated
<code>theo</code>	the theoretical value of $K_{i\bullet}(r)$ for a marked Poisson process, namely πr^2

together with a column or columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $K_{i\bullet}(r)$ obtained by the edge corrections named.

Warnings

The argument `i` is interpreted as a level of the factor `X$marks`. It is converted to a character string if it is not already a character string. The value `i=1` does **not** refer to the first level of the factor.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

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

See Also

[Kdot](#), [Kinhom](#), [Kcross.inhom](#), [Kmulti.inhom](#), [pcf](#)

Examples

```
# Lansing Woods data
data(lansing)
lansing <- lansing[seq(1,lansing$n, by=10)]
ma <- split(lansing)$maple
lg <- unmark(lansing)

# Estimate intensities by nonparametric smoothing
lambdaM <- density.ppp(ma, sigma=0.15, at="points")
lambdaIdot <- density.ppp(lg, sigma=0.15, at="points")
K <- Kdot.inhom(lansing, "maple", lambdaI=lambdaM,
                 lambdaIdot=lambdaIdot)
```

```

# Equivalent
K <- Kdot.inhom(lansing, "maple", sigma=0.15)

# synthetic example: type A points have intensity 50,
#                      type B points have intensity 50 + 100 * x
lamB <- as.im(function(x,y){50 + 100 * x}, owin())
lamdot <- as.im(function(x,y) { 100 + 100 * x}, owin())
X <- superimpose(A=runifpoispp(50), B=rpoispp(lamB))
K <- Kdot.inhom(X, "B", lambdaI=lamB, lambdadot=lamdot)

```

Kest*K-function***Description**

Estimates Ripley's reduced second moment function $K(r)$ from a point pattern in a window of arbitrary shape.

Usage

```
Kest(X, ..., r=NULL, breaks=NULL,
      correction=c("border", "isotropic", "Ripley", "translate"),
      nlarge=3000, domain=NULL, var.approx=FALSE, ratio=FALSE)
```

Arguments

X	The observed point pattern, from which an estimate of $K(r)$ will be computed. An object of class "ppp", or data in any format acceptable to as.ppp() .
...	Ignored.
r	Optional. Vector of values for the argument r at which $K(r)$ should be evaluated. Users are advised <i>not</i> to specify this argument; there is a sensible default.
breaks	Optional. An alternative to the argument r . Not normally invoked by the user. See the Details section.
correction	Optional. A character vector containing any selection of the options "none", "border", "bord.modif", "isotropic", "Ripley", "translate", "none" or "best". It specifies the edge correction(s) to be applied.
nlarge	Optional. Efficiency threshold. If the number of points exceeds nlarge , then only the border correction will be computed (by default), using a fast algorithm.
domain	Optional. Calculations will be restricted to this subset of the window. See Details.
var.approx	Logical. If TRUE, the approximate variance of $\hat{K}(r)$ under CSR will also be computed.
ratio	Logical. If TRUE, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.

Details

The K function (variously called “Ripley’s K -function” and the “reduced second moment function”) of a stationary point process X is defined so that $\lambda K(r)$ equals the expected number of additional random points within a distance r of a typical random point of X . Here λ is the intensity of the process, i.e. the expected number of points of X per unit area. The K function is determined by the second order moment properties of X .

An estimate of K derived from a spatial point pattern dataset can be used in exploratory data analysis and formal inference about the pattern (Cressie, 1991; Diggle, 1983; Ripley, 1977, 1988). In exploratory analyses, the estimate of K is a useful statistic summarising aspects of inter-point “dependence” and “clustering”. For inferential purposes, the estimate of K is usually compared to the true value of K for a completely random (Poisson) point process, which is $K(r) = \pi r^2$. Deviations between the empirical and theoretical K curves may suggest spatial clustering or spatial regularity.

This routine `Kest` estimates the K function of a stationary point process, given observation of the process inside a known, bounded window. The argument X is interpreted as a point pattern object (of class “`ppp`”, see [ppp.object](#)) and can be supplied in any of the formats recognised by [as.ppp\(\)](#).

The estimation of K is hampered by edge effects arising from the unobservability of points of the random pattern outside the window. An edge correction is needed to reduce bias (Baddeley, 1998; Ripley, 1988). The corrections implemented here are

border the border method or “reduced sample” estimator (see Ripley, 1988). This is the least efficient (statistically) and the fastest to compute. It can be computed for a window of arbitrary shape.

isotropic/Ripley Ripley’s isotropic correction (see Ripley, 1988; Ohser, 1983). This is implemented for rectangular and polygonal windows (not for binary masks).

translate Translation correction (Ohser, 1983). Implemented for all window geometries, but slow for complex windows.

best Selects the best edge correction that is available for the geometry of the window. Currently this is Ripley’s isotropic correction for a rectangular or polygonal window, and the translation correction for masks.

none Uncorrected estimate. An estimate of the K function *without* edge correction. (i.e. setting $e_{ij} = 1$ in the equation below. This estimate is **biased** and should not be used for data analysis, *unless* you have an extremely large point pattern (more than 100,000 points)).

The estimates of $K(r)$ are of the form

$$\hat{K}(r) = \frac{a}{n(n-1)} \sum_i \sum_j I(d_{ij} \leq r) e_{ij}$$

where a is the area of the window, n is the number of data points, and the sum is taken over all ordered pairs of points i and j in X . Here d_{ij} is the distance between the two points, and $I(d_{ij} \leq r)$ is the indicator that equals 1 if the distance is less than or equal to r . The term e_{ij} is the edge correction weight (which depends on the choice of edge correction listed above).

Note that this estimator assumes the process is stationary (spatially homogeneous). For inhomogeneous point patterns, see [Kinhom](#).

If the point pattern X contains more than about 3000 points, the isotropic and translation edge corrections can be computationally prohibitive. The computations for the border method are much faster, and are statistically efficient when there are large numbers of points. Accordingly, if the number of points in X exceeds the threshold `nlarge`, then only the border correction will be computed. Setting `nlarge=Inf` or `correction="best"` will prevent this from happening. Setting `nlarge=0` is equivalent to selecting only the border correction with `correction="border"`.

If X contains more than about 100,000 points, even the border correction is time-consuming. You may want to consider setting `correction="none"` in this case. There is an even faster algorithm for the uncorrected estimate.

Approximations to the variance of $\hat{K}(r)$ are available, for the case of the isotropic edge correction estimator, **assuming complete spatial randomness** (Ripley, 1988; Lotwick and Silverman, 1982; Diggle, 2003, pp 51-53). If `var.approx=TRUE`, then the result of `Kest` also has a column named `rip` values of Ripley's (1988) approximation to $\text{var}(\hat{K}(r))$, and (if the window is a rectangle) a column named `ls` giving values of Lotwick and Silverman's (1982) approximation.

If the argument `domain` is given, the calculations will be restricted to a subset of the data. In the formula for $K(r)$ above, the *first* point i will be restricted to lie inside `domain`. The result is an approximately unbiased estimate of $K(r)$ based on pairs of points in which the first point lies inside `domain` and the second point is unrestricted. This is useful in bootstrap techniques. The argument `domain` should be a window (object of class "owin") or something acceptable to `as.owin`. It must be a subset of the window of the point pattern X .

The estimator `Kest` ignores marks. Its counterparts for multitype point patterns are `Kcross`, `Kdot`, and for general marked point patterns see `Kmulti`.

Some writers, particularly Stoyan (1994, 1995) advocate the use of the “pair correlation function”

$$g(r) = \frac{K'(r)}{2\pi r}$$

where $K'(r)$ is the derivative of $K(r)$. See `pcf` on how to estimate this function.

Value

An object of class "fv", see `fv.object`, which can be plotted directly using `plot.fv`.

Essentially a data frame containing columns

<code>r</code>	the vector of values of the argument r at which the function K has been estimated
<code>theo</code>	the theoretical value $K(r) = \pi r^2$ for a stationary Poisson process

together with columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $K(r)$ obtained by the edge corrections named.

If `var.approx=TRUE` then the return value also has columns `rip` and `ls` containing approximations to the variance of $\hat{K}(r)$ under CSR.

If `ratio=TRUE` then the return value also has two attributes called "numerator" and "denominator" which are "fv" objects containing the numerators and denominators of each estimate of $K(r)$.

Envelopes, significance bands and confidence intervals

To compute simulation envelopes for the K -function under CSR, use `envelope`.

To compute a confidence interval for the true K -function, use `varblock` or `lohboot`.

Warnings

The estimator of $K(r)$ is approximately unbiased for each fixed r . Bias increases with r and depends on the window geometry. For a rectangular window it is prudent to restrict the r values to a maximum of 1/4 of the smaller side length of the rectangle. Bias may become appreciable for point patterns consisting of fewer than 15 points.

While $K(r)$ is always a non-decreasing function, the estimator of K is not guaranteed to be non-decreasing. This is rarely a problem in practice.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

References

- Baddeley, A.J. Spatial sampling and censoring. In O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*. Chapman and Hall, 1998. Chapter 2, pages 37–78.
- Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.
- Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.
- Ohser, J. (1983) On estimators for the reduced second moment measure of point processes. *Mathematische Operationsforschung und Statistik, series Statistics*, **14**, 63 – 71.
- Ripley, B.D. (1977) Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, Series B*, **39**, 172 – 212.
- Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.
- Stoyan, D, Kendall, W.S. and Mecke, J. (1995) *Stochastic geometry and its applications*. 2nd edition. Springer Verlag.
- Stoyan, D. and Stoyan, H. (1994) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

See Also

- [localK](#) to extract individual summands in the K function.
- [pcf](#) for the pair correlation.
- [Fest](#), [Gest](#), [Jest](#) for alternative summary functions.
- [Kcross](#), [Kdot](#), [Kinhom](#), [Kmulti](#) for counterparts of the K function for multitype point patterns.
- [reduced.sample](#) for the calculation of reduced sample estimators.

Examples

```
pp <- runifpoint(50)
K <- Kest(pp)
data(cells)
K <- Kest(cells, correction="isotropic")
plot(K)
plot(K, main="K function for cells")
# plot the L function
plot(K, sqrt(iso/pi) ~ r)
plot(K, sqrt(. / pi) ~ r, ylab="L(r)", main="L function for cells")
```

Kest.fft	<i>K-function using FFT</i>
-----------------	-----------------------------

Description

Estimates the reduced second moment function $K(r)$ from a point pattern in a window of arbitrary shape, using the Fast Fourier Transform.

Usage

```
Kest.fft(X, sigma, r=NULL, breaks=NULL)
```

Arguments

X	The observed point pattern, from which an estimate of $K(r)$ will be computed. An object of class "ppp", or data in any format acceptable to as.ppp() .
sigma	standard deviation of the isotropic Gaussian smoothing kernel.
r	vector of values for the argument r at which $K(r)$ should be evaluated. There is a sensible default.
breaks	An alternative to the argument r . Not normally invoked by the user. See Details.

Details

This is an alternative to the function [Kest](#) for estimating the K function. It may be useful for very large patterns of points.

Whereas [Kest](#) computes the distance between each pair of points analytically, this function discretises the point pattern onto a rectangular pixel raster and applies Fast Fourier Transform techniques to estimate $K(t)$. The hard work is done by the function [Kmeasure](#).

The result is an approximation whose accuracy depends on the resolution of the pixel raster. The resolution is controlled by setting the parameter **npixel** in [spatstat.options](#).

Value

An object of class "fv" (see [fv.object](#)).

Essentially a data frame containing columns

r	the vector of values of the argument r at which the function K has been estimated
border	the estimates of $K(r)$ for these values of r
theo	the theoretical value $K(r) = \pi r^2$ for a stationary Poisson process

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.
- Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.
- Ohser, J. (1983) On estimators for the reduced second moment measure of point processes. *Mathematische Operationsforschung und Statistik, series Statistics*, **14**, 63 – 71.
- Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.
- Stoyan, D, Kendall, W.S. and Mecke, J. (1995) *Stochastic geometry and its applications*. 2nd edition. Springer Verlag.
- Stoyan, D. and Stoyan, H. (1994) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

See Also

[Kest](#), [Kmeasure](#), [spatstat.options](#)

Examples

```
pp <- runifpoint(10000)
## Not run:
spatstat.options(npixel=512)

## End(Not run)

Kpp <- Kest.fft(pp, 0.01)
plot(Kpp)
```

Description

Estimates the inhomogeneous K function of a non-stationary point pattern.

Usage

```
Kinhom(X, lambda=NULL, ..., r = NULL, breaks = NULL,
  correction=c("border", "bord.modif", "isotropic", "translate"),
  renormalise=TRUE,
  normpower=1,
  nlarge = 1000,
  lambda2=NULL, reciplambda=NULL, reciplambda2=NULL,
  sigma=NULL, varcov=NULL)
```

Arguments

X	The observed data point pattern, from which an estimate of the inhomogeneous K function will be computed. An object of class "ppp" or in a format recognised by as.ppp()
lambda	Optional. Values of the estimated intensity function. Either a vector giving the intensity values at the points of the pattern X, a pixel image (object of class "im") giving the intensity values at all locations, a fitted point process model (object of class "ppm") or a function(x,y) which can be evaluated to give the intensity value at any location.
...	Extra arguments. Ignored if lambda is present. Passed to density.ppp if lambda is omitted.
r	vector of values for the argument r at which the inhomogeneous K function should be evaluated. Not normally given by the user; there is a sensible default.
breaks	An alternative to the argument r. Not normally invoked by the user. See Details.
correction	A character vector containing any selection of the options "border", "bord.modif", "isotropic", "Ripley", "translate", "none" or "best". It specifies the edge correction(s) to be applied.
renormalise	Logical. Whether to renormalise the estimate. See Details.
normpower	Integer (usually either 1 or 2). Normalisation power. See Details.
nlarge	Optional. Efficiency threshold. If the number of points exceeds nlarge, then only the border correction will be computed, using a fast algorithm.
lambda2	Advanced use only. Matrix containing estimates of the products $\lambda(x_i)\lambda(x_j)$ of the intensities at each pair of data points x_i and x_j .
reciplambda	Alternative to lambda. Values of the estimated reciprocal $1/\lambda$ of the intensity function. Either a vector giving the reciprocal intensity values at the points of the pattern X, a pixel image (object of class "im") giving the reciprocal intensity values at all locations, or a function(x,y) which can be evaluated to give the reciprocal intensity value at any location.
reciplambda2	Advanced use only. Alternative to lambda2. A matrix giving values of the estimated reciprocal products $1/\lambda(x_i)\lambda(x_j)$ of the intensities at each pair of data points x_i and x_j .
sigma, varcov	Optional arguments passed to density.ppp to control the smoothing bandwidth, when lambda is estimated by kernel smoothing.

Details

This computes a generalisation of the K function for inhomogeneous point patterns, proposed by Baddeley, Moller and Waagepetersen (2000).

The “ordinary” K function (variously known as the reduced second order moment function and Ripley’s K function), is described under [Kest](#). It is defined only for stationary point processes.

The inhomogeneous K function $K_{\text{inhom}}(r)$ is a direct generalisation to nonstationary point processes. Suppose x is a point process with non-constant intensity $\lambda(u)$ at each location u . Define $K_{\text{inhom}}(r)$ to be the expected value, given that u is a point of x , of the sum of all terms $1/\lambda(x_j)$ over all points x_j in the process separated from u by a distance less than r . This reduces to the ordinary K function if $\lambda()$ is constant. If x is an inhomogeneous Poisson process with intensity function $\lambda(u)$, then $K_{\text{inhom}}(r) = \pi r^2$.

Given a point pattern dataset, the inhomogeneous K function can be estimated essentially by summing the values $1/(\lambda(x_i)\lambda(x_j))$ for all pairs of points x_i, x_j separated by a distance less than r .

This allows us to inspect a point pattern for evidence of interpoint interactions after allowing for spatial inhomogeneity of the pattern. Values $K_{\text{inhom}}(r) > \pi r^2$ are suggestive of clustering.

The argument `lambda` should supply the (estimated) values of the intensity function λ . It may be either

- a numeric vector** containing the values of the intensity function at the points of the pattern `X`.
- a pixel image** (object of class "im") assumed to contain the values of the intensity function at all locations in the window.
- a fitted point process model** (object of class "ppm") whose fitted `trend` can be used as the fitted intensity.
- a function** which can be evaluated to give values of the intensity at any locations.
- omitted:** if `lambda` is omitted, then it will be estimated using a 'leave-one-out' kernel smoother.

If `lambda` is a numeric vector, then its length should be equal to the number of points in the pattern `X`. The value `lambda[i]` is assumed to be the (estimated) value of the intensity $\lambda(x_i)$ for the point x_i of the pattern X . Each value must be a positive number; NA's are not allowed.

If `lambda` is a pixel image, the domain of the image should cover the entire window of the point pattern. If it does not (which may occur near the boundary because of discretisation error), then the missing pixel values will be obtained by applying a Gaussian blur to `lambda` using `blur`, then looking up the values of this blurred image for the missing locations. (A warning will be issued in this case.)

If `lambda` is a function, then it will be evaluated in the form `lambda(x, y)` where `x` and `y` are vectors of coordinates of the points of `X`. It should return a numeric vector with length equal to the number of points in `X`.

If `lambda` is omitted, then it will be estimated using a 'leave-one-out' kernel smoother, as described in Baddeley, Moller and Waagepetersen (2000). The estimate `lambda[i]` for the point `X[i]` is computed by removing `X[i]` from the point pattern, applying kernel smoothing to the remaining points using `density.ppp`, and evaluating the smoothed intensity at the point `X[i]`. The smoothing kernel bandwidth is controlled by the arguments `sigma` and `varcov`, which are passed to `density.ppp` along with any extra arguments.

Edge corrections are used to correct bias in the estimation of K_{inhom} . Each edge-corrected estimate of $K_{\text{inhom}}(r)$ is of the form

$$\widehat{K}_{\text{inhom}}(r) = \sum_i \sum_j \frac{1\{d_{ij} \leq r\}e(x_i, x_j, r)}{\lambda(x_i)\lambda(x_j)}$$

where d_{ij} is the distance between points x_i and x_j , and $e(x_i, x_j, r)$ is an edge correction factor. For the 'border' correction,

$$e(x_i, x_j, r) = \frac{1(b_i > r)}{\sum_j 1(b_j > r)/\lambda(x_j)}$$

where b_i is the distance from x_i to the boundary of the window. For the 'modified border' correction,

$$e(x_i, x_j, r) = \frac{1(b_i > r)}{\text{area}(W \ominus r)}$$

where $W \ominus r$ is the eroded window obtained by trimming a margin of width r from the border of the original window. For the 'translation' correction,

$$e(x_i, x_j, r) = \frac{1}{\text{area}(W \cap (W + (x_j - x_i)))}$$

and for the ‘isotropic’ correction,

$$e(x_i, x_j, r) = \frac{1}{\text{area}(W)g(x_i, x_j)}$$

where $g(x_i, x_j)$ is the fraction of the circumference of the circle with centre x_i and radius $\|x_i - x_j\|$ which lies inside the window.

If `renormalise=TRUE` (the default), then the estimates are multiplied by $c^{\text{normpower}}$ where $c = \text{area}(W)/\sum(1/\lambda(x_i))$. This rescaling reduces the variability and bias of the estimate in small samples and in cases of very strong inhomogeneity. The default value of `normpower` is 1 (for consistency with previous versions of `spatstat`) but the most sensible value is 2, which would correspond to rescaling the `lambda` values so that $\sum(1/\lambda(x_i)) = \text{area}(W)$.

If the point pattern X contains more than about 1000 points, the isotropic and translation edge corrections can be computationally prohibitive. The computations for the border method are much faster, and are statistically efficient when there are large numbers of points. Accordingly, if the number of points in X exceeds the threshold `nlarge`, then only the border correction will be computed. Setting `nlarge=Inf` or `correction="best"` will prevent this from happening. Setting `nlarge=0` is equivalent to selecting only the border correction with `correction="border"`.

The pair correlation function can also be applied to the result of `Kininhom`; see [pcf](#).

Value

An object of class “fv” (see [fv.object](#)).

Essentially a data frame containing at least the following columns,

<code>r</code>	the vector of values of the argument r at which $K_{\text{inhom}}(r)$ has been estimated
<code>theo</code>	vector of values of πr^2 , the theoretical value of $K_{\text{inhom}}(r)$ for an inhomogeneous Poisson process

and containing additional columns according to the choice specified in the `correction` argument. The additional columns are named `border`, `trans` and `iso` and give the estimated values of $K_{\text{inhom}}(r)$ using the border correction, translation correction, and Ripley isotropic correction, respectively.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A., Moller, J. and Waagepetersen, R. (2000) Non- and semiparametric estimation of interaction in inhomogeneous point patterns. *Statistica Neerlandica* **54**, 329–350.

See Also

[Kest](#), [pcf](#)

Examples

```

data(lansing)
# inhomogeneous pattern of maples
X <- unmark(split(lansing)$maple)

# (1) intensity function estimated by model-fitting
# Fit spatial trend: polynomial in x and y coordinates
fit <- ppm(X, ~ polynom(x,y,2), Poisson())
# (a) predict intensity values at points themselves,
#      obtaining a vector of lambda values
lambda <- predict(fit, locations=X, type="trend")
# inhomogeneous K function
Ki <- Kinhom(X, lambda)
plot(Ki)
# (b) predict intensity at all locations,
#      obtaining a pixel image
lambda <- predict(fit, type="trend")
Ki <- Kinhom(X, lambda)
plot(Ki)

# (2) intensity function estimated by heavy smoothing
Ki <- Kinhom(X, sigma=0.1)
plot(Ki)

# (3) simulated data: known intensity function
lamfun <- function(x,y) { 50 + 100 * x }
# inhomogeneous Poisson process
Y <- rpoispp(lamfun, 150, owin())
# inhomogeneous K function
Ki <- Kinhom(Y, lamfun)
plot(Ki)

# How to make simulation envelopes:
# Example shows method (2)
## Not run:
smo <- density.ppp(X, sigma=0.1)
Ken <- envelope(X, Kinhom, nsim=99,
                 simulate=expression(rpoispp(smo)),
                 sigma=0.1, correction="trans")
plot(Ken)

## End(Not run)

```

Description

Compute the Kaplan-Meier and Reduced Sample estimators of a survival time distribution function, using histogram techniques

Usage

```
km.rs(o, cc, d, breaks)
```

Arguments

<code>o</code>	vector of observed survival times
<code>cc</code>	vector of censoring times
<code>d</code>	vector of non-censoring indicators
<code>breaks</code>	Vector of breakpoints to be used to form histograms.

Details

This function is needed mainly for internal use in **spatstat**, but may be useful in other applications where you want to form the Kaplan-Meier estimator from a huge dataset.

Suppose T_i are the survival times of individuals $i = 1, \dots, M$ with unknown distribution function $F(t)$ which we wish to estimate. Suppose these times are right-censored by random censoring times C_i . Thus the observations consist of right-censored survival times $\tilde{T}_i = \min(T_i, C_i)$ and non-censoring indicators $D_i = 1\{T_i \leq C_i\}$ for each i .

The arguments to this function are vectors `o`, `cc`, `d` of observed values of \tilde{T}_i , C_i and D_i respectively. The function computes histograms and forms the reduced-sample and Kaplan-Meier estimates of $F(t)$ by invoking the functions `kaplan.meier` and `reduced.sample`. This is efficient if the lengths of `o`, `cc`, `d` (i.e. the number of observations) is large.

The vectors `km` and `hazard` returned by `kaplan.meier` are (histogram approximations to) the Kaplan-Meier estimator of $F(t)$ and its hazard rate $\lambda(t)$. Specifically, `km[k]` is an estimate of $F(\text{breaks}[k+1])$, and `lambda[k]` is an estimate of the average of $\lambda(t)$ over the interval $(\text{breaks}[k], \text{breaks}[k+1])$. This approximation is exact only if the survival times are discrete and the histogram breaks are fine enough to ensure that each interval $(\text{breaks}[k], \text{breaks}[k+1])$ contains only one possible value of the survival time.

The vector `rs` is the reduced-sample estimator, `rs[k]` being the reduced sample estimate of $F(\text{breaks}[k+1])$. This value is exact, i.e. the use of histograms does not introduce any approximation error in the reduced-sample estimator.

Value

A list with five elements

<code>rs</code>	Reduced-sample estimate of the survival time c.d.f. $F(t)$
<code>km</code>	Kaplan-Meier estimate of the survival time c.d.f. $F(t)$
<code>hazard</code>	corresponding Nelson-Aalen estimate of the hazard rate $\lambda(t)$
<code>r</code>	values of t for which $F(t)$ is estimated
<code>breaks</code>	the breakpoints vector

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`reduced.sample`, `kaplan.meier`

Kmeasure	<i>Reduced Second Moment Measure</i>
----------	--------------------------------------

Description

Estimates the reduced second moment measure κ from a point pattern in a window of arbitrary shape.

Usage

```
Kmeasure(X, sigma, edge=TRUE, ..., varcov=NULL)
```

Arguments

X	The observed point pattern, from which an estimate of κ will be computed. An object of class "ppp", or data in any format acceptable to as.ppp() .
sigma	Standard deviation σ of the Gaussian smoothing kernel. Incompatible with varcov.
edge	logical value indicating whether an edge correction should be applied.
...	Ignored.
varcov	Variance-covariance matrix of the Gaussian smoothing kernel. Incompatible with sigma.

Details

Given a point pattern dataset, this command computes an estimate of the reduced second moment measure κ of the point process. The result is a pixel image whose pixel values are estimates of the density of the reduced second moment measure.

The reduced second moment measure κ can be regarded as a generalisation of the more familiar K -function. An estimate of κ derived from a spatial point pattern dataset can be useful in exploratory data analysis. Its advantage over the K -function is that it is also sensitive to anisotropy and directional effects.

In a nutshell, the command Kmeasure computes a smoothed version of the *Fry plot*. As explained under [fryplot](#), the Fry plot is a scatterplot of the vectors joining all pairs of points in the pattern. The reduced second moment measure is (essentially) defined as the average of the Fry plot over different realisations of the point process. The command Kmeasure effectively smooths the Fry plot of a dataset to obtain an estimate of the reduced second moment measure.

In formal terms, the reduced second moment measure κ of a stationary point process X is a measure defined on the two-dimensional plane such that, for a ‘typical’ point x of the process, the expected number of other points y of the process such that the vector $y - x$ lies in a region A , equals $\lambda\kappa(A)$. Here λ is the intensity of the process, i.e. the expected number of points of X per unit area.

The K -function is a special case. The function value $K(t)$ is the value of the reduced second moment measure for the disc of radius t centred at the origin; that is, $K(t) = \kappa(b(0, t))$.

The command Kmeasure computes an estimate of κ from a point pattern dataset X, which is assumed to be a realisation of a stationary point process, observed inside a known, bounded window. Marks are ignored.

The algorithm approximates the point pattern and its window by binary pixel images, introduces a Gaussian smoothing kernel and uses the Fast Fourier Transform [fft](#) to form a density estimate of κ . The calculation corresponds to the edge correction known as the “translation correction”.

The Gaussian smoothing kernel may be specified by either of the arguments `sigma` or `varcov`. If `sigma` is a single number, this specifies an isotropic Gaussian kernel with standard deviation `sigma` on each coordinate axis. If `sigma` is a vector of two numbers, this specifies a Gaussian kernel with standard deviation `sigma[1]` on the x axis, standard deviation `sigma[2]` on the y axis, and zero correlation between the x and y axes. If `varcov` is given, this specifies the variance-covariance matrix of the Gaussian kernel. There do not seem to be any well-established rules for selecting the smoothing kernel in this context.

The density estimate of κ is returned in the form of a real-valued pixel image. Pixel values are estimates of the normalised second moment density at the centre of the pixel. (The uniform Poisson process would have values identically equal to 1.) The image x and y coordinates are on the same scale as vector displacements in the original point pattern window. The point $x=0$, $y=0$ corresponds to the ‘typical point’. A peak in the image near $(0,0)$ suggests clustering; a dip in the image near $(0,0)$ suggests inhibition; peaks or dips at other positions suggest possible periodicity.

If desired, the value of $\kappa(A)$ for a region A can be estimated by computing the integral of the pixel image over the domain A , i.e.\ summing the pixel values and multiplying by pixel area, using `integral.im`. One possible application is to compute anisotropic counterparts of the K -function (in which the disc of radius t is replaced by another shape). See Examples.

Value

A real-valued pixel image (an object of class “`im`”, see `im.object`) whose pixel values are estimates of the density of the reduced second moment measure at each location.

Warning

Some writers use the term *reduced second moment measure* when they mean the K -function. This has caused confusion.

As originally defined, the reduced second moment measure is a measure, obtained by modifying the second moment measure, while the K -function is a function obtained by evaluating this measure for discs of increasing radius. In `spatstat`, the K -function is computed by `Kest` and the reduced second moment measure is computed by `Kmeasure`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Stoyan, D, Kendall, W.S. and Mecke, J. (1995) *Stochastic geometry and its applications*. 2nd edition. Springer Verlag.
- Stoyan, D. and Stoyan, H. (1994) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

See Also

`Kest`, `fryplot`, `spatstat.options`, `integral.im`, `im.object`

Examples

```
data(cells)
plot(Kmeasure(cells, 0.05))
# shows pronounced dip around origin consistent with strong inhibition
```

```

data(redwood)
plot(Kmeasure(redwood, 0.03), col=grey(seq(1,0,length=32)))
# shows peaks at several places, reflecting clustering and ?periodicity
M <- Kmeasure(cells, 0.05)
# evaluate measure on a sector
W <- as.owin(M)
ang <- as.im(atan2, W)
rad <- as.im(function(x,y){sqrt(x^2+y^2)}, W)
sector <- solutionset(ang > 0 & ang < 1 & rad < 0.6)
integral.im(M[sector, drop=FALSE])

```

Kmodel*K function of a model***Description**

Returns the theoretical K function or the pair correlation function of a point process model.

Usage

```

Kmodel(model, ...)
pcfmodel(model, ...)
## S3 method for class 'kppm'
Kmodel(model, ...)
## S3 method for class 'kppm'
pcfmodel(model, ...)

```

Arguments

- | | |
|-------|--|
| model | A fitted cluster point process model, typically obtained from the model-fitting algorithm kppm . An object of class "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. Currently the only method is 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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[kppm](#), [Kest](#), [pcf](#)

Examples

```
data(redwood)
fit <- kppm(redwood, ~x, "MatClust")
K <- Kmodel(fit)
K(c(0.1, 0.2))
curve(K(x), from=0, to=0.25)
```

Kmulti

Marked K-Function

Description

For a marked point pattern, estimate the multitype K function which counts the expected number of points of subset J within a given distance from a typical point in subset I .

Usage

```
Kmulti(X, I, J, r=NULL, breaks=NULL, correction, ..., ratio=FALSE)
```

Arguments

X	The observed point pattern, from which an estimate of the multitype K function $K_{IJ}(r)$ will be computed. It must be a marked point pattern. See under Details.
I	Subset index specifying the points of X from which distances are measured. See Details.
J	Subset index specifying the points in X to which distances are measured. See Details.
r	numeric vector. The values of the argument r at which the multitype K function $K_{IJ}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r .
breaks	An alternative to the argument r . Not normally invoked by the user. See the Details section.
correction	A character vector containing any selection of the options "border", "bord.modif", "isotropic", "Ripley", "translate", "none" or "best". It specifies the edge correction(s) to be applied.
...	Ignored.
ratio	Logical. If TRUE, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.

Details

The function *Kmulti* generalises *Kest* (for unmarked point patterns) and *Kdot* and *Kcross* (for multitype point patterns) to arbitrary marked point patterns.

Suppose X_I, X_J are subsets, possibly overlapping, of a marked point process. The multitype K function is defined so that $\lambda_J K_{IJ}(r)$ equals the expected number of additional random points of X_J within a distance r of a typical point of X_I . Here λ_J is the intensity of X_J i.e. the expected number of points of X_J per unit area. The function K_{IJ} is determined by the second order moment properties of X .

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to *as.ppp*.

The arguments I and J specify two subsets of the point pattern. They may be any type of subset indices, for example, logical vectors of length equal to *npoints*(X), or integer vectors with entries in the range 1 to *npoints*(X), or negative integer vectors.

Alternatively, I and J may be **functions** that will be applied to the point pattern X to obtain index vectors. If I is a function, then evaluating $I(X)$ should yield a valid subset index. This option is useful when generating simulation envelopes using *envelope*.

The argument r is the vector of values for the distance r at which $K_{IJ}(r)$ should be evaluated. It is also used to determine the breakpoints (in the sense of *hist*) for the computation of histograms of distances.

First-time users would be strongly advised not to specify r . However, if it is specified, r must satisfy $r[1] = 0$, and $\max(r)$ must be larger than the radius of the largest disc contained in the window.

This algorithm assumes that X can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in X as $X>window$) may have arbitrary shape.

Biases due to edge effects are treated in the same manner as in *Kest*. The edge corrections implemented here are

border the border method or "reduced sample" estimator (see Ripley, 1988). This is the least efficient (statistically) and the fastest to compute. It can be computed for a window of arbitrary shape.

isotropic/Ripley Ripley's isotropic correction (see Ripley, 1988; Ohser, 1983). This is currently implemented only for rectangular windows.

translate Translation correction (Ohser, 1983). Implemented for all window geometries.

The pair correlation function *pcf* can also be applied to the result of *Kmulti*.

Value

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

Essentially a data frame containing numeric columns

r	the values of the argument r at which the function $K_{IJ}(r)$ has been estimated
theo	the theoretical value of $K_{IJ}(r)$ for a marked Poisson process, namely πr^2

together with a column or columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $K_{IJ}(r)$ obtained by the edge corrections named.

If *ratio*=TRUE then the return value also has two attributes called "numerator" and "denominator" which are "fv" objects containing the numerators and denominators of each estimate of $K(r)$.

Warnings

The function K_{IJ} is not necessarily differentiable.

The border correction (reduced sample) estimator of K_{IJ} used here is pointwise approximately unbiased, but need not be a nondecreasing function of r , while the true K_{IJ} must be nondecreasing.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Cressie, N.A.C. *Statistics for spatial data*. John Wiley and Sons, 1991.
- Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 1983.
- Diggle, P. J. (1986). Displaced amacrine cells in the retina of a rabbit : analysis of a bivariate spatial point pattern. *J. Neurosci. Meth.* **18**, 115–125.
- Harkness, R.D and Isham, V. (1983) A bivariate spatial point pattern of ants' nests. *Applied Statistics* **32**, 293–303
- Lotwick, H. W. and Silverman, B. W. (1982). Methods for analysing spatial processes of several types of points. *J. Royal Statist. Soc. Ser. B* **44**, 406–413.
- Ripley, B.D. *Statistical inference for spatial processes*. Cambridge University Press, 1988.
- Stoyan, D, Kendall, W.S. and Mecke, J. *Stochastic geometry and its applications*. 2nd edition. Springer Verlag, 1995.
- Van Lieshout, M.N.M. and Baddeley, A.J. (1999) Indices of dependence between types in multivariate point patterns. *Scandinavian Journal of Statistics* **26**, 511–532.

See Also

[Kcross](#), [Kdot](#), [Kest](#), [pcf](#)

Examples

```
# Longleaf Pine data: marks represent diameter

K <- Kmulti(longleaf, longleaf$marks <= 15, longleaf$marks >= 25)
plot(K)
# functions determining subsets
f1 <- function(X) { marks(X) <= 15 }
f2 <- function(X) { marks(X) >= 15 }
K <- Kmulti(longleaf, f1, f2)
```

Kmulti.inhom*Inhomogeneous Marked K-Function***Description**

For a marked point pattern, estimate the inhomogeneous version of the multitype K function which counts the expected number of points of subset J within a given distance from a typical point in subset I , adjusted for spatially varying intensity.

Usage

```
Kmulti.inhom(X, I, J, lambdaI=NULL, lambdaJ=NULL,
             ...,
             r=NULL, breaks=NULL,
             correction=c("border", "isotropic", "Ripley", "translate"),
             lambdaIJ=NULL,
             sigma=NULL, varcov=NULL)
```

Arguments

X	The observed point pattern, from which an estimate of the inhomogeneous multitype K function $K_{IJ}(r)$ will be computed. It must be a marked point pattern. See under Details.
I	Subset index specifying the points of X from which distances are measured. See Details.
J	Subset index specifying the points in X to which distances are measured. See Details.
lambdaI	Optional. Values of the estimated intensity of the sub-process $X[I]$. Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the points in $X[I]$, or a function(x,y) which can be evaluated to give the intensity value at any location.
lambdaJ	Optional. Values of the estimated intensity of the sub-process $X[J]$. Either a pixel image (object of class "im"), a numeric vector containing the intensity values at each of the points in $X[J]$, or a function(x,y) which can be evaluated to give the intensity value at any location.
...	Ignored.
r	Optional. Numeric vector. The values of the argument r at which the multitype K function $K_{IJ}(r)$ should be evaluated. There is a sensible default. First-time users are strongly advised not to specify this argument. See below for important conditions on r .
breaks	An alternative to the argument r. Not normally invoked by the user. See the Details section.
correction	A character vector containing any selection of the options "border", "bord.modif", "isotropic", "Ripley", "translate", "none" or "best". It specifies the edge correction(s) to be applied.
lambdaIJ	Optional. A matrix containing estimates of the product of the intensities lambdaI and lambdaJ for each pair of points, the first point
sigma, varcov	Optional arguments passed to density.ppp to control the smoothing bandwidth, when lambda is estimated by kernel smoothing.

Details

The function `Kmulti.inhom` is the counterpart, for spatially-inhomogeneous marked point patterns, of the multitype K function `Kmulti`.

Suppose X is a marked point process, with marks of any kind. Suppose X_I , X_J are two sub-processes, possibly overlapping. Typically X_I would consist of those points of X whose marks lie in a specified range of mark values, and similarly for X_J . Suppose that $\lambda_I(u)$, $\lambda_J(u)$ are the spatially-varying intensity functions of X_I and X_J respectively. Consider all the pairs of points (u, v) in the point process X such that the first point u belongs to X_I , the second point v belongs to X_J , and the distance between u and v is less than a specified distance r . Give this pair (u, v) the numerical weight $1/(\lambda_I(u)\lambda_J(v))$. Calculate the sum of these weights over all pairs of points as described. This sum (after appropriate edge-correction and normalisation) is the estimated inhomogeneous multitype K function.

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to `as.ppp`.

The arguments I and J specify two subsets of the point pattern. They may be any type of subset indices, for example, logical vectors of length equal to `npoints(X)`, or integer vectors with entries in the range 1 to `npoints(X)`, or negative integer vectors.

Alternatively, I and J may be **functions** that will be applied to the point pattern X to obtain index vectors. If I is a function, then evaluating $I(X)$ should yield a valid subset index. This option is useful when generating simulation envelopes using `envelope`.

The argument `lambdaI` supplies the values of the intensity of the sub-process identified by index I . It may be either

a pixel image (object of class "im") which gives the values of the intensity of $X[I]$ at all locations in the window containing X ;

a numeric vector containing the values of the intensity of $X[I]$ evaluated only at the data points of $X[I]$. The length of this vector must equal the number of points in $X[I]$.

a function of the form `function(x,y)` which can be evaluated to give values of the intensity at any locations.

omitted: if `lambdaI` is omitted then it will be estimated using a leave-one-out kernel smoother.

If `lambdaI` is omitted, then it will be estimated using a ‘leave-one-out’ kernel smoother, as described in Baddeley, Moller and Waagepetersen (2000). The estimate of `lambdaI` for a given point is computed by removing the point from the point pattern, applying kernel smoothing to the remaining points using `density.ppp`, and evaluating the smoothed intensity at the point in question. The smoothing kernel bandwidth is controlled by the arguments `sigma` and `varcov`, which are passed to `density.ppp` along with any extra arguments.

Similarly `lambdaJ` supplies the values of the intensity of the sub-process identified by index J .

The argument r is the vector of values for the distance r at which $K_{IJ}(r)$ should be evaluated. It is also used to determine the breakpoints (in the sense of `hist`) for the computation of histograms of distances.

First-time users would be strongly advised not to specify r . However, if it is specified, r must satisfy $r[1] = 0$, and `max(r)` must be larger than the radius of the largest disc contained in the window.

Biases due to edge effects are treated in the same manner as in `Kinhom`. The edge corrections implemented here are

border the border method or “reduced sample” estimator (see Ripley, 1988). This is the least efficient (statistically) and the fastest to compute. It can be computed for a window of arbitrary shape.

isotropic/Ripley Ripley's isotropic correction (see Ripley, 1988; Ohser, 1983). This is currently implemented only for rectangular windows.

translate Translation correction (Ohser, 1983). Implemented for all window geometries.

The pair correlation function **pcf** can also be applied to the result of *Kmulti.inhom*.

Value

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

Essentially a data frame containing numeric columns

r	the values of the argument r at which the function $K_{IJ}(r)$ has been estimated
theo	the theoretical value of $K_{IJ}(r)$ for a marked Poisson process, namely πr^2

together with a column or columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $K_{IJ}(r)$ obtained by the edge corrections named.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A., Moller, J. and Waagepetersen, R. (2000) Non- and semiparametric estimation of interaction in inhomogeneous point patterns. *Statistica Neerlandica* **54**, 329–350.

See Also

Kmulti, **Kdot.inhom**, **Kcross.inhom**, **pcf**

Examples

```
# Finnish Pines data: marked by diameter and height
plot(finpine, which.marks="height")
I <- (marks(finpine)$height <= 2)
J <- (marks(finpine)$height > 3)
K <- Kmulti.inhom(finpine, I, J)
plot(K)
# functions determining subsets
f1 <- function(X) { marks(X)$height <= 2 }
f2 <- function(X) { marks(X)$height > 3 }
K <- Kmulti.inhom(finpine, f1, f2)
```

kppm	<i>Fit Cluster or Cox Point Process Model</i>
------	---

Description

Fit a homogeneous or inhomogeneous cluster process or Cox point process model to a point pattern.

Usage

```
kppm(X, trend = ~1, clusters = "Thomas", covariates = NULL, ...,
      statistic="K", statargs=list())
```

Arguments

<code>X</code>	Point pattern (object of class "ppp") to which the model should be fitted.
<code>trend</code>	An R formula, with no left hand side, specifying the form of the log intensity.
<code>clusters</code>	Character string determining the cluster model. Partially matched. Options are "Thomas", "MatClust", "Cauchy", "VarGamma" and "LGCP".
<code>covariates</code>	The values of any spatial covariates (other than the Cartesian coordinates) required by the model. A named list of pixel images, functions, windows or numeric constants.
<code>...</code>	Arguments passed to <code>thomas.estK</code> or <code>thomas.estpcf</code> or <code>matclust.estK</code> or <code>matclust.estpcf</code> or <code>lgcp.estK</code> or <code>lgcp.estpcf</code> or <code>cauchy.estK</code> or <code>cauchy.estpcf</code> or <code>vargamma.estK</code> or <code>vargamma.estpcf</code> controlling the minimum contrast fitting algorithm.
<code>statistic</code>	The choice of summary statistic: either "K" or "pcf".
<code>statargs</code>	Optional list of arguments to be used when calculating the summary statistic. See Details.

Details

This function fits a Cox point process model to the point pattern dataset `X`. Cox models are suitable for spatially clustered point patterns.

The model may be either a *Poisson cluster process* or a *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 Matern 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, and `clusters="LGCP"` for the log-Gaussian Cox process. The first four models are Poisson cluster processes.

If the trend is constant (~ 1) then the model is *homogeneous*. 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 (matching the theoretical K -function of the model to the empirical K -function of the data, as explained in `mincontrast`).

Otherwise, the model is *inhomogeneous*. The algorithm first estimates the intensity function of the point process, by fitting a Poisson process with log intensity of the form specified by the formula `trend`. Then the inhomogeneous K function is estimated by `Kinhom` using this fitted intensity. Finally 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 `stargs`, as shown in the Examples.

Value

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

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>, and Rolf Turner <r.turner@auckland.ac.nz> with contributions from Abdollah Jalilian and Rasmus Waagepetersen.

References

Jalilian, A., Guan, Y. and Waagepetersen, R. (2011) Decomposition of variance for spatial Cox processes. Manuscript submitted for publication.

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`, `predict.kppm`, `simulate.kppm`, `update.kppm`, `vcov.kppm`, `methods.kppm`, `Kmodel.kppm`, `pcfmodel.kppm`.

Fitting algorithms: `thomas.estK`, `matclust.estK`, `lgcp.estK`, `cauchy.estK`, `vargamma.estK`, `thomas.estpcf`, `matclust.estpcf`, `lgcp.estpcf`, `cauchy.estpcf`, `vargamma.estpcf`, `mincontrast`.

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

See also `ppm`

Examples

```
data(redwood)
kppm(redwood, ~1, "Thomas")
kppm(redwood, ~x, "MatClust")
kppm(redwood, ~x, "MatClust", statistic="pcf", stargs=list(stoyan=0.2))
kppm(redwood, ~1, "LGCP", statistic="pcf")
kppm(redwood, ~x, cluster="Cauchy", statistic="K")
kppm(redwood, cluster="VarGamma", nu.ker = 0.5, statistic="pcf")
if(require(RandomFields) && RandomFieldsSafe()) {
  kppm(redwood, ~x, "LGCP", statistic="pcf",
        covmodel=list(model="matern", nu=0.3))
}
```

Kres	<i>Residual K Function</i>
------	----------------------------

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> Ege Rubak and Jesper Moller.

References

Baddeley, A., Rubak, E. and Moller, J. (2011) Score, pseudo-score and residual diagnostics for spatial point process models. *Statistical Science* **26**, 613–646.

See Also

Related functions: [Kcom](#), [Kest](#).

Alternative functions: [Gres](#), [psstG](#), [psstA](#), [psst](#).

Point process models: [ppm](#).

Examples

```

data(cells)
fit0 <- ppm(cells, ~1) # uniform Poisson

K0 <- Kres(fit0)
K0
plot(K0)
# isotropic-correction estimate
plot(K0, ires ~ r)
# uniform Poisson is clearly not correct

fit1 <- ppm(cells, ~1, Strauss(0.08))

K1 <- Kres(fit1)
plot(K1, ires ~ r)
# fit looks approximately OK; try adjusting interaction distance

plot(Kres(cells, interaction=Strauss(0.12)))

# How to make envelopes
## Not run:
E <- envelope(fit1, Kres, interaction=as.interact(fit1), nsim=19)
plot(E)

## End(Not run)

# For computational efficiency
Kc <- Kcom(fit1)
K1 <- Kres(Kc)

```

Kscaled

Locally Scaled K-function

Description

Estimates the template K function of a locally-scaled point process.

Usage

```

Kscaled(X, lambda=NULL, ..., r = NULL, breaks = NULL,
        correction=c("border", "isotropic", "translate"),
        sigma=NULL, varcov=NULL)
Lscaled(...)

```

Arguments

- | | |
|---------------------|--|
| <code>X</code> | The observed data point pattern, from which an estimate of the locally scaled K function will be computed. An object of class "ppp" or in a format recognised by <code>as.ppp()</code> . |
| <code>lambda</code> | Optional. Values of the estimated intensity function. Either a vector giving the intensity values at the points of the pattern <code>X</code> , a pixel image (object of class "im") giving the intensity values at all locations, or a function(<code>x,y</code>) which can be evaluated to give the intensity value at any location. |

...	Arguments passed from Lscaled to Kscaled and from Kscaled to density.ppp if lambda is omitted.
r	vector of values for the argument r at which the locally scaled K function should be evaluated. Not normally given by the user; there is a sensible default.
breaks	An alternative to the argument r. Not normally invoked by the user. See Details.
correction	A character vector containing any selection of the options "border", "isotropic", "Ripley", "translate", "none" or "best". It specifies the edge correction(s) to be applied.
sigma, varcov	Optional arguments passed to density.ppp to control the smoothing bandwidth, when lambda is estimated by kernel smoothing.

Details

Kscaled computes an estimate of the K function for a locally scaled point process. Lscaled computes the corresponding L function $L(r) = \sqrt{K(r)/\pi}$.

Locally scaled point processes are a class of models for inhomogeneous point patterns, introduced by Hahn et al (2003). They include inhomogeneous Poisson processes, and many other models.

The template K function of a locally-scaled process is a counterpart of the “ordinary” Ripley K function, in which the distances between points of the process are measured on a spatially-varying scale (such that the locally rescaled process has unit intensity).

The template K function is an indicator of interaction between the points. For an inhomogeneous Poisson process, the theoretical template K function is approximately equal to $K(r) = \pi r^2$. Values $K_{\text{scaled}}(r) > \pi r^2$ are suggestive of clustering.

Kscaled computes an estimate of the template K function and Lscaled computes the corresponding L function $L(r) = \sqrt{K(r)/\pi}$.

The locally scaled interpoint distances are computed using an approximation proposed by Hahn (2007). The Euclidean distance between two points is multiplied by the average of the square roots of the intensity values at the two points.

The argument lambda should supply the (estimated) values of the intensity function λ . It may be either

a numeric vector containing the values of the intensity function at the points of the pattern X.

a pixel image (object of class "im") assumed to contain the values of the intensity function at all locations in the window.

a function which can be evaluated to give values of the intensity at any locations.

omitted: if lambda is omitted, then it will be estimated using a ‘leave-one-out’ kernel smoother.

If lambda is a numeric vector, then its length should be equal to the number of points in the pattern X. The value lambda[i] is assumed to be the the (estimated) value of the intensity $\lambda(x_i)$ for the point x_i of the pattern X. Each value must be a positive number; NA's are not allowed.

If lambda is a pixel image, the domain of the image should cover the entire window of the point pattern. If it does not (which may occur near the boundary because of discretisation error), then the missing pixel values will be obtained by applying a Gaussian blur to lambda using blur, then looking up the values of this blurred image for the missing locations. (A warning will be issued in this case.)

If lambda is a function, then it will be evaluated in the form lambda(x,y) where x and y are vectors of coordinates of the points of X. It should return a numeric vector with length equal to the number of points in X.

If `lambda` is omitted, then it will be estimated using a ‘leave-one-out’ kernel smoother, as described in Baddeley, Moller and Waagepetersen (2000). The estimate `lambda[i]` for the point $X[i]$ is computed by removing $X[i]$ from the point pattern, applying kernel smoothing to the remaining points using `density.ppp`, and evaluating the smoothed intensity at the point $X[i]$. The smoothing kernel bandwidth is controlled by the arguments `sigma` and `varcov`, which are passed to `density.ppp` along with any extra arguments.

Edge corrections are used to correct bias in the estimation of K_{scaled} . First the interpoint distances are rescaled, and then edge corrections are applied as in `Kest`. See `Kest` for details of the edge corrections and the options for the argument `correction`.

The pair correlation function can also be applied to the result of `Kscaled`; see `pcf` and `pcf.fv`.

Value

An object of class “fv” (see `fv.object`).

Essentially a data frame containing at least the following columns,

<code>r</code>	the vector of values of the argument r at which the pair correlation function $g(r)$ has been estimated
<code>theo</code>	vector of values of πr^2 , the theoretical value of $K_{\text{scaled}}(r)$ for an inhomogeneous Poisson process

and containing additional columns according to the choice specified in the `correction` argument. The additional columns are named `border`, `trans` and `iso` and give the estimated values of $K_{\text{scaled}}(r)$ using the border correction, translation correction, and Ripley isotropic correction, respectively.

Author(s)

Ute Hahn, Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Hahn, U. (2007) *Global and Local Scaling in the Statistics of Spatial Point Processes*. Habilitationsschrift, Universitaet Augsburg.
- Hahn, U., Jensen, E.B.V., van Lieshout, M.N.M. and Nielsen, L.S. (2003) Inhomogeneous spatial point processes by location-dependent scaling. *Advances in Applied Probability* **35**, 319–336.
- Prokesova, M., Hahn, U. and Vedel Jensen, E.B. (2006) Statistics for locally scaled point patterns. In A. Baddeley, P. Gregori, J. Mateu, R. Stoica and D. Stoyan (eds.) *Case Studies in Spatial Point Pattern Modelling*. Lecture Notes in Statistics 185. New York: Springer Verlag. Pages 99–123.

See Also

`Kest`, `pcf`

Examples

```
data(bronzefilter)
X <- unmark(bronzefilter)
K <- Kscaled(X)
fit <- ppm(X, ~x)
lam <- predict(fit)
K <- Kscaled(X, lam)
```

kstest.ppm*Kolmogorov-Smirnov Test for Point Process Model*

Description

Performs a Kolmogorov-Smirnov test of goodness-of-fit of a Poisson point process model. The test compares the observed and predicted distributions of the values of a spatial covariate.

Usage

```
kstest(...)
## S3 method for class 'ppp'
kstest(X, covariate, ..., jitter=TRUE)
## S3 method for class 'ppm'
kstest(model, covariate, ..., jitter=TRUE)
## S3 method for class 'slrm'
kstest(model, covariate, ..., modelname=NULL, covname=NULL)
```

Arguments

X	A point pattern (object of class "ppp").
model	A fitted point process model (object of class "ppm") 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.
...	Arguments passed to ks.test to control the test.
jitter	Logical flag. If <code>jitter=TRUE</code> , 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 <code>model</code> and <code>covariate</code> to be used in labelling plot axes.

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 the Kolmogorov-Smirnov test.

The function `kstest` is generic, with methods for point patterns ("ppp"), point process models ("ppm") and spatial logistic regression models ("slrm").

- If `X` is a point pattern dataset (object of class "ppp"), then `kstest(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") then `kstest(model, ...)` performs a test of goodness-of-fit for this fitted model. In this case, `model` should be a Poisson point process.

- If `model` is a fitted spatial logistic regression (object of class "slrm") then `kstest(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 the classical Kolmogorov-Smirnov 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 Kolmogorov-Smirnov test of uniformity is applied using `ks.test`.

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

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 "kstest" for which there is a plot method `plot.kstest`. 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 "kstest" 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 `kstest` 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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Baddeley, A., Turner, R., Moller, 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.kstest`, `quadrat.test`, `bermantest`, `ks.test`, `ppm`

Examples

```
# real data: NZ trees
data(nztrees)

# test of CSR using x coordinate
kstest(nztrees, "x")

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

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

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

# 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
kstest(fit0, "x")
# test right model
kstest(fit1, "x")

# multitype point pattern
```

```
data(amacrine)
kstest(amacrine, "x")
yimage <- as.im(function(x,y){y}, W=as.owin(amacrine))
kstest(ppm(amacrine, ~marks+y), yimage)
```

LambertW*Lambert's W Function***Description**

Computes Lambert's W-function.

Usage

```
LambertW(x)
```

Arguments

<code>x</code>	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.

Value

Numeric vector.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Corless, R, Gonnet, G, Hare, D, Jeffrey, D and Knuth, D (1996), On the Lambert W function. *Computational Mathematics*, **5**, 325–359.
- Roy, R and Olver, F (2010), Lambert W function. In Olver, F, Lozier, D and Boisvert, R (eds.), *NIST Handbook of Mathematical Functions*, Cambridge University Press.

Examples

```
LambertW(exp(1))
```

lansing*Lansing Woods Point Pattern*

Description

Locations and botanical classification of trees in Lansing Woods.

The data come from an investigation of a 924 ft x 924 ft (19.6 acre) plot in Lansing Woods, Clinton County, Michigan USA by D.J. Gerrard. The data give the locations of 2251 trees and their botanical classification (into hickories, maples, red oaks, white oaks, black oaks and miscellaneous trees). The original plot size (924 x 924 feet) has been rescaled to the unit square.

Usage

```
data(lansing)
```

Format

An object of class "ppp" representing the point pattern of tree locations. Entries include

x	Cartesian <i>x</i> -coordinate of tree
y	Cartesian <i>y</i> -coordinate of tree
marks	factor with levels indicating species of each tree

The levels of marks are blackoak, hickory, maple, misc, redoak and whiteoak. See [ppp.object](#) for details of the format of a point pattern object.

References

- Besag, J. (1978) Some methods of statistical analysis for spatial data. *Bull. Internat. Statist. Inst.* **44**, 77–92.
- Cox, T.F. (1976) The robust estimation of the density of a forest stand using a new conditioned distance method. *Biometrika* **63**, 493–500.
- Cox, T.F. (1979) A method for mapping the dense and sparse regions of a forest stand. *Applied Statistics* **28**, 14–19.
- Cox, T.F. and Lewis, T. (1976) A conditioned distance ratio method for analysing spatial patterns. *Biometrika* **63**, 483–492.
- Diggle, P.J. (1979a) The detection of random heterogeneity in plant populations. *Biometrics* **33**, 390–394.
- Diggle, P.J. (1979b) Statistical methods for spatial point patterns in ecology. *Spatial and temporal analysis in ecology*. R.M. Cormack and J.K. Ord (eds.) Fairland: International Co-operative Publishing House. pages 95–150.
- Diggle, P.J. (1981) Some graphical methods in the analysis of spatial point patterns. In *Interpreting Multivariate Data*. V. Barnett (eds.) John Wiley and Sons. Pages 55–73.
- Diggle, P.J. (1983) *Statistical analysis of spatial point patterns*. Academic Press.
- Gerrard, D.J. (1969) Competition quotient: a new measure of the competition affecting individual forest trees. Research Bulletin 20, Agricultural Experiment Station, Michigan State University.
- Lotwick, H.W. (1981) *Spatial stochastic point processes*. PhD thesis, University of Bath, UK.
- Ord, J.K. (1978) How many trees in a forest? *Mathematical Scientist* **3**, 23–33.

Examples

```
data(lansing)
plot(lansing)
plot(split(lansing))
plot(split(lansing)$maple)
```

`latest.news`

Print News About Latest Version of Package

Description

Prints the news documentation for the current version of **spatstat** or another specified package.

Usage

```
latest.news(package = "spatstat")
```

Arguments

package	Name of package for which the news file
---------	---

Details

By default, this function prints the news documentation about changes in the current installed version of the **spatstat** package.

If `package` is given, then the function reads the news for the specified package from its NEWS file (if it has one) and prints only the entries that refer to the current version of the package.

To see the news for all previous versions as well as the current version, use the R utility [news](#). See the Examples.

Value

Null.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[news](#)

Examples

```
# current news
latest.news()
# all news
news(package="spatstat")
```

layered*Create List of Plotting Layers*

Description

Given several objects which are capable of being plotted, create a list containing these objects as if they were successive layers of a plot. The list can then be plotted in different ways.

Usage

```
layered(..., plotargs = NULL)
```

Arguments

- | | |
|-----------------------|--|
| ... | Objects which can be plotted by <code>plot</code> . |
| <code>plotargs</code> | Default values of the plotting arguments for each of the objects. A list of lists of arguments of the form <code>name=value</code> . |

Details

Layering is a simple mechanism for controlling a high-level plot that is composed of several successive plots, for example, a background and a foreground plot. The layering mechanism makes it easier to issue the `plot` command, to switch on or off the plotting of each individual layer, and to control the plotting arguments that are passed to each layer.

Each individual layer in the plot should be saved as an object that can be plotted using `plot`. It will typically belong to some class, which has a method for the generic function `plot`.

The command `layered` simply saves the objects ... as a list of class "layered". This list can then be plotted by the method `plot.layered`. Thus, you only need to type a single `plot` command to produce the multi-layered plot. Individual layers of the plot can be switched on or off, or manipulated, using arguments to `plot.layered`.

The argument `plotargs` contains default values of the plotting arguments for each layer. It should be a list, with one entry for each object in Each entry of `plotargs` should be a list of arguments in the form `name=value`, which are recognised by the `plot` method for the relevant layer.

Value

A list, belonging to the class "layered".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[plot.layered](#)

Examples

```
data(cells)
D <- distmap(cells)
L <- layered(D, cells)
L
L <- layered(D, cells,
  plotargs=list(list(ribbon=FALSE), list(pch=16)))
plot(L)
```

Lcross

Multitype L-function (cross-type)

Description

Calculates an estimate of the cross-type L-function for a multitype point pattern.

Usage

```
Lcross(X, i, j, ...)
```

Arguments

X	The observed point pattern, from which an estimate of the cross-type L function $L_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of <code>marks(X)</code> .
j	The type (mark value) of the points in X to which distances are measured. A character string (or something that will be converted to a character string). Defaults to the second level of <code>marks(X)</code> .
...	Arguments passed to Kcross .

Details

The cross-type L-function is a transformation of the cross-type K-function,

$$L_{ij}(r) = \sqrt{\frac{K_{ij}(r)}{\pi}}$$

where $K_{ij}(r)$ is the cross-type K-function from type i to type j. See [Kcross](#) for information about the cross-type K-function.

The command `Lcross` first calls [Kcross](#) to compute the estimate of the cross-type K-function, and then applies the square root transformation.

For a marked point pattern in which the points of type i are independent of the points of type j, the theoretical value of the L-function is $L_{ij}(r) = r$. The square root also has the effect of stabilising the variance of the estimator, so that L_{ij} is more appropriate for use in simulation envelopes and hypothesis tests.

Value

An object of class "fv", see [fv.object](#), which can be plotted directly using [plot.fv](#).

Essentially a data frame containing columns

`r` the vector of values of the argument r at which the function L_{ij} has been estimated

`theo` the theoretical value $L_{ij}(r) = r$ for a stationary Poisson process

together with columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function L_{ij} obtained by the edge corrections named.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[Kcross](#), [Ldot](#), [Lest](#)

Examples

```
data(amacrine)
L <- Lcross(amacrine, "off", "on")
plot(L)
```

Lcross.inhom

Inhomogeneous Cross Type L Function

Description

For a multitype point pattern, estimate the inhomogeneous version of the cross-type L function.

Usage

```
Lcross.inhom(X, i, j, ...)
```

Arguments

`X` The observed point pattern, from which an estimate of the inhomogeneous cross type L function $L_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.

`i` The type (mark value) of the points in `X` from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of `marks(X)`.

`j` The type (mark value) of the points in `X` to which distances are measured. A character string (or something that will be converted to a character string). Defaults to the second level of `marks(X)`.

`...` Other arguments passed to [Kcross.inhom](#).

Details

This is a generalisation of the function [Lcross](#) to include an adjustment for spatially inhomogeneous intensity, in a manner similar to the function [Linhom](#).

All the arguments are passed to [Kcross.inhom](#), which estimates the inhomogeneous multitype K function $K_{ij}(r)$ for the point pattern. The resulting values are then transformed by taking $L(r) = \sqrt{K(r)/\pi}$.

Value

An object of class "fv" (see [fv.object](#)).

Essentially a data frame containing numeric columns

r	the values of the argument r at which the function $L_{ij}(r)$ has been estimated
theo	the theoretical value of $L_{ij}(r)$ for a marked Poisson process, identically equal to r

together with a column or columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $L_{ij}(r)$ obtained by the edge corrections named.

Warnings

The arguments i and j are always interpreted as levels of the factor X\$marks. They are converted to character strings if they are not already character strings. The value i=1 does **not** refer to the first level of the factor.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

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

See Also

[Lcross](#), [Linhom](#), [Kcross.inhom](#)

Examples

```
# Lansing Woods data
data(lansing)
lansing <- lansing[seq(1,lansing$n, by=10)]
ma <- split(lansing)$maple
wh <- split(lansing)$whiteoak

# method (1): estimate intensities by nonparametric smoothing
lambdaM <- density.ppp(ma, sigma=0.15, at="points")
lambdaW <- density.ppp(wh, sigma=0.15, at="points")
L <- Lcross.inhom(lansing, "whiteoak", "maple", lambdaW, lambdaM)

# method (2): fit parametric intensity model
```

```

fit <- ppm(lansing, ~marks * polynom(x,y,2))
# evaluate fitted intensities at data points
# (these are the intensities of the sub-processes of each type)
inten <- fitted(fit, dataonly=TRUE)
# split according to types of points
lambda <- split(inten, lansing$marks)
L <- Lcross.inhom(lansing, "whiteoak", "maple",
                   lambda$whiteoak, lambda$maple)

# synthetic example: type A points have intensity 50,
# type B points have intensity 100 * x
lamB <- as.im(function(x,y){50 + 100 * x}, owin())
X <- superimpose(A=runifpoispp(50), B=rpoispp(lamB))
L <- Lcross.inhom(X, "A", "B",
                   lambdaI=as.im(50, X>window), lambdaJ=lamB)

```

Ldot*Multitype L-function (i-to-any)***Description**

Calculates an estimate of the multitype L-function (from type *i* to any type) for a multitype point pattern.

Usage

```
Ldot(X, i, ...)
```

Arguments

- | | |
|----------|---|
| <i>X</i> | The observed point pattern, from which an estimate of the dot-type <i>L</i> function $L_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details. |
| <i>i</i> | The type (mark value) of the points in <i>X</i> from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of <code>marks(X)</code> . |
| ... | Arguments passed to Kdot . |

Details

This command computes

$$L_{i\bullet}(r) = \sqrt{\frac{K_{i\bullet}(r)}{\pi}}$$

where $K_{i\bullet}(r)$ is the multitype *K*-function from points of type *i* to points of any type. See [Kdot](#) for information about $K_{i\bullet}(r)$.

The command `Ldot` first calls [Kdot](#) to compute the estimate of the *i*-to-any *K*-function, and then applies the square root transformation.

For a marked Poisson point process, the theoretical value of the *L*-function is $L_{i\bullet}(r) = r$. The square root also has the effect of stabilising the variance of the estimator, so that $L_{i\bullet}$ is more appropriate for use in simulation envelopes and hypothesis tests.

Value

An object of class "fv", see [fv.object](#), which can be plotted directly using [plot.fv](#).

Essentially a data frame containing columns

`r` the vector of values of the argument r at which the function $L_{i\bullet}$ has been estimated

`theo` the theoretical value $L_{i\bullet}(r) = r$ for a stationary Poisson process

together with columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $L_{i\bullet}$ obtained by the edge corrections named.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[Kdot](#), [Lcross](#), [Lest](#)

Examples

```
data(amacrine)
L <- Ldot(amacrine, "off")
plot(L)
```

Ldot.inhom

Inhomogeneous Multitype L Dot Function

Description

For a multitype point pattern, estimate the inhomogeneous version of the dot L function.

Usage

```
Ldot.inhom(X, i, ...)
```

Arguments

`X` The observed point pattern, from which an estimate of the inhomogeneous cross type L function $L_{i\bullet}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). See under Details.

`i` The type (mark value) of the points in `X` from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of `marks(X)`.

`...` Other arguments passed to [Kdot.inhom](#).

Details

This a generalisation of the function [Ldot](#) to include an adjustment for spatially inhomogeneous intensity, in a manner similar to the function [Linhom](#).

All the arguments are passed to [Kdot.inhom](#), which estimates the inhomogeneous multitype K function $K_{i\bullet}(r)$ for the point pattern. The resulting values are then transformed by taking $L(r) = \sqrt{K(r)/\pi}$.

Value

An object of class "fv" (see [fv.object](#)).

Essentially a data frame containing numeric columns

r	the values of the argument r at which the function $L_{i\bullet}(r)$ has been estimated
theo	the theoretical value of $L_{i\bullet}(r)$ for a marked Poisson process, identical to r .

together with a column or columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $L_{i\bullet}(r)$ obtained by the edge corrections named.

Warnings

The argument i is interpreted as a level of the factor Xmarks$. It is converted to a character string if it is not already a character string. The value $i=1$ does **not** refer to the first level of the factor.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

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

See Also

[Ldot](#), [Linhom](#), [Kdot.inhom](#), [Lcross.inhom](#).

Examples

```
# Lansing Woods data
data(lansing)
lansing <- lansing[seq(1,lansing$n, by=10)]
ma <- split(lansing)$maple
lg <- unmark(lansing)

# Estimate intensities by nonparametric smoothing
lambdaM <- density.ppp(ma, sigma=0.15, at="points")
lambdadot <- density.ppp(lg, sigma=0.15, at="points")
L <- Ldot.inhom(lansing, "maple", lambdaI=lambdaM,
                 lambdadot=lambdadot)

# synthetic example: type A points have intensity 50,
```

```
#           type B points have intensity 50 + 100 * x
lamB <- as.im(function(x,y){50 + 100 * x}, owin())
lamdot <- as.im(function(x,y) { 100 + 100 * x}, owin())
X <- superimpose(A=runifpoispp(50), B=rpoispp(lamB))
L <- Ldot.inhom(X, "B", lambdaI=lamB, lambdadot=lamdot)
```

lengths.psp*Lengths of Line Segments***Description**

Computes the length of each line segment in a line segment pattern.

Usage

```
lengths.psp(x)
```

Arguments

x	A line segment pattern (object of class "psp").
---	---

Details

The length of each line segment is computed and the lengths are returned as a numeric vector.

Value

Numeric vector.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[summary.psp](#), [midpoints.psp](#), [angles.psp](#)

Examples

```
a <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
b <- lengths.psp(a)
```

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 $\exp(\epsilon)$ occurring 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 $d/\text{sigma0}$.

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

Lennard-Jones, J.E. (1924) On the determination of molecular fields. *Proc Royal Soc London A* **106**, 463–477.

See Also

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

Examples

```
data(demopat)
demopat
X <- unmark(demopat)
X
fit <- ppm(X, ~1, LennardJones(), rbord=500)
fit
plot(fitin(fit), xlim=c(0,50))
```

Description

Calculates an estimate of the L -function (Besag’s transformation of Ripley’s K -function) for a spatial point pattern.

Usage

```
Lest(X, ...)
```

Arguments

- X The observed point pattern, from which an estimate of $L(r)$ will be computed.
 An object of class "ppp", or data in any format acceptable to [as.ppp\(\)](#).
- ... Other arguments passed to [Kest](#) to control the estimation procedure.

Details

This command computes an estimate of the L -function for the spatial point pattern X. The L -function is a transformation of Ripley's K -function,

$$L(r) = \sqrt{\frac{K(r)}{\pi}}$$

where $K(r)$ is the K -function.

See [Kest](#) for information about Ripley's K -function. The transformation to L was proposed by Besag (1977).

The command [Lest](#) first calls [Kest](#) to compute the estimate of the K -function, and then applies the square root transformation.

For a completely random (uniform Poisson) point pattern, the theoretical value of the L -function is $L(r) = r$. The square root also has the effect of stabilising the variance of the estimator, so that K is more appropriate for use in simulation envelopes and hypothesis tests.

See [Kest](#) for the list of arguments.

Value

An object of class "fv", see [fv.object](#), which can be plotted directly using [plot.fv](#).

Essentially a data frame containing columns

- | | |
|------|---|
| r | the vector of values of the argument r at which the function L has been estimated |
| theo | the theoretical value $L(r) = r$ for a stationary Poisson process |

together with columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $L(r)$ obtained by the edge corrections named.

Variance approximations

If the argument `var.approx=TRUE` is given, the return value includes columns `rip` and `ls` containing approximations to the variance of $\hat{L}(r)$ under CSR. These are obtained by the delta method from the variance approximations described in [Kest](#).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

Besag, J. (1977) Discussion of Dr Ripley's paper. *Journal of the Royal Statistical Society, Series B*, **39**, 193–195.

See Also

[Kest](#), [pcf](#)

Examples

```
data(cells)
L <- Lest(cells)
plot(L, main="L function for cells")
```

letterR

Window in Shape of Letter R

Description

A window in the shape of the capital letter R, for use in demonstrations.

Usage

```
data(letterR)
```

Format

An object of class "owin" representing the capital letter R, in the same font as the R package logo. See [owin.object](#) for details of the format.

Source

Adrian Baddeley

levelset

Level Set of a Pixel Image

Description

Given a pixel image, find all pixels which have values less than a specified threshold value (or greater than a threshold, etc), and assemble these pixels into a window.

Usage

```
levelset(X, thresh, compare="<=")
```

Arguments

X	A pixel image (object of class "im").
thresh	Threshold value. A single number or value compatible with the pixel values in X.
compare	Character string specifying one of the comparison operators "<", ">", "==" , "<=", ">=" , "!=".

Details

If X is a pixel image with numeric values, then `levelset(X, thresh)` finds the region of space where the pixel values are less than or equal to the threshold value `thresh`. This region is returned as a spatial window.

The argument `compare` specifies how the pixel values should be compared with the threshold value. Instead of requiring pixel values to be less than or equal to `thresh`, you can specify that they must be less than (<), greater than (>), equal to (==), greater than or equal to (>=), or not equal to (!=) the threshold value `thresh`.

If X has non-numeric pixel values (for example, logical or factor values) it is advisable to use only the comparisons == and !=, unless you really know what you are doing.

For more complicated logical comparisons, see [solutionset](#).

Value

A spatial window (object of class "owin", see [owin.object](#)) containing the pixels satisfying the constraint.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[im.object](#), [as.owin](#), [solutionset](#).

Examples

```
# test image
X <- as.im(function(x,y) { x^2 - y^2 }, unit.square())

W <- levelset(X, 0.2)
W <- levelset(X, -0.3, ">")

# compute area of level set
area.owin(levelset(X, 0.1))
```

leverage.ppmLeverage 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=list())
```

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

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 (2011).

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 trend 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 plotted (by `plot.leverage.ppm`) or converted to a pixel image by `as.im` (see `as.im.leverage.ppm`).

Value

An object of class "leverage.ppm" that can be plotted by `plot.leverage.ppm`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

References

Baddeley, A. and Chang, Y.M. and Song, Y. (2011) Leverage and influence diagnostics for spatial point process models. *Scandinavian Journal of Statistics*, in press.

See Also

[influence.ppm](#), [dfbetas.ppm](#), [plot.leverage.ppm](#)

Examples

```
X <- rpoispp(function(x,y) { exp(3+3*x) })
fit <- ppm(X, ~x+y)
plot(leverage(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(sigma2=1,alpha=1),
           covmodel=list(model="exponential"),
           lambda=NULL,
           q = 1/4, p = 2, rmin = NULL, rmax = NULL, ...)
```

Arguments

- | | |
|---|---|
| X
startpar
covmodel
lambda
q, p
rmin, rmax
... | Data to which the model will be fitted. Either a point pattern or a summary statistic. See Details.
Vector of starting values for the parameters of the log-Gaussian Cox process model.
Specification of the covariance model for the log-Gaussian field. See Details.
Optional. An estimate of the intensity of the point process.
Optional. Exponents for the contrast criterion.
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 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 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 (Moller 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

$$K(r) = \int_0^r 2\pi s \exp(C(s)) ds.$$

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. 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*. It may be any of the covariance functions recognised by the command [Covariance](#) in the **RandomFields** package. The default is the exponential covariance $c(r) = e^{-r}$ so that the scaled covariance is

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

The argument *covmodel* should be of the form `list(model="modelname", ...)` where *modelname* is the string name of one of the covariance models recognised by the command [Covariance](#) in the **RandomFields** package, and ... are arguments of the form *tag*=*value* giving the values of parameters controlling the shape of these models. For example the exponential covariance is specified by `covmodel=list(model="exponential")` while the Matern covariance with exponent $\nu = 0.3$ is specified by `covmodel=list(model="matern", nu=0.3)`.

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:

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

Note

This function is considerably slower than [lgcp.estpcf](#) because of the computation time required for the integral in the K -function.

Computation can be accelerated, at the cost of less accurate results, by setting `spatstat.options(fastK.lgcp=TRUE)`.

Author(s)

Rasmus Waagepetersen <rwd@math.auc.dk>. Adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> Further modifications by Rasmus Waagepetersen and Shen Guochun.

References

- Moller, J., Syversveen, A. and Waagepetersen, R. (1998) Log Gaussian Cox Processes. *Scandinavian Journal of Statistics* **25**, 451–482.
- Moller, J. and Waagepetersen, R. (2003). Statistical Inference and Simulation for Spatial Point Processes. Chapman and Hall/CRC, Boca Raton.
- Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

- [lgcp.estpcf](#) for alternative method of fitting LGCP.
- [matclust.estK](#), [thomas.estK](#) for other models.
- [mincontrast](#) for the generic minimum contrast fitting algorithm, including important parameters that affect the accuracy of the fit.
- [Covariance in the RandomFields package](#), for covariance function models.
- [Kest](#) for the K function.

Examples

```

u <- lgcp.estK(redwood, c(sigma2=0.1, alpha=1))
u
if(interactive()) plot(u)

if(require(RandomFields) && RandomFieldsSafe()) {
  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 (with exponential covariance function) to a point pattern dataset by the Method of Minimum Contrast using the pair correlation function.

Usage

```
lgcp.estpcf(X,
            startpar=c(sigma2=1,alpha=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 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 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 (Moller 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`. It may be any of the covariance functions recognised by the command [Covariance](#) in the **RandomFields** package. The default is the exponential covariance $c(r) = e^{-r}$ so that the scaled covariance is

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

The argument `covmodel` should be of the form `list(model="modelname", ...)` where `modelname` is the string name of one of the covariance models recognised by the command [Covariance](#) in the **RandomFields** package, and `...` are arguments of the form `tag=value` giving the values of parameters controlling the shape of these models. For example the exponential covariance is specified by `covmodel=list(model="exponential")` while the Matern covariance with exponent $\nu = 0.3$ is specified by `covmodel=list(model="matern", nu=0.3)`.

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>, with modifications by Shen Guochun and Rasmus Waagepetersen <rwd@math.auc.dk>

References

- Moller, J, Syversveen, A. and Waagepetersen, R. (1998) Log Gaussian Cox Processes. *Scandinavian Journal of Statistics* **25**, 451–482.
- Moller, J. and Waagepetersen, R. (2003). Statistical Inference and Simulation for Spatial Point Processes. Chapman and Hall/CRC, Boca Raton.
- Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

- [lgcp.estK](#) for alternative method of fitting LGCP.
- [matclust.estpcf](#), [thomas.estpcf](#) for other models.
- [mincontrast](#) for the generic minimum contrast fitting algorithm, including important parameters that affect the accuracy of the fit.
- [Covariance](#) in the **RandomFields** package, for covariance function models.
- [pcf](#) for the pair correlation function.

Examples

```
data(redwood)
u <- lgcp.estpcf(redwood, c(sigma2=0.1, alpha=1))
u
plot(u)
if(require(RandomFields) && RandomFieldsSafe()) {
  lgcp.estpcf(redwood, covmodel=list(model="matern", nu=0.3))
}
```

lineardisc*Compute Disc of Given Radius in Linear Network*

Description

Computes the ‘disc’ of given radius and centre in a linear network.

Usage

```
lineardisc(L, x = locator(1), r, plotit = TRUE,
           cols=c("blue", "red", "green"))
countends(L, x = locator(1), r)
```

Arguments

L	Linear network (object of class "linnet").
x	Location of centre of disc. Either a point pattern (object of class "ppp") containing exactly 1 point, or a numeric vector of length 2.
r	Radius of disc.
plotit	Logical. Whether to plot the disc.
cols	Colours for plotting the disc. A numeric or character vector of length 3 specifying the colours of the disc centre, disc lines and disc endpoints respectively.

Details

The ‘disc’ $B(u, r)$ of centre x and radius r in a linear network L is the set of all points u in L such that the shortest path distance from x to u is less than or equal to r . This is a union of line segments contained in L .

The *relative boundary* of the disc $B(u, r)$ is the set of points v such that the shortest path distance from x to v is *equal* to r .

The function `lineardisc` computes the disc of radius r and its relative boundary, optionally plots them, and returns them. The faster function `countends` simply counts the number of points in the relative boundary.

Value

The value of `lineardisc` is a list with two entries:

lines	Line segment pattern (object of class "psp") representing the interior disc
endpoints	Point pattern (object of class "ppp") representing the relative boundary of the disc.

The value of `countends` is an integer giving the number of points in the relative boundary.

Author(s)

Ang Qi Wei <aqw07398@hotmail.com> and Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

References

- Ang, Q.W. (2010) *Statistical methodology for events on a network*. Master's thesis, School of Mathematics and Statistics, University of Western Australia.
- Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. To appear in *Scandinavian Journal of Statistics*.

See Also

[linnet](#)

Examples

```
example(linnet)
lineardisc(letterA, c(0,3), 1.6)
# count the endpoints
countends(letterA, c(0,3), 1.6)
# cross-check (slower)
lineardisc(letterA, c(0,3), 1.6, plotit=FALSE)$endpoints$n
```

linearK

Linear K Function

Description

Computes an estimate of the linear K function for a point pattern on a linear network.

Usage

```
linearK(X, r=NULL, ..., correction="Ang")
```

Arguments

X	Point pattern on linear network (object of class "lpp").
r	Optional. Numeric vector of values of the function argument r . There is a sensible default.
...	Ignored.
correction	Geometry correction. Either "none" or "Ang". See Details.

Details

This command computes the linear K function from point pattern data on a linear network.

If `correction="none"`, the calculations do not include any correction for the geometry of the linear network. The result is the network K function as defined by Okabe and Yamada (2001).

If `correction="Ang"`, the pair counts are weighted using Ang's correction (Ang, 2010; Ang et al, 2012).

Value

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

Author(s)

Ang Qi Wei <aqw07398@hotmail.com> and Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

References

- Ang, Q.W. (2010) Statistical methodology for spatial point patterns on a linear network. MSc thesis, University of Western Australia.
- Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. To appear in *Scandinavian Journal of Statistics*.
- Okabe, A. and Yamada, I. (2001) The K-function method on a network and its computational implementation. *Geographical Analysis* **33**, 271-290.

See Also

[compileK](#), [lpp](#)

Examples

```
data(simpnet)
X <- rpoislpp(5, simpnet)
linearK(X)
linearK(X, correction="none")
```

linearKinhom

Inhomogeneous Linear K Function

Description

Computes an estimate of the inhomogeneous linear K function for a point pattern on a linear network.

Usage

```
linearKinhom(X, lambda=NULL, r=NULL, ..., correction="Ang", normalise=TRUE)
```

Arguments

- | | |
|------------|---|
| X | Point pattern on linear network (object of class "lpp"). |
| lambda | Intensity values for the point pattern. Either a numeric vector, a function, a pixel image (object of class "im" or "linim") or a fitted point process model (object of class "ppm" or "lppm"). |
| r | Optional. Numeric vector of values of the function argument r . There is a sensible default. |
| ... | Ignored. |
| correction | Geometry correction. Either "none" or "Ang". See Details. |
| normalise | Logical. If TRUE (the default), the denominator of the estimator is data-dependent (equal to the sum of the reciprocal intensities at the data points), which reduces the sampling variability. If FALSE, the denominator is the length of the network. |

Details

This command computes the inhomogeneous version of the linear K function from point pattern data on a linear network.

If `lambda = NULL` the result is equivalent to the homogeneous K function `linearK`. If `lambda` is given, then it is expected to provide estimated values of the intensity of the point process at each point of X . The argument `lambda` may be a numeric vector (of length equal to the number of points in X), or a `function(x,y)` that will be evaluated at the points of X to yield numeric values, or a pixel image (object of class "im") or a fitted point process model (object of class "ppm" or "lppm").

If `correction="none"`, the calculations do not include any correction for the geometry of the linear network. If `correction="Ang"`, the pair counts are weighted using Ang's correction (Ang, 2010).

Value

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

Author(s)

Ang Qi Wei <aqw07398@hotmail.com> and Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

References

Ang, Q.W. (2010) Statistical methodology for spatial point patterns on a linear network. MSc thesis, University of Western Australia.

Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. To appear in *Scandinavian Journal of Statistics*.

See Also

[lpp](#)

Examples

```
data(simpnet)
X <- rpoislpp(5, simpnet)
fit <- lppm(X, ~x)
K <- linearKinhom(X, lambda=fit)
plot(K)
```

linearpf

Linear Pair Correlation Function

Description

Computes an estimate of the linear pair correlation function for a point pattern on a linear network.

Usage

```
linearpf(X, r=NULL, ..., correction="Ang")
```

Arguments

X	Point pattern on linear network (object of class "lpp").
r	Optional. Numeric vector of values of the function argument r . There is a sensible default.
...	Arguments passed to <code>density.default</code> to control the smoothing.
correction	Geometry correction. Either "none" or "Ang". See Details.

Details

This command computes the linear pair correlation function from point pattern data on a linear network.

The pair correlation function is estimated from the shortest-path distances between each pair of data points, using the fixed-bandwidth kernel smoother `density.default`, with a bias correction at each end of the interval of r values. To switch off the bias correction, set `endcorrect=FALSE`.

If `correction="none"`, the calculations do not include any correction for the geometry of the linear network. The result is an estimate of the first derivative of the network K function defined by Okabe and Yamada (2001).

If `correction="Ang"`, the pair counts are weighted using Ang's correction (Ang, 2010). The result is an estimate of the pair correlation function in the linear network.

Value

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

Author(s)

Ang Qi Wei <aqw07398@hotmail.com> and Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

References

- Ang, Q.W. (2010) Statistical methodology for spatial point patterns on a linear network. MSc thesis, University of Western Australia.
- Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. To appear in *Scandinavian Journal of Statistics*.
- Okabe, A. and Yamada, I. (2001) The K-function method on a network and its computational implementation. *Geographical Analysis* **33**, 271-290.

See Also

`linearK`, `linearpccf.inhom`, `lpp`

Examples

```
data(simpnet)
X <- rpoislpp(5, simpnet)
linearpacf(X)
linearpacf(X, correction="none")
```

linearpctinhom*Inhomogeneous Linear Pair Correlation Function***Description**

Computes an estimate of the inhomogeneous linear pair correlation function for a point pattern on a linear network.

Usage

```
linearpctinhom(X, lambda=NULL, r=NULL, ..., correction="Ang", normalise=TRUE)
```

Arguments

X	Point pattern on linear network (object of class "lpp").
lambda	Intensity values for the point pattern. Either a numeric vector, a function, a pixel image (object of class "im") or a fitted point process model (object of class "ppm" or "lppm").
r	Optional. Numeric vector of values of the function argument r . There is a sensible default.
...	Arguments passed to <code>density.default</code> to control the smoothing.
correction	Geometry correction. Either "none" or "Ang". See Details.
normalise	Logical. If TRUE (the default), the denominator of the estimator is data-dependent (equal to the sum of the reciprocal intensities at the data points), which reduces the sampling variability. If FALSE, the denominator is the length of the network.

Details

This command computes the inhomogeneous version of the linear pair correlation function from point pattern data on a linear network.

If `lambda` = `NULL` the result is equivalent to the homogeneous pair correlation function `linearpctf`. If `lambda` is given, then it is expected to provide estimated values of the intensity of the point process at each point of `X`. The argument `lambda` may be a numeric vector (of length equal to the number of points in `X`), or a function(`x, y`) that will be evaluated at the points of `X` to yield numeric values, or a pixel image (object of class "im") or a fitted point process model (object of class "ppm" or "lppm").

If `correction="none"`, the calculations do not include any correction for the geometry of the linear network. If `correction="Ang"`, the pair counts are weighted using Ang's correction (Ang, 2010).

Value

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

Author(s)

Ang Qi Wei <aqw07398@hotmail.com> and Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

References

- Ang, Q.W. (2010) Statistical methodology for spatial point patterns on a linear network. MSc thesis, University of Western Australia.
- Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. To appear in *Scandinavian Journal of Statistics*.
- Okabe, A. and Yamada, I. (2001) The K-function method on a network and its computational implementation. *Geographical Analysis* **33**, 271-290.

See Also

[linearppcf](#), [linearKinhom](#), [lpp](#)

Examples

```
data(simplenet)
X <- rpoislpp(5, simplenet)
fit <- lppm(X, ~x)
K <- linearpccinhom(X, lambda=fit)
plot(K)
```

Linhom

L-function

Description

Calculates an estimate of the inhomogeneous version of the *L*-function (Besag's transformation of Ripley's *K*-function) for a spatial point pattern.

Usage

`Linhom(...)`

Arguments

... Arguments passed to [Kinhom](#) to estimate the inhomogeneous K-function.

Details

This command computes an estimate of the inhomogeneous version of the *L*-function for a spatial point pattern

The original *L*-function is a transformation (proposed by Besag) of Ripley's *K*-function,

$$L(r) = \sqrt{\frac{K(r)}{\pi}}$$

where $K(r)$ is the Ripley *K*-function of a spatially homogeneous point pattern, estimated by [Kest](#).

The inhomogeneous *L*-function is the corresponding transformation of the inhomogeneous *K*-function, estimated by [Kinhom](#). It is appropriate when the point pattern clearly does not have a homogeneous intensity of points. It was proposed by Baddeley, Moller and Waagepetersen (2000).

The command `Linhom` first calls `Kinhom` to compute the estimate of the inhomogeneous K-function, and then applies the square root transformation.

For a Poisson point pattern (homogeneous or inhomogeneous), the theoretical value of the inhomogeneous L -function is $L(r) = r$. The square root also has the effect of stabilising the variance of the estimator, so that L is more appropriate for use in simulation envelopes and hypothesis tests.

Value

An object of class "fv", see `fv.object`, which can be plotted directly using `plot.fv`.

Essentially a data frame containing columns

<code>r</code>	the vector of values of the argument r at which the function L has been estimated
<code>theo</code>	the theoretical value $L(r) = r$ for a stationary Poisson process

together with columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $L(r)$ obtained by the edge corrections named.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A., Moller, J. and Waagepetersen, R. (2000) Non- and semiparametric estimation of interaction in inhomogeneous point patterns. *Statistica Neerlandica* **54**, 329–350.

See Also

`Kest`, `Lest`, `Kinhom`, `pcf`

Examples

```
data(japanesepines)
X <- japanesepines
L <- Linhom(X, sigma=0.1)
plot(L, main="Inhomogeneous L function for Japanese Pines")
```

`linim`

Create Pixel Image on Linear Network

Description

Creates an object of class "linim" that represents a pixel image on a linear network.

Usage

```
linim(L, Z, ..., df=NULL)
```

Arguments

L	Linear network (object of class "linnet").
Z	Pixel image (object of class "im").
...	Ignored.
df	Advanced use only. Data frame giving full details of the mapping between the pixels of Z and the lines of L. See Details.

Details

This command creates an object of class "linim" that represents a pixel image defined on a linear network. Typically such objects are used to represent the result of smoothing or model-fitting on the network. Most users will not need to call `linim` directly.

The argument L is a linear network (object of class "linnet"). It gives the exact spatial locations of the line segments of the network, and their connectivity.

The argument Z is a pixel image object of class "im" that gives a pixellated approximation of the function values.

For increased efficiency, advanced users may specify the optional argument df. This is a data frame giving the precomputed mapping between the pixels of Z and the line segments of L. It should have columns named xc, yc containing the coordinates of the pixel centres, x, y containing the projections of these pixel centres onto the linear network, mapXY identifying the line segment on which each projected point lies, and tp giving the parametric position of (x, y) along the segment.

Value

Object of class "linim" that also inherits the class "im". There is a special method for plotting this class.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

References

- Ang, Q.W. (2010) *Statistical methodology for events on a network*. Master's thesis, School of Mathematics and Statistics, University of Western Australia.
- Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. To appear in *Scandinavian Journal of Statistics*.
- McSwiggan, G., Nair, M.G. and Baddeley, A. (2012) Fitting Poisson point process models to events on a linear network. Manuscript in preparation.

See Also

[plot.linim](#), [linnet](#), [im](#).

Examples

```
example(linnet)
M <- as.mask.psp(as.psp(letterA))
Z <- as.im(function(x,y) {x-y}, W=M)
X <- linim(letterA, Z)
X
```

linnet*Create a Linear Network***Description**

Creates an object of class "linnet" representing a network of line segments.

Usage

```
linnet(vertices, m, edges)
```

Arguments

<code>vertices</code>	Point pattern (object of class "ppp") specifying the vertices of the network.
<code>m</code>	Adjacency matrix. A matrix of logical values equal to TRUE when the corresponding vertices are joined by a line. (Specify either <code>m</code> or <code>edges</code> .)
<code>edges</code>	Edge list. A two-column matrix of integers, specifying all pairs of vertices that should be joined by an edge. (Specify either <code>m</code> or <code>edges</code> .)

Details

An object of class "linnet" represents a network of straight line segments in two dimensions. The function `linnet` creates such an object from the minimal information: the spatial location of each vertex (endpoint, crossing point or meeting point of lines) and information about which vertices are joined by an edge.

This function can take some time to execute, because the algorithm computes various properties of the network that are stored in the resulting object.

Value

Object of class "linnet" representing the linear network.

Author(s)

Ang Qi Wei <aqw07398@hotmail.com> and Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

See Also

[simpenet](#) for an example of a linear network.

[methods.linnet](#) for methods applicable to `linnet` objects.

[ppp](#), [psp](#).

Examples

```
# letter 'A' specified by adjacency matrix
v <- ppp(x=(-2):2, y=3*c(0,1,2,1,0), c(-3,3), c(-1,7))
m <- matrix(FALSE, 5,5)
for(i in 1:4) m[i,i+1] <- TRUE
m[2,4] <- TRUE
m <- m | t(m)
```

```

letterA <- linnet(v, m)
plot(letterA)

# letter 'A' specified by edge list
edg <- cbind(1:4, 2:5)
edg <- rbind(edg, c(2,4))
letterA <- linnet(v, edges=edg)

```

localK*Neighbourhood density function***Description**

Computes the neighbourhood density function, a local version of the K -function or L -function, defined by Getis and Franklin (1987).

Usage

```
localK(X, ..., correction = "Ripley", verbose = TRUE, rvalue=NULL)
localL(X, ..., correction = "Ripley", verbose = TRUE, rvalue=NULL)
```

Arguments

<code>X</code>	A point pattern (object of class "ppp").
<code>...</code>	Ignored.
<code>correction</code>	String specifying the edge correction to be applied. Options are "none", "translate", "Ripley", "isotropic" or "best". Only one correction may be specified.
<code>verbose</code>	Logical flag indicating whether to print progress reports during the calculation.
<code>rvalue</code>	Optional. A <i>single</i> value of the distance argument r at which the function L or K should be computed.

Details

The command `localL` computes the *neighbourhood density function*, a local version of the L -function (Besag's transformation of Ripley's K -function) that was proposed by Getis and Franklin (1987). The command `localK` computes the corresponding local analogue of the K -function.

Given a spatial point pattern X , the neighbourhood density function $L_i(r)$ associated with the i th point in X is computed by

$$L_i(r) = \sqrt{\frac{a}{(n-1)\pi} \sum_j e_{ij}}$$

where the sum is over all points $j \neq i$ that lie within a distance r of the i th point, a is the area of the observation window, n is the number of points in X , and e_{ij} is an edge correction term (as described in [Kest](#)). The value of $L_i(r)$ can also be interpreted as one of the summands that contributes to the global estimate of the L function.

By default, the function $L_i(r)$ or $K_i(r)$ is computed for a range of r values for each point i . The results are stored as a function value table (object of class "fv") with a column of the table containing the function estimates for each point of the pattern X .

Alternatively, if the argument `rvalue` is given, and it is a single number, then the function will only be computed for this value of r , and the results will be returned as a numeric vector, with one entry of the vector for each point of the pattern X .

Inhomogeneous counterparts of `localK` and `localL` are computed by `localKinhom` and `localLinhom`.

Value

If *rvalue* is given, the result is a numeric vector of length equal to the number of points in the point pattern.

If *rvalue* is absent, the result is an object of class "fv", see [fv.object](#), which can be plotted directly using [plot.fv](#). Essentially a data frame containing columns

<i>r</i>	the vector of values of the argument <i>r</i> at which the function <i>K</i> has been estimated
<i>theo</i>	the theoretical value $K(r) = \pi r^2$ or $L(r) = r$ for a stationary Poisson process

together with columns containing the values of the neighbourhood density function for each point in the pattern. Column *i* corresponds to the *i*th point. The last two columns contain the *r* and *theo* values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

Getis, A. and Franklin, J. (1987) Second-order neighbourhood analysis of mapped point patterns. *Ecology* **68**, 473–477.

See Also

[Kest](#), [Lest](#), [localKinhom](#), [localLinhom](#).

Examples

```

data(ponderosa)
X <- ponderosa

# compute all the local L functions
L <- localL(X)

# plot all the local L functions against r
plot(L, main="local L functions for ponderosa", legend=FALSE)

# plot only the local L function for point number 7
plot(L, iso007 ~ r)

# compute the values of L(r) for r = 12 metres
L12 <- localL(X, rvalue=12)

# Spatially interpolate the values of L12
# Compare Figure 5(b) of Getis and Franklin (1987)
X12 <- X %mark% L12
Z <- smooth.ppp(X12, sigma=5, dimyx=128)

plot(Z, col=topo.colors(128), main="smoothed neighbourhood density")
contour(Z, add=TRUE)
points(X, pch=16, cex=0.5)

```

localKinhom

Inhomogeneous Neighbourhood Density Function

Description

Computes spatially-weighted versions of the local K -function or L -function.

Usage

```
localKinhom(X, lambda, ...,
            correction = "Ripley", verbose = TRUE, rvalue=NULL,
            sigma = NULL, varcov = NULL)
localLinhom(X, lambda, ...,
            correction = "Ripley", verbose = TRUE, rvalue=NULL,
            sigma = NULL, varcov = NULL)
```

Arguments

X	A point pattern (object of class "ppp").
lambda	Optional. Values of the estimated intensity function. Either a vector giving the intensity values at the points of the pattern X, a pixel image (object of class "im") giving the intensity values at all locations, a fitted point process model (object of class "ppm") or a function(x,y) which can be evaluated to give the intensity value at any location.
...	Extra arguments. Ignored if lambda is present. Passed to density.ppp if lambda is omitted.
correction	String specifying the edge correction to be applied. Options are "none", "translate", "Ripley", "isotropic" or "best". Only one correction may be specified.
verbose	Logical flag indicating whether to print progress reports during the calculation.
rvalue	Optional. A <i>single</i> value of the distance argument r at which the function L or K should be computed.
sigma, varcov	Optional arguments passed to density.ppp to control the kernel smoothing procedure for estimating lambda, if lambda is missing.

Details

The functions `localKinhom` and `localLinhom` are inhomogeneous or weighted versions of the neighbourhood density function implemented in `localK` and `localL`.

Given a spatial point pattern X , the inhomogeneous neighbourhood density function $L_i(r)$ associated with the i th point in X is computed by

$$L_i(r) = \sqrt{\frac{1}{\pi} \sum_j \frac{e_{ij}}{\lambda_j}}$$

where the sum is over all points $j \neq i$ that lie within a distance r of the i th point, λ_j is the estimated intensity of the point pattern at the point j , and e_{ij} is an edge correction term (as described in [Kest](#)). The value of $L_i(r)$ can also be interpreted as one of the summands that contributes to the global estimate of the inhomogeneous L function (see [Linhom](#)).

By default, the function $L_i(r)$ or $K_i(r)$ is computed for a range of r values for each point i . The results are stored as a function value table (object of class "fv") with a column of the table containing the function estimates for each point of the pattern X .

Alternatively, if the argument `rvalue` is given, and it is a single number, then the function will only be computed for this value of r , and the results will be returned as a numeric vector, with one entry of the vector for each point of the pattern X .

Value

If `rvalue` is given, the result is a numeric vector of length equal to the number of points in the point pattern.

If `rvalue` is absent, the result is an object of class "fv", see `fv.object`, which can be plotted directly using `plot.fv`. Essentially a data frame containing columns

<code>r</code>	the vector of values of the argument r at which the function K has been estimated
<code>theo</code>	the theoretical value $K(r) = \pi r^2$ or $L(r) = r$ for a stationary Poisson process

together with columns containing the values of the neighbourhood density function for each point in the pattern. Column i corresponds to the i th point. The last two columns contain the `r` and `theo` values.

Author(s)

Mike Kuhn, Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`Kinhom`, `Linhom`, `localK`, `localL`.

Examples

```

data(ponderosa)
X <- ponderosa

# compute all the local L functions
L <- localLinhom(X)

# plot all the local L functions against r
plot(L, main="local L functions for ponderosa", legend=FALSE)

# plot only the local L function for point number 7
plot(L, iso007 ~ r)

# compute the values of L(r) for r = 12 metres
L12 <- localL(X, rvalue=12)

```

localpcf	<i>Local pair correlation function</i>
----------	--

Description

Computes individual contributions to the pair correlation function from each data point.

Usage

```
localpcf(X, ..., delta=NULL, rmax=NULL, nr=512, stoyan=0.15)
localpcfinhom(X, ..., delta=NULL, rmax=NULL, nr=512, stoyan=0.15,
              lambda=NULL, sigma=NULL, varcov=NULL)
```

Arguments

X	A point pattern (object of class "ppp").
delta	Smoothing bandwidth for pair correlation. The halfwidth of the Epanechnikov kernel.
rmax	Optional. Maximum value of distance r for which pair correlation values $g(r)$ should be computed.
nr	Optional. Number of values of distance r for which pair correlation $g(r)$ should be computed.
stoyan	Optional. The value of the constant c in Stoyan's rule of thumb for selecting the smoothing bandwidth delta.
lambda	Optional. Values of the estimated intensity function, for the inhomogeneous pair correlation. Either a vector giving the intensity values at the points of the pattern X, a pixel image (object of class "im") giving the intensity values at all locations, a fitted point process model (object of class "ppm") or a function(x,y) which can be evaluated to give the intensity value at any location.
sigma, varcov, ...	These arguments are ignored by localpcf but are passed by localpcfinhom (when lambda=NULL) to the function density.ppp to control the kernel smoothing estimation of lambda.

Details

localpcf computes the contribution, from each individual data point in a point pattern X, to the empirical pair correlation function of X. These contributions are sometimes known as LISA (local indicator of spatial association) functions based on pair correlation.

localpcfinhom computes the corresponding contribution to the *inhomogeneous* empirical pair correlation function of X.

Given a spatial point pattern X, the local pcf $g_i(r)$ associated with the i th point in X is computed by

$$g_i(r) = \frac{a}{2\pi n} \sum_j k(d_{i,j} - r)$$

where the sum is over all points $j \neq i$, a is the area of the observation window, n is the number of points in X, and d_{ij} is the distance between points i and j. Here k is the Epanechnikov kernel,

$$k(t) = \frac{3}{4\delta} \max(0, 1 - \frac{t^2}{\delta^2}).$$

Edge correction is performed using the border method (for the sake of computational efficiency): the estimate $g_i(r)$ is set to NA if $r > b_i$, where b_i is the distance from point i to the boundary of the observation window.

The smoothing bandwidth δ may be specified. If not, it is chosen by Stoyan's rule of thumb $\delta = c/\hat{\lambda}$ where $\hat{\lambda} = n/a$ is the estimated intensity and c is a constant, usually taken to be 0.15. The value of c is controlled by the argument `stoyan`.

For `localpcfinhom`, the optional argument `lambda` specifies the values of the estimated intensity function. If `lambda` is given, it should be either a numeric vector giving the intensity values at the points of the pattern X , a pixel image (object of class "im") giving the intensity values at all locations, a fitted point process model (object of class "ppm") or a function(x, y) which can be evaluated to give the intensity value at any location. If `lambda` is not given, then it will be estimated using a leave-one-out kernel density smoother as described in `pcfinhom`.

Value

An object of class "fv", see [fv.object](#), which can be plotted directly using [plot.fv](#). Essentially a data frame containing columns

<code>r</code>	the vector of values of the argument r at which the function K has been estimated
<code>theo</code>	the theoretical value $K(r) = \pi r^2$ or $L(r) = r$ for a stationary Poisson process

together with columns containing the values of the local pair correlation function for each point in the pattern. Column `i` corresponds to the `i`th point. The last two columns contain the `r` and `theo` values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[localK](#), [localKinhom](#), [pcf](#), [pcfinhom](#)

Examples

```
data(ponderosa)
X <- ponderosa

g <- localpcf(X, stoyan=0.5)
colo <- c(rep("grey", npoints(X)), "blue")
a <- plot(g, main=c("local pair correlation functions", "Ponderosa pines"),
          legend=FALSE, col=colo, lty=1)

# plot only the local pair correlation function for point number 7
plot(g, est007 ~ r)

gi <- localpcfinhom(X, stoyan=0.5)
a <- plot(gi, main=c("inhomogeneous local pair correlation functions",
                     "Ponderosa pines"),
           legend=FALSE, col=colo, lty=1)
```

logLik.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 for a fitted Gibbs point process model.

Usage

```
## S3 method for class 'ppm'
logLik(object, ..., warn=TRUE)
## S3 method for class 'ppm'
extractAIC(fit, scale=0, k=2, ...)
## 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 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.

Details

These functions are methods for the generic commands [logLik](#), [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

$$AIC = -2 \log(L) + k \times edf$$

where L is the maximised likelihood of the fitted model, and edf is 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 (if `fit` is a Gibbs model).

The method [nobs.ppm](#) returns the number of points in the original data point pattern to which the model was fitted.

The R functions [AIC](#) and [step](#) use these methods.

Value

A numerical value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

`ppm`, `as.owin`, `coef.ppm`, `fitted.ppm`, `formula.ppm`, `model.frame.ppm`, `model.matrix.ppm`,
`plot.ppm`, `predict.ppm`, `residuals.ppm`, `simulate.ppm`, `summary.ppm`, `terms.ppm`, `update.ppm`,
`vcov.ppm`.

Examples

```
data(cells)
fit <- ppm(cells, ~x)
nobs(fit)
logLik(fit)
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@maths.uwa.edu.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[slrm](#)

Examples

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

lohboot

Bootstrap Confidence Bands for Summary Function

Description

Computes a bootstrap confidence band for a summary function of a point process.

Usage

```
lohboot(X,
        fun=c("pcf", "Kest", "pcfinhom", "Kinhom"),
        ..., nsim=200, confidence=0.95, type=7)
```

Arguments

X	A point pattern (object of class "ppp").
fun	Name of the summary function to be computed: one of the strings "pcf", "Kest", "pcfinhom" or "Kinhom".
...	Arguments passed to the corresponding local version of the summary function (see Details).
nsim	Number of bootstrap simulations.
confidence	Confidence level, as a fraction between 0 and 1.
type	Integer. Argument passed to quantile controlling the way the quantiles are calculated.

Details

This algorithm computes confidence bands for the true value of the summary statistic `fun` using the bootstrap method of Loh (2008).

If `fun="pcf"`, for example, the algorithm computes a pointwise $100 * \alpha$ percent confidence interval for the true value of the pair correlation function `pcf` for the point process. It starts by computing the array of *local* pair correlation functions, `localpcf`, of the data pattern `X`. This array consists of the contributions to `pcf` from each data point. Then these contributions are resampled `nsim` times with replacement; from each resampled dataset the total contribution is computed, yielding `nsim` random pair correlation functions. The pointwise $\alpha/2$ and $1 - \alpha/2$ quantiles of these functions are computed.

To control the smoothing and estimation algorithm, use the arguments `...`, which are passed to the local version of the summary function, as shown below:

fun	local version
<code>pcf</code>	<code>localpcf</code>
<code>Kest</code>	<code>localK</code>
<code>pcfinhom</code>	<code>localpcfinhom</code>
<code>Kinhom</code>	<code>localKinhom</code>

An alternative to `lohboot` is `varblock`.

Value

A function value table (object of class "fv") containing columns giving the estimate of the summary function, the upper and lower limits of the bootstrap confidence interval, and the theoretical value of the summary function for a Poisson process.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

Loh, J.M. (2008) A valid and fast spatial bootstrap for correlation functions. *The Astrophysical Journal*, **681**, 726–734.

See Also

Summary functions `Kest`, `pcf`, `Kinhom`, `pcfinhom`, `localK`, `localpcf`, `localKinhom`, `localpcfinhom`.

See `varblock` for an alternative bootstrap technique.

Examples

```
p <- lohboot(simdat, stoyan=0.5)
plot(p, shade=c("lo", "hi"))
```

longleaf*Longleaf Pines Point Pattern*

Description

Locations and sizes of Longleaf pine trees. A marked point pattern.

The data record the locations and diameters of 584 Longleaf pine (*Pinus palustris*) trees in a 200 x 200 metre region in southern Georgia (USA). They were collected and analysed by Platt, Evans and Rathbun (1988).

This is a marked point pattern; the mark associated with a tree is its diameter at breast height (dbh), a convenient measure of its size. Several analyses have considered only the “adult” trees which are conventionally defined as those trees with dbh greater than or equal to 30 cm.

The pattern is regarded as spatially inhomogeneous.

Usage

```
data(longleaf)
```

Format

An object of class "ppp" representing the point pattern of tree locations. Entries include

x	Cartesian <i>x</i> -coordinate of tree
y	Cartesian <i>y</i> -coordinate of tree
marks	diameter at breast height, in centimetres.

See [ppp.object](#) for details of the format of a point pattern object.

Source

Platt, Evans and Rathbun (1988)

References

Platt, W. J., Evans, G. W. and Rathbun, S. L. (1988) The population dynamics of a long-lived Conifer (*Pinus palustris*). *The American Naturalist* **131**, 491–525.

Rathbun, S. L. and Cressie, N. (1994) A space-time survival point process for a longleaf pine forest in southern Georgia. *Journal of the American Statistical Association* **89**, 1164–1173.

Examples

```
data(longleaf)
plot(longleaf)
plot(cut(longleaf, breaks=c(0,30,Inf), labels=c("Sapling","Adult")))
```

lpp*Create Point Pattern on Linear Network***Description**

Creates an object of class "lpp" that represents a point pattern on a linear network.

Usage

```
lpp(X, L)
```

Arguments

- | | |
|----------------|---|
| <code>X</code> | Locations of the points. A matrix or data frame of coordinates, or a point pattern object (of class "ppp") or other data acceptable to as.ppp . |
| <code>L</code> | Linear network (object of class "linnet"). |

Details

This command creates an object of class "lpp" that represents a point pattern on a linear network.

Normally `X` is a point pattern. The points of `X` should lie on the lines of `L`.

Alternatively `X` may be a matrix or data frame containing at least two columns.

- Usually the first two columns of `X` will be interpreted as spatial coordinates, and any remaining columns as marks.
- The exception occurs if `X` is a data frame with columns named `x`, `y`, `seg` and `tp`. Then `x` and `y` will be interpreted as spatial coordinates, and `seg` and `tp` as local coordinates, with `seg` indicating which line segment of `L` the point lies on, and `tp` indicating how far along the segment the point lies (normalised to 1). Any remaining columns will be interpreted as marks.

Value

An object of class "lpp". Also inherits the class "ppx".

Note on changed format

The internal format of "lpp" objects was changed in **spatstat** version 1.28-0. Objects in the old format are still handled correctly, but computations are faster in the new format. To convert an object `X` from the old format to the new format, use `X <- lpp(as.ppp(X), as.linnet(X))`.

Author(s)

Ang Qi Wei <aqw07398@hotmail.com> and Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

See Also

[methods.lpp](#) and [methods.ppx](#) for methods applicable to `lpp` objects.

[linnet](#) for linear networks.

Random point patterns on a linear network can be generated by [rpoislpp](#) or [runiflpp](#).

Examples

```
example(linnet)
xx <- list(x=c(-1.5,0,0.5,1.5), y=c(1.5,3,4.5,1.5))
X <- lpp(xx, letterA)
plot(X)
X
summary(X)
```

lppm

Fit Point Process Model to Point Pattern on Linear Network

Description

Fit a point process model to a point pattern dataset on a linear network

Usage

```
lppm(X, ...)
```

Arguments

X	Object of class "lpp" specifying a point pattern on a linear network.
...	Arguments passed to ppm .

Details

This function fits a point process model to data that specify a point pattern on a linear network. It is a counterpart of the model-fitting function [ppm](#) designed to work with objects of class "lpp" instead of "ppp".

The argument X should be an object of class "lpp" (created by the command [lpp](#)) specifying a point pattern on a linear network.

The subsequent arguments ... will be passed to [ppm](#). They specify the form of the model.

Value

An object of class "lppm" representing the fitted model. There are methods for [print](#), [predict](#), [coef](#) and similar functions.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

References

- Ang, Q.W. (2010) *Statistical methodology for events on a network*. Master's thesis, School of Mathematics and Statistics, University of Western Australia.
- Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. To appear in *Scandinavian Journal of Statistics*.
- McSwiggan, G., Nair, M.G. and Baddeley, A. (2012) Fitting Poisson point process models to events on a linear network. Manuscript in preparation.

See Also

[methods.lppm](#), [predict.lppm](#), [ppm](#), [lpp](#).

Examples

```
example(lpp)
lppm(X, ~1)
```

lurking

Lurking variable plot

Description

Plot spatial point process residuals against a covariate

Usage

```
lurking(object, covariate, type="eem",
        cumulative=TRUE,
        clipwindow=default.clipwindow(object),
        rv,
        plot.sd, plot.it=TRUE,
        typename,
        covname,
        oldstyle=FALSE, check=TRUE,
        ...,
        splineargs=list(spar=0.5))
```

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 <i>Details</i> 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).
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 .
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 <i>Details</i> .

<code>plot.it</code>	Logical value indicating whether plots should be shown. If <code>plot.it=FALSE</code> , only the computed coordinates for the plots are returned. See <code>Value</code> .
<code>typename</code>	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.
<code>covname</code>	A string name for the covariate, to be used in axis labels of plots.
<code>oldstyle</code>	Logical flag indicating whether error bounds should be plotted using the approximation given in the original paper (<code>oldstyle=TRUE</code>), or using the correct asymptotic formula (<code>oldstyle=FALSE</code>).
<code>check</code>	Logical flag indicating whether the integrity of the data structure in <code>object</code> should be checked.
<code>...</code>	Arguments passed to <code>plot.default</code> and <code>lines</code> to control the plot behaviour.
<code>splineargs</code>	A list of arguments passed to <code>smooth.spline</code> for the estimation of the derivatives in the case <code>cumulative=FALSE</code> .

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.

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 (dotted lines).

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 $E(H(z)) = \text{area of the subset of the window satisfying } Z(u) \leq z$.

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

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

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Baddeley, A., Turner, R., Moller, J. and Hazelton, M. (2005) Residual analysis for spatial point processes. *Journal of the Royal Statistical Society, Series B* **67**, 617–666.
- Baddeley, A., Moller, J. and Pakes, A.G. (2006) Properties of residuals for spatial point processes. To appear.
- 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

```
data(nztrees)
lurking(nztrees, expression(x))
fit <- ppm(nztrees, ~x, Poisson())
lurking(fit, expression(x))
lurking(fit, expression(x), cumulative=FALSE)
```

lut

Lookup Tables

Description

Create a lookup table.

Usage

```
lut(outputs, ..., range=NULL, breaks=NULL, inputs=NULL)
```

Arguments

outputs	Vector of output values
...	Ignored.
range	Interval of numbers to be mapped. A numeric vector of length 2, specifying the ends of the range of values to be mapped. Incompatible with breaks or inputs.
inputs	Input values to which the output values are associated. A factor or vector of the same length as outputs. Incompatible with breaks or range.
breaks	Breakpoints for the lookup table. A numeric vector of length equal to <code>length(outputs)+1</code> . Incompatible with range or inputs.

Details

A lookup table is a function, mapping input values to output values.

The command `lut` creates an object representing a lookup table, which can then be used to control various behaviour in the `spatstat` package. It can also be used to compute the output value assigned to any input value.

The argument `outputs` specifies the output values to which input data values will be mapped. It should be a vector of any atomic type (e.g. numeric, logical, character, complex) or factor values.

Exactly one of the arguments `range`, `inputs` or `breaks` must be specified by name.

If `inputs` is given, then it should be a vector or factor, of the same length as `outputs`. The entries of `inputs` can be any atomic type (e.g. numeric, logical, character, complex) or factor values. The resulting lookup table associates the value `inputs[i]` with the value `outputs[i]`.

If `range` is given, then it determines the interval of the real number line that will be mapped. It should be a numeric vector of length 2.

If `breaks` is given, then it determines intervals of the real number line which are mapped to each output value. It should be a numeric vector, of length at least 2, with entries that are in increasing order. Infinite values are allowed. Any number in the range between `breaks[i]` and `breaks[i+1]` will be mapped to the value `outputs[i]`.

The result is an object of class "lut". There is a print method for this class. Some plot commands in the **spatstat** package accept an object of this class as a specification of a lookup table.

The result is also a function f which can be used to compute the output value assigned to any input data value. That is, f(x) returns the output value assigned to x. This also works for vectors of input data values.

Value

A function, which is also an object of class "lut".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[colourmap](#).

Examples

```
# lookup table for real numbers, using breakpoints
cr <- lut(factor(c("low", "medium", "high")), breaks=c(0,5,10,15))
cr
cr(3.2)
cr(c(3,5,7))
# lookup table for discrete set of values
ct <- lut(c(0,1), inputs=c(FALSE, TRUE))
ct(TRUE)
```

Description

Estimate the marked connection function of a multitype point pattern.

Usage

```
markconnect(X, i, j, r=NULL,
            correction=c("isotropic", "Ripley", "translate"),
            method="density", ..., normalise=FALSE)
```

Arguments

- | | |
|---|--|
| X | The observed point pattern. An object of class "ppp" or something acceptable to as.ppp . |
| i | Number or character string identifying the type (mark value) of the points in X from which distances are measured. |
| j | Number or character string identifying the type (mark value) of the points in X to which distances are measured. |

<code>r</code>	numeric vector. The values of the argument r at which the mark connection function $p_{ij}(r)$ should be evaluated. There is a sensible default.
<code>correction</code>	A character vector containing any selection of the options "isotropic", "Ripley" or "translate". It specifies the edge correction(s) to be applied.
<code>method</code>	A character vector indicating the user's choice of density estimation technique to be used. Options are "density", "loess", "sm" and "smrep".
<code>...</code>	Arguments passed to the density estimation routine (density , loess or sm.density) selected by <code>method</code> .
<code>normalise</code>	If TRUE, normalise the pair connection function by dividing it by $p_i p_j$, the estimated probability that randomly-selected points will have marks i and j .

Details

The mark connection function $p_{ij}(r)$ of a multitype point process X is a measure of the dependence between the types of two points of the process a distance r apart.

Informally $p_{ij}(r)$ is defined as the conditional probability, given that there is a point of the process at a location u and another point of the process at a location v separated by a distance $\|u - v\| = r$, that the first point is of type i and the second point is of type j . See Stoyan and Stoyan (1994).

If the marks attached to the points of X are independent and identically distributed, then $p_{ij}(r) \equiv p_i p_j$ where p_i denotes the probability that a point is of type i . Values larger than this, $p_{ij}(r) > p_i p_j$, indicate positive association between the two types, while smaller values indicate negative association.

The argument `X` must be a point pattern (object of class "ppp") or any data that are acceptable to [as.ppp](#). It must be a multitype point pattern (a marked point pattern with factor-valued marks).

The argument `r` is the vector of values for the distance r at which $p_{ij}(r)$ is estimated. There is a sensible default.

This algorithm assumes that `X` can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in `X` as `X>window`) may have arbitrary shape.

Biases due to edge effects are treated in the same manner as in [Kest](#). The edge corrections implemented here are

isotropic/Ripley Ripley's isotropic correction (see Ripley, 1988; Ohser, 1983). This is implemented only for rectangular and polygonal windows (not for binary masks).

translate Translation correction (Ohser, 1983). Implemented for all window geometries, but slow for complex windows.

Note that the estimator assumes the process is stationary (spatially homogeneous).

The mark connection function is estimated using density estimation techniques. The user can choose between

"density" which uses the standard kernel density estimation routine [density](#), and works only for evenly-spaced `r` values;

"loess" which uses the function [loess](#) in the package [modreg](#);

"sm" which uses the function [sm.density](#) in the package [sm](#) and is extremely slow;

"smrep" which uses the function [sm.density](#) in the package [sm](#) and is relatively fast, but may require manual control of the smoothing parameter `hmult`.

Value

An object of class "fv" (see [fv.object](#)).

Essentially a data frame containing numeric columns

r the values of the argument r at which the mark connection function $p_{ij}(r)$ has been estimated

theo the theoretical value of $p_{ij}(r)$ when the marks attached to different points are independent

together with a column or columns named "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $p_{ij}(r)$ obtained by the edge corrections named.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

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

See Also

Multitype pair correlation [pcfcross](#) and multitype K-functions [Kcross](#), [Kdot](#).

Use [alltypes](#) to compute the mark connection functions between all pairs of types.

Mark correlation [markcorr](#) and mark variogram [markvario](#) for numeric-valued marks.

Examples

```
# Hughes' amacrine data
# Cells marked as 'on'/'off'
data(amacrine)
M <- markconnect(amacrine, "on", "off")
plot(M)

# Compute for all pairs of types at once
plot(alltypes(amacrine, markconnect))
```

markcorr

Mark Correlation Function

Description

Estimate the marked correlation function of a marked point pattern.

Usage

```
markcorr(X, f = function(m1, m2) { m1 * m2}, r=NULL,
         correction=c("isotropic", "Ripley", "translate"),
         method="density", ...,
         f1=NULL, normalise=TRUE, fargs=NULL)
```

Arguments

<code>X</code>	The observed point pattern. An object of class "ppp" or something acceptable to <code>as.ppp</code> .
<code>f</code>	Optional. Test function f used in the definition of the mark correlation function. An R function with at least two arguments. There is a sensible default.
<code>r</code>	Optional. Numeric vector. The values of the argument r at which the mark correlation function $k_f(r)$ should be evaluated. There is a sensible default.
<code>correction</code>	A character vector containing any selection of the options "isotropic", "Ripley", "translate", "none" or "best". It specifies the edge correction(s) to be applied.
<code>method</code>	A character vector indicating the user's choice of density estimation technique to be used. Options are "density", "loess", "sm" and "smrep".
<code>...</code>	Arguments passed to the density estimation routine (<code>density</code> , <code>loess</code> or <code>sm.density</code>) selected by <code>method</code> .
<code>f1</code>	An alternative to <code>f</code> . If this argument is given, then f is assumed to take the form $f(u, v) = f_1(u)f_1(v)$.
<code>normalise</code>	If <code>normalise=FALSE</code> , compute only the numerator of the expression for the mark correlation.
<code>fargs</code>	Optional. A list of extra arguments to be passed to the function <code>f</code> or <code>f1</code> .

Details

By default, this command calculates an estimate of Stoyan's mark correlation $k_{mm}(r)$ for the point pattern.

Alternatively if the argument `f` or `f1` is given, then it calculates Stoyan's generalised mark correlation $k_f(r)$ with test function f .

Theoretical definitions are as follows (see Stoyan and Stoyan (1994, p. 262)):

- For a point process X with numeric marks, Stoyan's mark correlation function $k_{mm}(r)$, is

$$k_{mm}(r) = \frac{E_{0u}[M(0)M(u)]}{E[M, M']}$$

where E_{0u} denotes the conditional expectation given that there are points of the process at the locations 0 and u separated by a distance r , and where $M(0), M(u)$ denote the marks attached to these two points. On the denominator, M, M' are random marks drawn independently from the marginal distribution of marks, and E is the usual expectation.

- For a multitype point process X , the mark correlation is

$$k_{mm}(r) = \frac{P_{0u}[M(0)M(u)]}{P[M = M']}$$

where P and P_{0u} denote the probability and conditional probability.

- The *generalised* mark correlation function $k_f(r)$ of a marked point process X , with test function f , is

$$k_f(r) = \frac{E_{0u}[f(M(0), M(u))]}{E[f(M, M')]} \quad \text{Equation 1}$$

The test function f is any function $f(m_1, m_2)$ with two arguments which are possible marks of the pattern, and which returns a nonnegative real value. Common choices of f are: for continuous nonnegative real-valued marks,

$$f(m_1, m_2) = m_1 m_2$$

for discrete marks (multitype point patterns),

$$f(m_1, m_2) = 1(m_1 = m_2)$$

and for marks taking values in $[0, 2\pi]$,

$$f(m_1, m_2) = \sin(m_1 - m_2)$$

Note that $k_f(r)$ is not a “correlation” in the usual statistical sense. It can take any nonnegative real value. The value 1 suggests “lack of correlation”: if the marks attached to the points of X are independent and identically distributed, then $k_f(r) \equiv 1$. The interpretation of values larger or smaller than 1 depends on the choice of function f .

The argument X must be a point pattern (object of class "ppp") or any data that are acceptable to [as.ppp](#). It must be a marked point pattern.

The argument f determines the function to be applied to pairs of marks. It has a sensible default, which depends on the kind of marks in X . If the marks are numeric values, then $f <- \text{function}(m1, m2) \{ m1 * m2 \}$ computes the product of two marks. If the marks are a factor (i.e. if X is a multitype point pattern) then $f <- \text{function}(m1, m2) \{ m1 == m2 \}$ yields the value 1 when the two marks are equal, and 0 when they are unequal. These are the conventional definitions for numerical marks and multitype points respectively.

The argument f may be specified by the user. It must be an R function, accepting two arguments $m1$ and $m2$ which are vectors of equal length containing mark values (of the same type as the marks of X). (It may also take additional arguments, passed through $fargs$). It must return a vector of numeric values of the same length as $m1$ and $m2$. The values must be non-negative, and NA values are not permitted.

Alternatively the user may specify the argument $f1$ instead of f . This indicates that the test function f should take the form $f(u, v) = f1(u)f1(v)$ where $f1(u)$ is given by the argument $f1$. The argument $f1$ should be an R function with at least one argument. (It may also take additional arguments, passed through $fargs$).

The argument r is the vector of values for the distance r at which $k_f(r)$ is estimated.

This algorithm assumes that X can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in X as $X>window$) may have arbitrary shape.

Biases due to edge effects are treated in the same manner as in [Kest](#). The edge corrections implemented here are

isotropic/Ripley Ripley’s isotropic correction (see Ripley, 1988; Ohser, 1983). This is implemented only for rectangular and polygonal windows (not for binary masks).

translate Translation correction (Ohser, 1983). Implemented for all window geometries, but slow for complex windows.

Note that the estimator assumes the process is stationary (spatially homogeneous).

The numerator and denominator of the mark correlation function (in the expression above) are estimated using density estimation techniques. The user can choose between

“density” which uses the standard kernel density estimation routine [density](#), and works only for evenly-spaced r values;

"loess" which uses the function `loess` in the package **modreg**;
 "sm" which uses the function `sm.density` in the package **sm** and is extremely slow;
 "smrep" which uses the function `sm.density` in the package **sm** and is relatively fast, but may require manual control of the smoothing parameter `hmult`.

If `normalise=FALSE` then the algorithm will compute only the numerator

$$c_f(r) = E_{0u} f(M(0), M(u))$$

of the expression for the mark correlation function.

Value

A function value table (object of class "fv") or a list of function value tables, one for each column of marks.

An object of class "fv" (see [fv.object](#)) is essentially a data frame containing numeric columns

<code>r</code>	the values of the argument r at which the mark correlation function $k_f(r)$ has been estimated
<code>theo</code>	the theoretical value of $k_f(r)$ when the marks attached to different points are independent, namely 1

together with a column or columns named "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the mark correlation function $k_f(r)$ obtained by the edge corrections named.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

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

See Also

Mark variogram [markvario](#) for numeric marks.

Mark connection function [markconnect](#) and multitype K-functions [Kcross](#), [Kdot](#) for factor-valued marks.

[markcorrint](#) to estimate the indefinite integral of the mark correlation function.

Examples

```
# CONTINUOUS-VALUED MARKS:
# (1) Spruces
# marks represent tree diameter
data(spruces)
# mark correlation function
ms <- markcorr(spruces)
plot(ms)

# (2) simulated data with independent marks
```

```

X <- rpoispp(100)
X <- X %mark% runif(X$n)
## Not run:
Xc <- markcorr(X)
plot(Xc)

## End(Not run)

# MULTITYPE DATA:
# Hughes' amacrine data
# Cells marked as 'on'/'off'
data(amacrine)
# (3) Kernel density estimate with Epanechnikov kernel
# (as proposed by Stoyan & Stoyan)
M <- markcorr(amacrine, function(m1,m2) {m1==m2},
               correction="translate", method="density",
               kernel="epanechnikov")
plot(M)
# Note: kernel="epanechnikov" comes from help(density)

# (4) Same again with explicit control over bandwidth
## Not run:
M <- markcorr(amacrine,
               correction="translate", method="density",
               kernel="epanechnikov", bw=0.02)
# see help(density) for correct interpretation of 'bw'

## End(Not run)

```

markcorrint*Mark Correlation Integral***Description**

Estimates the mark correlation integral of a marked point pattern.

Usage

```
markcorrint(X, f = NULL, r = NULL,
            correction = c("isotropic", "Ripley", "translate"), ...,
            f1 = NULL, normalise = TRUE, returnL = FALSE, fargs = NULL)
```

Arguments

- | | |
|----------------|--|
| <code>X</code> | The observed point pattern. An object of class "ppp" or something acceptable to as.ppp . |
| <code>f</code> | Optional. Test function f used in the definition of the mark correlation function. An R function with at least two arguments. There is a sensible default. |
| <code>r</code> | Optional. Numeric vector. The values of the argument r at which the mark correlation function $k_f(r)$ should be evaluated. There is a sensible default. |

correction	A character vector containing any selection of the options "isotropic", "Ripley" or "translate". It specifies the edge correction(s) to be applied.
...	Ignored.
f1	An alternative to f. If this argument is given, then f is assumed to take the form $f(u, v) = f_1(u)f_1(v)$.
normalise	If normalise=FALSE, compute only the numerator of the expression for the mark correlation.
returnL	Compute the analogue of the K-function if returnL=FALSE or the analogue of the L-function if returnL=TRUE.
fargs	Optional. A list of extra arguments to be passed to the function f or f1.

Details

Given a marked point pattern X, this command estimates the weighted indefinite integral

$$K_f(r) = 2\pi \int_0^r sk_f(s)ds$$

of the mark correlation function $k_f(r)$. See [markcorr](#) for a definition of the mark correlation function.

The use of the weighted indefinite integral was advocated by Penttinen et al (1992). The relationship between K_f and k_f is analogous to the relationship between the classical K-function $K(r)$ and the pair correlation function $g(r)$.

If returnL=FALSE then the function $K_f(r)$ is returned; otherwise the function

$$L_f(r) = \sqrt{K_f(r)/pi}$$

is returned.

Value

An object of class "fv" (see [fv.object](#)).

Essentially a data frame containing numeric columns

r	the values of the argument r at which the mark correlation integral $K_f(r)$ has been estimated
theo	the theoretical value of $K_f(r)$ when the marks attached to different points are independent, namely πr^2

together with a column or columns named "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the mark correlation integral $K_f(r)$ obtained by the edge corrections named (if returnL=FALSE).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Penttinen, A., Stoyan, D. and Henttonen, H. M. (1992) Marked point processes in forest statistics. *Forest Science* **38** (1992) 806-824.
 Illian, J., Penttinen, A., Stoyan, H. and Stoyan, D. (2008) *Statistical analysis and modelling of spatial point patterns*. Chichester: John Wiley.

See Also

[markcorr](#) to estimate the mark correlation function.

Examples

```
# CONTINUOUS-VALUED MARKS:
# (1) Spruces
# marks represent tree diameter
data(spruces)
# mark correlation function
ms <- markcorrint(spruces)
plot(ms)

# (2) simulated data with independent marks
X <- rpoispp(100)
X <- X %mark% runif(X$n)
Xc <- markcorrint(X)
plot(Xc)

# MULTITYPE DATA:
# Hughes' amacrine data
# Cells marked as 'on'/'off'
data(amacrine)
M <- markcorrint(amacrine, function(m1,m2) {m1==m2},
                  correction="translate")
plot(M)
```

marks

*Marks of a Point Pattern***Description**

Extract or change the marks attached to a point pattern dataset.

Usage

```
marks(x, ...)
## S3 method for class 'ppp'
marks(x, ..., dfok=TRUE)
## S3 method for class 'ppx'
marks(x, ..., drop=TRUE)
marks(x, ...) <- value
## S3 replacement method for class 'ppp'
marks(x, ..., dfok=TRUE) <- value
## S3 replacement method for class 'ppx'
marks(x, ...) <- value
setmarks(x, value)
x %mark% value
```

Arguments

x	Point pattern dataset (object of class "ppp" or "ppx").
...	Ignored.
dfok	Logical. If FALSE, data frames of marks are not permitted and will generate an error.
drop	Logical. If TRUE, a data frame consisting of a single column of marks will be converted to a vector or factor.
value	Vector, data frame or hyperframe of mark values, or NULL.

Details

These functions extract or change the marks attached to the points of the point pattern x.

The expression `marks(x)` extracts the marks of x. The assignment `marks(x) <- value` assigns new marks to the dataset x, and updates the dataset x in the current environment. The expression `setmarks(x,value)` or equivalently `x %mark% value` returns a point pattern obtained by replacing the marks of x by value, but does not change the dataset x itself.

For point patterns in two-dimensional space (objects of class "ppp") the marks can be a vector, a factor, or a data frame.

For general point patterns (objects of class "ppx") the marks can be a vector, a factor, a data frame or a hyperframe.

For the assignment `marks(x) <- value`, the value should be a vector or factor of length equal to the number of points in x, or a data frame or hyperframe with as many rows as there are points in x. If value is a single value, or a data frame or hyperframe with one row, then it will be replicated so that the same marks will be attached to each point.

To remove marks, use `marks(x) <- NULL` or `unmark(x)`.

Use `ppp` or `ppx` to create point patterns in more general situations.

Value

For `marks(x)`, the result is a vector, factor, data frame or hyperframe, containing the mark values attached to the points of x.

For `marks(x) <- value`, the result is the updated point pattern x (with the side-effect that the dataset x is updated in the current environment).

For `setmarks(x,value)` and `x %mark% value`, the return value is the point pattern obtained by replacing the marks of x by value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`ppp.object`, `ppx`, `unmark`, `hyperframe`

Examples

```
data(amacrine)
# extract marks
m <- marks(amacrine)
# recode the mark values "off", "on" as 0, 1
marks(amacrine) <- as.integer(m == "on")
```

marks.psp

Marks of a Line Segment Pattern

Description

Extract or change the marks attached to a line segment pattern.

Usage

```
## S3 method for class 'psp'
marks(x, ..., dfok=TRUE)
## S3 replacement method for class 'psp'
marks(x, ...) <- value
```

Arguments

<code>x</code>	Line segment pattern dataset (object of class "psp").
<code>...</code>	Ignored.
<code>dfok</code>	Logical. If FALSE, data frames of marks are not permitted and will generate an error.
<code>value</code>	Vector or data frame of mark values, or NULL.

Details

These functions extract or change the marks attached to each of the line segments in the pattern `x`. They are methods for the generic functions `marks` and `marks<-` for the class "psp" of line segment patterns.

The expression `marks(x)` extracts the marks of `x`. The assignment `marks(x) <- value` assigns new marks to the dataset `x`, and updates the dataset `x` in the current environment.

The marks can be a vector, a factor, or a data frame.

For the assignment `marks(x) <- value`, the `value` should be a vector or factor of length equal to the number of segments in `x`, or a data frame with as many rows as there are segments in `x`. If `value` is a single value, or a data frame with one row, then it will be replicated so that the same marks will be attached to each segment.

To remove marks, use `marks(x) <- NULL` or `unmark(x)`.

Value

For `marks(x)`, the result is a vector, factor or data frame, containing the mark values attached to the line segments of `x`. If there are no marks, the result is `NULL`.

For `marks(x) <- value`, the result is the updated line segment pattern `x` (with the side-effect that the dataset `x` is updated in the current environment).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[psp.object](#), [marks](#), [marks<-](#)

Examples

```
example(pps)
marks(X)
marks(X)[,2]
marks(X) <- 42
marks(X) <- NULL
```

markstat

Summarise Marks in Every Neighbourhood in a Point Pattern

Description

Visit each point in a point pattern, find the neighbouring points, and summarise their marks

Usage

```
markstat(X, fun, N, R, ...)
```

Arguments

X	A marked point pattern. An object of class "ppp".
fun	Function to be applied to the vector of marks.
N	Integer. If this argument is present, the neighbourhood of a point of X is defined to consist of the N points of X which are closest to it. This argument is incompatible with R.
R	Nonnegative numeric value. If this argument is present, the neighbourhood of a point of X is defined to consist of all points of X which lie within a distance R of it. This argument is incompatible with N.
...	extra arguments passed to the function fun. They must be given in the form name=value.

Details

This algorithm visits each point in the point pattern X, determines which points of X are “neighbours” of the current point, extracts the marks of these neighbouring points, applies the function fun to the marks, and collects the value or values returned by fun.

The definition of “neighbours” depends on the arguments N and R, exactly one of which must be given.

If N is given, then the neighbours of the current point are the N points of X which are closest to the current point (including the current point itself). If R is given, then the neighbourhood of the current point consists of all points of X which lie closer than a distance R from the current point.

Each point of X is visited; the neighbourhood of the current point is determined; the marks of these points are extracted as a vector v ; then the function fun is called as:

```
fun(v, ...)
```

where \dots are the arguments passed from the call to `markstat`.

The results of each call to fun are collected and returned according to the usual rules for `apply` and its relatives. See **Value** above.

This function is just a convenient wrapper for a common use of the function `applynbd`. For more complex tasks, use `applynbd`. To simply tabulate the marks in every R-neighbourhood, use `marktable`.

Value

Similar to the result of `apply`. if each call to fun returns a single numeric value, the result is a vector of dimension $X\$n$, the number of points in X . If each call to fun returns a vector of the same length m , then the result is a matrix of dimensions $c(m, n)$; note the transposition of the indices, as usual for the family of `apply` functions. If the calls to fun return vectors of different lengths, the result is a list of length $X\$n$.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

`applynbd`, `marktable`, `ppp.object`, `apply`

Examples

```
data(longleaf)

# average diameter of 5 closest neighbours of each tree
md <- markstat(longleaf, mean, N=5)

# range of diameters of trees within 10 metre radius
rd <- markstat(longleaf, range, R=10)
```

Description

Visit each point in a point pattern, find the neighbouring points, and compile a frequency table of the marks of these neighbour points.

Usage

```
marktable(X, R, exclude=TRUE)
```

Arguments

- X A marked point pattern. An object of class "ppp".
 R Neighbourhood radius.
 exclude Logical. If exclude=TRUE, the neighbours of a point do not include the point itself. If exclude=FALSE, a point belongs to its own neighbourhood.

Details

This algorithm visits each point in the point pattern X, inspects all the neighbouring points within a radius R of the current point, and compiles a frequency table of the marks attached to the neighbours.

The dataset X must be a multitype point pattern, that is, `marks(X)` must be a factor.

The result is a two-dimensional contingency table with one row for each point in the pattern, and one column for each possible mark value. The [i, j] entry in the table gives the number of neighbours of point i that have mark j.

To perform more complicated calculations on the neighbours of every point, use `markstat` or `applynbd`.

Value

A contingency table (object of class "table") with one row for each point in X, and one column for each possible mark value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
 Rolf Turner <r.turner@auckland.ac.nz>

See Also

`markstat`, `applynbd`, `Kcross`, `ppp.object`, `table`

Examples

```
data(amacrine)
head(marktable(amacrine, 0.1))
head(marktable(amacrine, 0.1, exclude=FALSE))
```

Description

Estimate the mark variogram of a marked point pattern.

Usage

```
markvario(X, correction = c("isotropic", "Ripley", "translate"),
r = NULL, method = "density", ..., normalise=FALSE)
```

Arguments

X	The observed point pattern. An object of class "ppp" or something acceptable to as.ppp . It must have marks which are numeric.
correction	A character vector containing any selection of the options "isotropic", "Ripley" or "translate". It specifies the edge correction(s) to be applied.
r	numeric vector. The values of the argument r at which the mark variogram $\gamma(r)$ should be evaluated. There is a sensible default.
method	A character vector indicating the user's choice of density estimation technique to be used. Options are "density", "loess", "sm" and "smrep".
...	Arguments passed to the density estimation routine (density , loess or sm.density) selected by method.
normalise	If TRUE, normalise the variogram by dividing it by the estimated mark variance.

Details

The mark variogram $\gamma(r)$ of a marked point process X is a measure of the dependence between the marks of two points of the process a distance r apart. It is informally defined as

$$\gamma(r) = E\left[\frac{1}{2}(M_1 - M_2)^2\right]$$

where $E[\cdot]$ denotes expectation and M_1, M_2 are the marks attached to two points of the process a distance r apart.

The mark variogram of a marked point process is analogous, but **not equivalent**, to the variogram of a random field in geostatistics. See Waelder and Stoyan (1996).

Value

An object of class "fv" (see [fv.object](#)).

Essentially a data frame containing numeric columns

r	the values of the argument r at which the mark variogram $\gamma(r)$ has been estimated
theo	the theoretical value of $\gamma(r)$ when the marks attached to different points are independent; equal to the sample variance of the marks

together with a column or columns named "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $\gamma(r)$ obtained by the edge corrections named.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Cressie, N.A.C. (1991) *Statistics for spatial data*. John Wiley and Sons, 1991.
- Mase, S. (1996) The threshold method for estimating annual rainfall. *Annals of the Institute of Statistical Mathematics* **48** (1996) 201-213.
- Waelder, O. and Stoyan, D. (1996) On variograms in point process statistics. *Biometrical Journal* **38** (1996) 895-905.

See Also

Mark correlation function [markcorr](#) for numeric marks.

Mark connection function [markconnect](#) and multitype K-functions [Kcross](#), [Kdot](#) for factor-valued marks.

Examples

```
# Longleaf Pine data
# marks represent tree diameter
data(longleaf)
# Subset of this large pattern
swcorner <- owin(c(0,100),c(0,100))
sub <- longleaf[, swcorner]
# mark correlation function
mv <- markvario(sub)
plot(mv)
```

matchingdist

Distance for a Point Pattern Matching

Description

Computes the distance associated with a matching between two point patterns.

Usage

```
matchingdist(matching, type = NULL, cutoff = NULL, q = NULL)
```

Arguments

matching	A point pattern matching (an object of class "pppmatching").
type	A character string giving the type of distance to be computed. One of "spa", "ace" or "mat". See details below.
cutoff	The value > 0 at which interpoint distances are cut off.
q	The order of the average that is applied to the interpoint distances. May be Inf , in which case the maximum of the interpoint distances is taken.

Details

Computes the distance specified by type, cutoff, and order for a point matching. If any of these arguments are not provided, the function uses the corresponding elements of matching (if available).

For the type "spa" (subpattern assignment) it is assumed that the points of the point pattern with the smaller cardinality m are matched to a m -point subpattern of the point pattern with the larger cardinality n in a 1-1 way. The distance is then given as the q -th order average of the m distances between matched points (minimum of Euclidean distance and cutoff) and $n - m$ "penalty distances" of value cutoff.

For the type "ace" (assignment only if cardinalities equal) the matching is assumed to be 1-1 if the cardinalities of the point patterns are the same, in which case the q -th order average of the matching

distances (minimum of Euclidean distance and cutoff) is taken. If the cardinalities are different, the matching may be arbitrary and the distance returned is always equal to cutoff.

For the type `mat` (mass transfer) it is assumed that each point of the point pattern with the smaller cardinality m has mass 1, each point of the point pattern with the larger cardinality n has mass m/n , and fractions of these masses are matched in such a way that each point contributes exactly its mass. The distance is then given as the q-th order weighted average of all distances (minimum of Euclidean distance and cutoff) of (partially) matched points with weights equal to the fractional masses divided by m .

If the cardinalities of the two point patterns are equal, `matchingdist(m, type, cutoff, q)` yields the same result no matter if `type` is "spa", "ace" or "mat".

Value

Numeric value of the distance associated with the matching.

Author(s)

Dominic Schuhmacher <dominic.schuhmacher@stat.unibe.ch> <http://www.dominic.schuhmacher.name>

See Also

`pppdist` `pppmatching.object`

Examples

```
# an optimal matching
X <- runifpoint(20)
Y <- runifpoint(20)
m.opt <- pppdist(X, Y)
summary(m.opt)
matchingdist(m.opt)
# is the same as the distance given by summary(m.opt)

# sequential nearest neighbour matching
# (go through all points of point pattern X in sequence
# and match each point with the closest point of Y that is
# still unmatched)
am <- matrix(0, 20, 20)
h <- matrix(c(1:20, rep(0,20)), 20, 2)
h[1,2] = nncross(X[1],Y)[1,2]
for (i in 2:20) {
  nn <- nncross(X[i],Y[-h[1:(i-1),2]])[1,2]
  h[i,2] <- ((1:20)[-h[1:(i-1),2]])[nn]
}
am[h] <- 1
m.nn <- ppmatching(X, Y, am)
matchingdist(m.nn, type="spa", cutoff=1, q=1)
# is >= the distance obtained for m.opt
# in most cases strictly >

## Not run:
par(mfrow=c(1,2))
plot(m.opt)
plot(m.nn)
```

```
text(X$x, X$y, 1:20, pos=1, offset=0.3, cex=0.8)

## End(Not run)
```

matclust.estK

Fit the Matern Cluster Point Process by Minimum Contrast

Description

Fits the Matern Cluster point process to a point pattern dataset by the Method of Minimum Contrast.

Usage

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

Arguments

X	Data to which the Matern 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 Matern 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 Matern 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 Matern Cluster point process to X, by finding the parameters of the Matern Cluster model which give the closest match between the theoretical K function of the Matern Cluster process and the observed K function. For a more detailed explanation of the Method of Minimum Contrast, see [mincontrast](#).

The Matern Cluster point process is described in Moller 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.

The theoretical K -function of the Matern Cluster process is

$$K(r) = \pi r^2 + \frac{1}{\kappa} h\left(\frac{r}{2R}\right)$$

where

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

for $z \leq 1$, and $h(z) = 1$ for $z > 1$. The theoretical intensity of the Matern 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 Matern Cluster process can be simulated, using [rMatClust](#).

Homogeneous or inhomogeneous Matern 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:

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

Author(s)

Rasmus Waagepetersen <[rw@math.auc.dk](mailto:rwd@math.auc.dk)> Adapted for [spatstat](#) by Adrian Baddeley <Adrian.Baddeley@csiro.au>
<http://www.maths.uwa.edu.au/~adrian/>

References

- Moller, J. and Waagepetersen, R. (2003). Statistical Inference and Simulation for Spatial Point Processes. Chapman and Hall/CRC, Boca Raton.
- Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

[kppm](#), [lgcp.estK](#), [thomas.estK](#), [mincontrast](#), [Kest](#), [rMatClust](#) to simulate the fitted model.

Examples

```
data(redwood)
u <- matclust.estK(redwood, c(kappa=10, R=0.1))
u
plot(u)
```

matclust.estpcf

Fit the Matern Cluster Point Process by Minimum Contrast Using Pair Correlation

Description

Fits the Matern 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,R=1), lambda=NULL,
                 q = 1/4, p = 2, rmin = NULL, rmax = NULL, ...,
                 pcfargs=list())
```

Arguments

X	Data to which the Matern 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 Matern 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 Matern 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 Matern Cluster point process to X , by finding the parameters of the Matern Cluster model which give the closest match between the theoretical pair correlation function of the Matern Cluster process and the observed pair correlation function. For a more detailed explanation of the Method of Minimum Contrast, see [mincontrast](#).

The Matern Cluster point process is described in Moller 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.

The theoretical pair correlation function of the Matern Cluster process is

$$g(r) = 1 + \frac{1}{4\pi R\kappa r} h\left(\frac{r}{2R}\right)$$

where

$$h(z) = \frac{16}{\pi} [z \arccos(z) - z^2 \sqrt{1-z^2}]$$

for $z \leq 1$, and $h(z) = 0$ for $z > 1$. The theoretical intensity of the Matern 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 Matern Cluster process can be simulated, using [rMatClust](#).

Homogeneous or inhomogeneous Matern 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:

<code>par</code>	Vector of fitted parameter values.
<code>fit</code>	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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

References

- Moller, J. and Waagepetersen, R. (2003). Statistical Inference and Simulation for Spatial Point Processes. Chapman and Hall/CRC, Boca Raton.
- Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

[kppm](#), [matclust.estK](#), [thomas.estpcf](#), [thomas.estK](#), [lgcp.estK](#), [mincontrast](#), [pcf](#), [rMatClust](#) to simulate the fitted model.

Examples

```
data(redwood)
u <- matclust.estpcf(redwood, c(kappa=10, R=0.1))
u
plot(u, legendpos="topright")
```

mean.im

Maximum, Minimum, Mean, Median, Range or Sum of Pixel Values in an Image

Description

Calculates the mean, median, range, sum, maximum or minimum of the pixel values in a pixel image.

Usage

```
## S3 method for class 'im'
max(x, ...)
## S3 method for class 'im'
min(x, ...)
## S3 method for class 'im'
mean(x, ...)
## S3 method for class 'im'
median(x, ...)
## S3 method for class 'im'
range(x, ...)
## S3 method for class 'im'
sum(x, ...)
```

Arguments

- | | |
|-----|--|
| x | A pixel image (object of class "im"). |
| ... | Arguments passed to mean.default . |

Details

These functions calculate the mean, median, range, sum, maximum or minimum of the pixel values in the image x .

An object of class "im" describes a pixel image. See [im.object](#)) for details of this class.

The function [mean.im](#) is a method for the generic function [mean](#) for the class "im". Similarly [median.im](#) is a method for the generic [median](#) and [range.im](#) is a method for [range](#).

If the image x is logical-valued, the mean value of x is the fraction of pixels that have the value TRUE. The median is not defined.

If the image x is factor-valued, then the mean of x is the mean of the integer codes of the pixel values. The median and range are not defined.

Any arguments in ... are passed to the default method, for example [mean.default](#). In particular, using the argument [trim](#) will compute the trimmed mean, as explained in the help for [mean.default](#).

Other information about an image can be obtained using [summary.im](#) or [quantile.im](#).

Value

A single number.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[mean](#), [median](#), [range](#), [sum](#), [mean.default](#), [median.default](#), [range.default](#), [quantile.im](#), [im.object](#), [summary.im](#).

Examples

```
X <- as.im(function(x,y) {x^2}, unit.square())
mean(X)
median(X)
range(X)

mean(X, trim=0.05)
```

Description

Methods for class "box3".

Usage

```
## S3 method for class 'box3'
print(x, ...)
## S3 method for class 'box3'
unitname(x)
## S3 replacement method for class 'box3'
unitname(x) <- value
```

Arguments

x	Object of class "box3" representing a three-dimensional box.
...	Other arguments passed to <code>print.default</code> .
value	Name of the unit of length. See <code>unitname</code> .

Details

These are methods for the generic functions `print` and `unitname` for the class "box3" of three-dimensional boxes.

The `print` method prints a description of the box, while the `unitname` method extracts the name of the unit of length in which the box coordinates are expressed.

Value

For `print.box3` the value is `NULL`. For `unitname.box3` an object of class "units".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`box3`, `print`, `unitname`

Examples

```
X <- box3(c(0,10),c(0,10),c(0,5), unitname=c("metre", "metres"))
X
unitname(X)
# Northern European usage
unitname(X) <- "meter"
```

Description

Methods for class "boxx".

Usage

```
## S3 method for class 'boxx'
print(x, ...)
## S3 method for class 'boxx'
unitname(x)
## S3 replacement method for class 'boxx'
unitname(x) <- value
```

Arguments

x	Object of class "boxx" representing a multi-dimensional box.
...	Other arguments passed to <code>print.default</code> .
value	Name of the unit of length. See <code>unitname</code> .

Details

These are methods for the generic functions `print` and `unitname` for the class "boxx" of multi-dimensional boxes.

The `print` method prints a description of the box, while the `unitname` method extracts the name of the unit of length in which the box coordinates are expressed.

Value

For `print.boxx` the value is `NULL`. For `unitname.boxx` an object of class "units".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`boxx`, `print`, `unitname`

Examples

```
X <- boxx(c(0,10),c(0,10),c(0,5),c(0,1), unitname=c("metre", "metres"))
X
unitname(X)
# Northern European usage
unitname(X) <- "meter"
```

Description

Methods for objects of the class "distfun".

Usage

```
## S3 method for class 'distfun'  
contour(x, ...)  
## S3 method for class 'distfun'  
persp(x, ...)  
## S3 method for class 'distfun'  
plot(x, ...)  
## S3 method for class 'distfun'  
print(x, ...)
```

Arguments

- x Object of class "distfun" representing a distance function.
... Named arguments controlling the plot. See Details.

Details

These are methods for the generic functions `print`, `plot`, `contour` and `persp` for the class "distfun" of distance functions. See `distfun` for explanation about this class.

The `print` method prints a description of the domain of the distance function and the spatial object of which it is the distance function. Any additional arguments ... are ignored.

The `plot`, `contour` and `persp` methods first convert x to a pixel image object using `as.im`, then display it using `plot.im`, `contour.im` or `persp.im`. Additional arguments ... are either passed to `as.im.function` to control the spatial resolution of the pixel image, or passed to `contour.im`, `persp.im` or `plot.im` to control the appearance of the plot.

Value

NULL.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`distfun`, `as.im`, `plot.im`, `persp.im`, `contour.im`, `spatstat.options`

Examples

```
data(letterR)  
f <- distfun(letterR)  
contour(f)  
contour(f, W=owin(c(1,5),c(-1,4)), eps=0.1)
```

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

<code>x,object</code>	An object of class "kppm", representing a fitted cluster point process model.
<code>...</code>	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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

See Also

`kppm`, `plot.kppm`, `predict.kppm`, `simulate.kppm`, `update.kppm`, `vcov.kppm`.

Examples

```
data(redwood)
fit <- kppm(redwood, ~x, "MatClust")
coef(fit)
formula(fit)
tf <- terms(fit)
labels(fit)
```

Description

These are methods for the class "linnet" of linear networks.

Usage

```
as.linnet(X, ...)
## S3 method for class 'linnet'
as.linnet(X, ...)
## S3 method for class 'lpp'
as.linnet(X, ..., fatal=TRUE)
## S3 method for class 'linnet'
as.owin(W, ...)
## S3 method for class 'linnet'
as.psp(x, ..., fatal=TRUE)
## S3 method for class 'linnet'
print(x, ...)
## S3 method for class 'linnet'
summary(object, ...)
## S3 method for class 'linnet'
unitname(x)
## S3 replacement method for class 'linnet'
unitname(x) <- value
```

Arguments

- x,X,object,W** An object of class "linnet" representing a linear network.
- ...** Arguments passed to other methods.
- value** A valid name for the unit of length for x. See [unitname](#).
- fatal** Logical value indicating whether data in the wrong format should lead to an error (fatal=TRUE) or a warning (fatal=FALSE).

Details

The function `as.linnet` is generic. It converts data from some other format into an object of class "linnet". The method `as.linnet.lpp` extracts the linear network information from an `lpp` object.

The other functions are methods for the generic commands `as.owin`, `as.psp`, `print`, `summary`, `unitname` and `unitname<-` for the class "linnet".

The method `as.owin.linnet` extracts the window containing the linear network, and returns it as an object of class "owin".

The method `as.psp.linnet` extracts the lines of the linear network as a line segment pattern (object of class "psp").

Value

For `as.linnet` the value is an object of class "linnet". For other functions, see the help file for the corresponding generic function.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

See Also

[linnet](#)

Examples

```
data(simplicenet)
simplicenet
summary(simplicenet)
unitname(simplicenet) <- c("cubit", "cubits")
```

Description

These are methods specifically for the class "lpp" of point patterns on linear networks.

Usage

```
## S3 method for class 'lpp'
as.ppp(X, ..., fatal=TRUE)
## S3 method for class 'lpp'
as.psp(x, ..., fatal=TRUE)
## S3 replacement method for class 'lpp'
marks(x, ...) <- value
## S3 method for class 'lpp'
print(x, ...)
## S3 method for class 'summary.lpp'
print(x, ...)
## S3 method for class 'lpp'
summary(object, ...)
## S3 method for class 'lpp'
unitname(x)
## S3 replacement method for class 'lpp'
unitname(x) <- value
## S3 method for class 'lpp'
unmark(X)
```

Arguments

x, X, object	An object of class "lpp" representing a point pattern on a linear network.
...	Arguments passed to other methods.
value	Replacement value for the marks or unitname of x. See Details.
fatal	Logical value indicating whether data in the wrong format should lead to an error (fatal=TRUE) or a warning (fatal=FALSE).

Details

These are methods for the generic functions `as.ppp`, `as.psp`, `marks<-`, `print`, `summary`, `unitname`, `unitname<-` and `unmark` for objects of the class "lpp".

For "marks<- .lpp" the replacement value should be either NULL, or a vector of length equal to the number of points in x, or a data frame with one row for each point in x.

For "unitname<- .lpp" the replacement value should be a valid name for the unit of length, as described in `unitname`.

Value

See the documentation on the corresponding generic function.

Other methods

An object of class "lpp" also inherits the class "ppx" for which many other methods are available. See `methods.ppx`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

See Also

`lpp`, `methods.ppx`

Examples

```
example(lpp)
X
as.ppp(X)
summary(X)
unitname(X) <- c("furlong", "furlongs")
```

Description

These are methods for the class "lppm" of fitted point process models on a linear network.

Usage

```
## S3 method for class 'lppm'
coef(object, ...)
## S3 method for class 'lppm'
extractAIC(fit, ...)
## S3 method for class 'lppm'
formula(x, ...)
## S3 method for class 'lppm'
logLik(object, ...)
## S3 method for class 'lppm'
plot(x, ..., type="trend")
## S3 method for class 'lppm'
print(x, ...)
## S3 method for class 'lppm'
terms(x, ...)
## S3 method for class 'lppm'
update(object, ...)
## S3 method for class 'lppm'
as.linnet(X, ...)
```

Arguments

<code>object, fit, x, X</code>	An object of class "lppm" representing a fitted point process model on a linear network.
<code>...</code>	Arguments passed to other methods, usually the method for the class "ppm".
<code>type</code>	Character string (either "trend" or "cif") determining whether to plot the fitted first order trend or the conditional intensity.

Details

These are methods for the generic commands `coef`, `extractAIC`, `formula`, `logLik`, `plot`, `print`, `terms` and `update` for the class "lppm".

Value

See the default methods.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

See Also

[lppm](#)

Examples

```
example(lpp)
fit <- lppm(X, ~x)
print(fit)
plot(fit)
coef(fit)
formula(fit)
terms(fit)
logLik(fit)
extractAIC(fit)
update(fit, ~1)
```

methods.pp3

Methods for three-dimensional point patterns

Description

Methods for class "pp3".

Usage

```
## S3 method for class 'pp3'
print(x, ...)
## S3 method for class 'summary.pp3'
print(x, ...)
## S3 method for class 'pp3'
summary(object, ...)
## S3 method for class 'pp3'
unitname(x)
## S3 replacement method for class 'pp3'
unitname(x) <- value
```

Arguments

x,object	Object of class "pp3".
...	Ignored.
value	Name of the unit of length. See unitname .

Details

These are methods for the generic functions [print](#), [summary](#), [unitname](#) and [unitname<-](#) for the class "pp3" of three-dimensional point patterns.

The [print](#) and [summary](#) methods print a description of the point pattern.

The [unitname](#) method extracts the name of the unit of length in which the point coordinates are expressed. The [unitname<-](#) method assigns the name of the unit of length.

Value

For [print.pp3](#) the value is NULL. For [unitname.pp3](#) an object of class "units".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[pp3](#), [print](#), [unitname](#) [unitname<-](#)

Examples

```
X <- pp3(runif(42),runif(42),runif(42), box3(c(0,1), unitname="mm"))
X
unitname(X)
unitname(X) <- c("foot", "feet")
summary(X)
```

Description

Methods for printing and plotting a general multidimensional space-time point pattern.

Usage

```
## S3 method for class 'ppx'
print(x, ...)
## S3 method for class 'ppx'
plot(x, ...)
## S3 method for class 'ppx'
unitname(x)
## S3 replacement method for class 'ppx'
unitname(x) <- value
```

Arguments

- x Multidimensional point pattern (object of class "ppx").
- ... Additional arguments passed to plot methods.
- value Name of the unit of length. See [unitname](#).

Details

These are methods for the generic functions [print](#), [plot](#), [unitname](#) and [unitname<-](#) for the class "ppx" of multidimensional point patterns.

The [print](#) method prints a description of the point pattern and its spatial domain.

The [unitname](#) method extracts the name of the unit of length in which the point coordinates are expressed. The [unitname<-](#) method assigns the name of the unit of length.

Value

For [print.ppx](#) the value is NULL. For [unitname.ppx](#) an object of class "units".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppx](#), [unitname](#)

methods.rho2hat

Methods for Intensity Functions of Two Spatial Covariates

Description

These are methods for the class "rho2hat".

Usage

```
## S3 method for class 'rho2hat'  
plot(x, ..., do.points=FALSE)  
## S3 method for class 'rho2hat'  
print(x, ...)
```

Arguments

x	An object of class "rho2hat".
...	Arguments passed to other methods.
do.points	Logical value indicating whether to plot the observed values of the covariates at the data points.

Details

These functions are methods for the generic commands [print](#) and [plot](#) for the class "rho2hat".

An object of class "rho2hat" is an estimate of the intensity of a point process, as a function of two given spatial covariates. See [rho2hat](#).

The method [plot.rho2hat](#) displays the estimated function ρ using [plot.fv](#), and optionally adds a [rug](#) plot of the observed values of the covariate.

Value

NULL.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

See Also

[rho2hat](#)

Examples

```
data(b ei)
attach(b ei.extra)
r2 <- rho2hat(b ei, elev, grad)
r2
plot(r2)
```

methods.rhohat

Methods for Intensity Functions of Spatial Covariate

Description

These are methods for the class "rhohat".

Usage

```
## S3 method for class 'rhohat'
print(x, ...)
## S3 method for class 'rhohat'
plot(x, ..., do.rug=TRUE)
## S3 method for class 'rhohat'
predict(object, ..., relative=FALSE)
```

Arguments

x,object	An object of class "rhohat" representing a smoothed estimate of the intensity function of a point process.
...	Arguments passed to other methods.
do.rug	Logical value indicating whether to plot the observed values of the covariate as a rug plot along the horizontal axis.
relative	Logical value indicating whether to compute the estimated point process intensity (relative=FALSE) or the relative risk (relative=TRUE) in the case of a relative risk estimate.

Details

These functions are methods for the generic commands `print`, `plot` and `predict` for the class "rhohat".

An object of class "rhohat" is an estimate of the intensity of a point process, as a function of a given spatial covariate. See [rhohat](#).

The method `plot.rhohat` displays the estimated function ρ using `plot.fv`, and optionally adds a `rug` plot of the observed values of the covariate.

The method `predict.rhohat` computes a pixel image of the intensity $\rho(Z(u))$ at each spatial location u , where Z is the spatial covariate.

Value

For `predict.rhohat` the value is a pixel image (object of class "im"). For other functions, the value is NULL.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

See Also

[rhohat](#)

Examples

```
X <- rpoispp(function(x,y){exp(3+3*x)})
rho <- rhohat(X, function(x,y){x})
rho
plot(rho)
Y <- predict(rho)
plot(Y)
#
fit <- ppm(X, ~x)
rho <- rhohat(fit, "y")
opa <- par(mfrow=c(1,2))
plot(predict(rho))
plot(predict(rho, relative=TRUE))
par(opa)
```

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'
terms(x, ...)
## S3 method for class 'slrm'
labels(object, ...)
```

Arguments

- | | |
|------------------------|---|
| <code>x, object</code> | An object of class "slrm", representing a fitted cluster point process model. |
| <code>...</code> | Arguments passed to other methods. |

Details

These functions are methods for the generic commands `formula`, `print`, `terms` and `labels` 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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

See Also

[slrm](#), [plot.slrm](#), [predict.slrm](#), [simulate.slrm](#), [update.slrm](#), [vcov.slrm](#), [coef.slrm](#).

Examples

```
data(redwood)
fit <- slrm(redwood ~ x)
coef(fit)
formula(fit)
tf <- terms(fit)
labels(fit)
```

Description

Methods for class "units".

Usage

```
## S3 method for class 'units'
print(x, ...)
## S3 method for class 'units'
summary(object, ...)
## S3 method for class 'units'
rescale(X,s)
## S3 method for class 'units'
compatible(A,B, ..., coerce=TRUE)
```

Arguments

- x,X,A,B,object Objects of class "units" representing units of length.
- s Conversion factor: the new units are s times the old units.
- ... Other arguments. For `print.units` these arguments are passed to `print.default`. For `summary.units` they are ignored. For `compatible.units` these arguments are other objects of class "units".
- coerce Logical. If TRUE, a null unit of length is compatible with any non-null unit.

Details

These are methods for the generic functions `print`, `summary`, `rescale` and `compatible` for the class "units".

An object of class "units" represents a unit of length.

The `print` method prints a description of the unit of length, and the `summary` method gives a more detailed description.

The `rescale` method changes the unit of length by rescaling it.

The `compatible` method tests whether two or more units of length are compatible.

Value

For `print.units` the value is `NULL`. For `summary.units` the value is an object of class `summary.units` (with its own `print` method). For `rescale.units` the value is another object of class "units". For `compatible.units` the result is logical.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`box3`, `print`, `unitname`

midpoints.psp

Midpoints of Line Segment Pattern

Description

Computes the midpoints of each line segment in a line segment pattern.

Usage

`midpoints.psp(x)`

Arguments

`x` A line segment pattern (object of class "psp").

Details

The midpoint of each line segment is computed.

Value

Point pattern (object of class "ppp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[summary.psp](#), [lengths.psp](#), [angles.psp](#)

Examples

```
a <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
b <- midpoints.psp(a)
```

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))
```

Arguments

<code>observed</code>	Summary statistic, computed for the data. An object of class "fv".
<code>theoretical</code>	An R language function that calculates the theoretical expected value of the summary statistic, given the model parameters. See Details.
<code>startpar</code>	Vector of initial values of the parameters of the point process model (passed to <code>theoretical</code>).
<code>...</code>	Additional arguments passed to the function <code>theoretical</code> and to the optimisation algorithm <code>optim</code> .
<code>ctrl</code>	Optional. List of arguments controlling the optimisation. See Details.
<code>fvlab</code>	Optional. List containing some labels for the return value. See Details.
<code>explain</code>	Optional. List containing strings that give a human-readable description of the model, the data and the summary statistic.

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 Matern cluster process, see [matclust.estK](#).

The Method of Minimum Contrast (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 `crtl` determines the contrast criterion (the objective function that will be minimised). The algorithm minimises the criterion

$$D(\theta) = \int_{r_{\min}}^{r_{\max}} |\hat{F}(r)^q - F_\theta(r)^q|^p dr$$

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, 2006).

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:

<code>par</code>	Vector of fitted parameter values.
<code>fit</code>	Function value table (object of class "fv") containing the observed values of the summary statistic (<code>observed</code>) and the theoretical values of the summary statistic computed from the fitted model parameters.
<code>opt</code>	The return value from the optimizer <code>optim</code> .
<code>crtl</code>	The control parameters of the algorithm.
<code>info</code>	List of explanatory strings.

Author(s)

Rasmus Waagepetersen <rwm@math.auc.dk>, adapted for `spatstat` by Adrian Baddeley <Adrian.Baddeley@csiro.au>
<http://www.maths.uwa.edu.au/~adrian/>

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

Waagepetersen, R. (2006). An estimation function approach to inference for inhomogeneous Neyman-Scott processes. Submitted.

See Also

`kppm`, `lgcp.estK`, `matclust.estK`, `thomas.estK`,

`miplot`

Morishita Index Plot

Description

Displays the Morishita Index Plot of a spatial point pattern.

Usage

```
miplot(X, ...)
```

Arguments

- | | |
|----------------|--|
| <code>X</code> | A point pattern (object of class "ppp") or something acceptable to <code>as.ppp</code> . |
| ... | Optional arguments to control the appearance of the plot. |

Details

Morishita (1959) defined an index of spatial aggregation for a spatial point pattern based on quadrat counts. The spatial domain of the point pattern is first divided into Q subsets (quadrats) of equal size and shape. The numbers of points falling in each quadrat are counted. Then the Morishita Index is computed as

$$\text{MI} = Q \frac{\sum_{i=1}^Q n_i(n_i - 1)}{N(N - 1)}$$

where n_i is the number of points falling in the i -th quadrat, and N is the total number of points. If the pattern is completely random, MI should be approximately equal to 1. Values of MI greater than 1 suggest clustering.

The *Morishita Index plot* is a plot of the Morishita Index MI against the linear dimension of the quadrats. The point pattern dataset is divided into 2×2 quadrats, then 3×3 quadrats, etc, and the Morishita Index is computed each time. This plot is an attempt to discern different scales of dependence in the point pattern data.

Value

None.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

References

- M. Morishita (1959) Measuring of the dispersion of individuals and analysis of the distributional patterns. Memoir of the Faculty of Science, Series E2, Kyushu University. Pages 215–235.

See Also

[quadratcount](#)

Examples

```
data(longleaf)
miplot(longleaf)
opa <- par(mfrow=c(2,3))
data(cells)
data(japanesepines)
data(redwood)
plot(cells)
plot(japanesepines)
plot(redwood)
miplot(cells)
miplot(japanesepines)
miplot(redwood)
par(opa)
```

model.depends

Identify Covariates Involved in each Model Term

Description

Given a fitted model (of any kind), identify which of the covariates is involved in each term of the model.

Usage

```
model.depends(object)
model.is.additive(object)
model.covariates(object, fitted=TRUE, offset=TRUE)
has.offset.term(object)
has.offset(object)
```

Arguments

- object** A fitted model of any kind.
- fitted, offset** Logical values determining which type of covariates to include.

Details

The object can be a fitted model of any kind, including models of the classes [lm](#), [glm](#) and [ppm](#).

To be precise, object must belong to a class for which there are methods for [formula](#), [terms](#) and [model.matrix](#).

The command `model.depends` determines the relationship between the original covariates (the data supplied when object was fitted) and the canonical covariates (the columns of the design matrix). It returns a logical matrix, with one row for each canonical covariate, and one column for each of the original covariates, with the i, j entry equal to TRUE if the i th canonical covariate depends on the j th original covariate.

If the model formula of object includes offset terms (see [offset](#)), then the return value of `model.depends` also has an attribute "offset". This is a logical value or matrix with one row for each offset term and one column for each of the original covariates, with the i, j entry equal to TRUE if the i th offset term depends on the j th original covariate.

The command `model.covariates` returns a character vector containing the names of all (original) covariates that were actually used to fit the model. By default, this includes all covariates that appear in the model formula, including offset terms as well as canonical covariate terms. To omit the offset terms, set `offset=FALSE`. To omit the canonical covariate terms, set `fitted=FALSE`.

The command `model.is.additive` determines whether the model is additive, in the sense that there is no canonical covariate that depends on two or more original covariates. It returns a logical value.

The command `has.offset.term` is a faster way to determine whether the model *formula* includes an offset term.

The functions `model.depends` and `has.offset.term` only detect offset terms which are present in the model formula. They do not detect numerical offsets in the model object, that were inserted using the `offset` argument in `lm`, `glm` etc. To detect the presence of offsets of both kinds, use `has.offset`.

Value

A logical value or matrix.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppm](#), `model.matrix`

Examples

```
x <- 1:10
y <- 3*x + 2
z <- rep(c(-1,1), 5)
fit <- lm(y ~ poly(x,2) + sin(z))
model.depends(fit)
model.covariates(fit)
model.is.additive(fit)

fitoff1 <- lm(y ~ x + offset(z))
fitoff2 <- lm(y ~ x, offset=z)
has.offset.term(fitoff1)
has.offset(fitoff1)
has.offset.term(fitoff2)
has.offset(fitoff2)
```

model.frame.ppm*Extract the Variables in a Point Process Model*

Description

Given a fitted point process model, this function returns a data frame containing all the variables needed to fit the model using the Berman-Turner device.

Usage

```
## S3 method for class 'ppm'
model.frame(formula, ...)

## S3 method for class 'kppm'
model.frame(formula, ...)

## S3 method for class 'lppm'
model.frame(formula, ...)
```

Arguments

formula	A fitted point process model. An object of class "ppm" or "kppm" or "lppm".
...	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" or "kppm" or "lppm").

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

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

See Also

[ppm](#), [kppm](#), [lppm](#), [model.frame](#), [model.matrix.ppm](#)

Examples

```
fit <- ppm(cells, ~x)
mf <- model.frame(fit)
kfit <- kppm(redwood, ~x, "Thomas")
kmf <- model.frame(kfit)
```

`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 'lppm'
model.images(object, L = as.linnet(object), ...)
```

Arguments

- | | |
|---------------------|--|
| <code>object</code> | The fitted point process model. An object of class "ppm" or "kppm" or "lppm". |
| <code>W</code> | A window (object of class "owin") in which the images should be computed. Defaults to the window in which the model was fitted. |
| <code>L</code> | A linear network (object of class "linnet") in which the images should be computed. Defaults to the network in which the model was fitted. |
| <code>...</code> | Other arguments (such as <code>na.action</code>) 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 "lppm" (produced by [lppm](#)).

The spatial covariates required by the model-fitting procedure are computed at every pixel location in the window `W`. For `lppm` objects, the covariates are computed at every location on the network `L`.

Note that the spatial covariates computed here are not the original covariates that were supplied when fitting the model. Rather, they are the covariates that actually 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 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 "listof" so that it can be plotted immediately.

Value

An object of class "listof" consisting of a named list of pixel images (objects of class "im"). This list can be plotted immediately using `plot.listof`.

For `model.images.lppm`, the images are also of class "linim".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`model.matrix.ppm`, `model.matrix.ppm`, `ppm.object`, `im`, `im.object`, `plot.listof`, `spatstat.options`

Examples

```
fit <- ppm(cells, ~x)
model.images(fit)
fit2 <- ppm(cells, ~cut(x, 3))
model.images(fit2)
```

`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), ..., keepNA=TRUE)

## S3 method for class 'kppm'
model.matrix(object, data=model.frame(object), ...,
keepNA=TRUE)

## S3 method for class 'lppm'
model.matrix(object, data=model.frame(object), ..., keepNA=TRUE)
```

Arguments

object	The fitted point process model. An object of class "ppm" or "kppm" or "lppm".
data	A model frame, containing the data required for the Berman-Turner device.
keepNA	Logical. Determines whether rows containing NA values will be deleted or retained.
...	Other arguments (such as na.action) passed to model.matrix.lm .

Details

These commands are methods for the generic function [model.matrix](#). They extracts the design matrix of a spatial point process model (class "ppm" or "kppm" or "lppm").

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 "lppm") fitted to spatial point pattern data. Such objects are produced by the model-fitting functions [ppm](#), [kppm](#) and [lppm](#).

The methods [model.matrix.ppm](#), [model.matrix.kppm](#) and [model.matrix.lppm](#) 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. Rows of the matrix correspond to quadrature points in the fitting procedure (provided keepNA=TRUE). Columns are covariates in the model.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[model.images](#), [ppm](#), [kppm](#), [ppm.object](#), [quad.ppm](#), [residuals.ppm](#), [model.matrix](#)

Examples

```
fit <- ppm(cells, ~x)
model.matrix(fit)
# matrix with two columns: '(Intercept)' and 'x'
kfit <- kppm(redwood, ~x, "Thomas")
m <- model.matrix(kfit)
```

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.

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`" that can be plotted by `plot.msr`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

References

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

See Also

`plot.msr`, `smooth.msr`, `[.msr`

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(types=NULL, hradii)
```

Arguments

types	Optional; vector of all possible types (i.e. the possible levels of the <code>marks</code> variable in the data)
hradii	Matrix of hard core radii

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))`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`ppm`, `pairwise.family`, `ppm.object`, `MultiStrauss`, `MultiStraussHard`, `Strauss`

Examples

```

h <- matrix(c(1,2,2,1), nrow=2,ncol=2)
MultiHard(hradii=h)
# prints a sensible description of itself
data(betacells)
h <- 15.0 * matrix(c(NA,1,1,NA), nrow=2,ncol=2)
ppm(betacells, ~1, MultiHard(,h))
# fit the stationary multitype hardcore process to 'betacells'
# Note the comma; needed since "types" is not specified.

```

multiplicity.hpp

Count Multiplicity of Duplicate Points

Description

Counts the number of duplicates for each point in a spatial point pattern.

Usage

```
multiplicity.hpp(x)
```

Arguments

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

Details

Two points in a point pattern are deemed to be identical if their x, y coordinates are the same, and their marks are also the same (if they carry marks). The Examples section illustrates how it is possible for a point pattern to contain a pair of identical points.

For each point in **x**, this function counts how many points are identical to it, and returns the vector of counts.

Value

A vector of integers (multiplicities) of length equal to the number of points in **x**.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppp.object](#), [duplicated.hpp](#), [unique.hpp](#)

Examples

```

X <- ppp(c(1,1,0.5), c(2,2,1), window=square(3))
m <- multiplicity.hpp(X)

# unique points in X, marked by multiplicity
first <- !duplicated(X)
Y <- X[first] %mark% m[first]

```

Description

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

Usage

```
MultiStrauss(types=NULL, radii)
```

Arguments

types	Optional; vector of all possible types (i.e. the possible levels of the marks variable in the data)
radii	Matrix of interaction radii

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))`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

`ppm`, `pairwise.family`, `ppm.object`, `Strauss`, `MultiHard`

Examples

```
r <- matrix(c(1,2,2,1), nrow=2,ncol=2)
MultiStrauss(radui=r)
# prints a sensible description of itself
data(betacells)
r <- 30.0 * matrix(c(1,2,2,1), nrow=2,ncol=2)
ppm(betacells, ~1, MultiStrauss(, r))
# fit the stationary multitype Strauss process to 'betacells'
# Note the comma; needed since "types" is not specified.

## Not run:
ppm(betacells, ~polynom(x,y,3), MultiStrauss(c("off","on"), r))
# fit a nonstationary Strauss process with log-cubic polynomial trend

## End(Not run)
```

Description

Creates an instance of the multitype/hard core Strauss point process model which can then be fitted to point pattern data.

Usage

```
MultiStraussHard(types=NULL, iradii, hradii)
```

Arguments

types	Optional; vector of all possible types (i.e. the possible levels of the marks variable in the data)
iradii	Matrix of interaction radii
hradii	Matrix of hard core radii

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))`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

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(iradii=r,hradii=h)
# prints a sensible description of itself
data(betacells)
r <- 30.0 * matrix(c(1,2,2,1), nrow=2,ncol=2)
h <- 15.0 * matrix(c(NA,1,1,NA), nrow=2,ncol=2)
ppm(betacells, ~1, MultiStraussHard(r,h))
# fit the stationary multitype hardcore Strauss process to 'betacells'
# Note the comma; needed since "types" is not specified.
```

Description

Data recording the spatial locations of gold deposits and associated geological features in the Murchison area of Western Australia. Extracted from a large scale (1:500,000) study of the Murchison area by the Geological Survey of Western Australia (Watkins and Hickman, 1990). The features recorded are

- the locations of gold deposits;

- the locations of geological faults;
- the region that contains greenstone bedrock.

The study region is contained in a 330×400 kilometre rectangle. At this scale, gold deposits are points, i.e. their spatial extent is negligible. Gold deposits in this region occur only in greenstone bedrock. Geological faults can be observed reliably only within the same region. However, some faults have been extrapolated (by geological “interpretation”) outside the greenstone boundary from information observed in the greenstone region.

These data were analysed by Foxall and Baddeley (2002) and Brown et al (2002); see also Groves et al (2000), Knox-Robinson and Groves (1997). The main aim is to predict the intensity of the point pattern of gold deposits from the more easily observable fault pattern.

Usage

```
data(murchison)
```

Format

`murchison` is a list with the following entries:

gold a point pattern (object of class "ppp") representing the point pattern of gold deposits. See [ppp.object](#) for details of the format.

faults a line segment pattern (object of class "psp") representing the geological faults. See [psp.object](#) for details of the format.

greenstone the greenstone bedrock region. An object of class "owin". Consists of multiple irregular polygons with holes.

All coordinates are given in **metres**.

Source

Data were kindly provided by Dr Carl Knox-Robinson of the Department of Geology and Geophysics, University of Western Australia. Permission to use the data is granted by Dr Tim Griffin, Geological Survey of Western Australia and by Dr Knox-Robinson. *Please make appropriate acknowledgement* to Watkins and Hickman (1990) and the Geological Survey of Western Australia.

References

- Brown, W.M., Gedeon, T.D., Baddeley, A.J. and Groves, D.I. (2002) Bivariate J-function and other graphical statistical methods help select the best predictor variables as inputs for a neural network method of mineral prospectivity mapping. In U. Bayer, H. Burger and W. Skala (eds.) *IAMG 2002: 8th Annual Conference of the International Association for Mathematical Geology*, Volume 1, 2002. International Association of Mathematical Geology. Pages 257–268.
- Foxall, R. and Baddeley, A. (2002) Nonparametric measures of association between a spatial point process and a random set, with geological applications. *Applied Statistics* **51**, 165–182.
- Groves, D.I., Goldfarb, R.J., Knox-Robinson, C.M., Ojala, J., Gardoll, S., Yun, G.Y. and Holyland, P. (2000) Late-kinematic timing of orogenic gold deposits and significance for computer-based exploration techniques with emphasis on the Yilgarn Block, Western Australia. *Ore Geology Reviews*, **17**, 1–38.
- Knox-Robinson, C.M. and Groves, D.I. (1997) Gold prospectivity mapping using a geographic information system (GIS), with examples from the Yilgarn Block of Western Australia. *Chronique de la Recherche Minière* **529**, 127–138.

Watkins, K.P. and Hickman, A.H. (1990) *Geological evolution and mineralization of the Murchison Province, Western Australia*. Bulletin 137, Geological Survey of Western Australia. 267 pages. Published by Department of Mines, Western Australia, 1990. Available online from Department of Industry and Resources, State Government of Western Australia, www.doir.wa.gov.au

Examples

```
if(interactive()) {  
  data(murchison)  
  plot(murchison$greenstone, main="Murchison data", col="lightgreen")  
  plot(murchison$gold, add=TRUE, pch="+", col="blue")  
  plot(murchison$faults, add=TRUE, col="red")  
}
```

nbfires

Point Patterns of New Brunswick Forest Fires

Description

Point patterns created from yearly records, provided by the New Brunswick Department of Natural Resources, of all fires falling under their jurisdiction for the years 1987 to 2003 inclusive (with the year 1988 omitted until further notice).

Usage

```
data(nbfires)
```

Format

Executing `data(nbfires)` creates two objects: `nbfires` and `nbw.rect`.

The object `nbfires` is a marked point pattern (an object of class "ppp") consisting of all of the fires in the years 1987 to 2003 inclusive, with the omission of 1988. The marks consist of a data frame of auxiliary information about the fires; see *Details*. Patterns for individual years can be extracted using the function `split.ppp()`. (See **Examples**.)

The object `nbw.rect` is a rectangular window which covers central New Brunswick. It is provided for use in illustrative and ‘practice’ calculations inasmuch as the use of a rectangular window simplifies some computations considerably.

Details

The coordinates of the fire locations were provided in terms of latitude and longitude, to the nearest minute of arc. These were converted to New Brunswick stereographic projection coordinates (Thomson, Mephan and Steeves, 1977) which was the coordinate system in which the map of New Brunswick — which constitutes the observation window for the pattern — was obtained. The conversion was done using a C program kindly provided by Jonathan Beaudoin of the Department of Geodesy and Geomatics, University of New Brunswick.

Finally the data and window were rescaled since the use of the New Brunswick stereographic projection coordinate system resulted in having to deal with coordinates which are expressed as very large integers with a bewildering number of digits. Amongst other things, these huge numbers tended to create very untidy axis labels on graphs. The width of the bounding box of the window

was made equal to 1000 (nameless) units. In addition the lower left hand corner of this bounding box was shifted to the origin. The height of the bounding box was changed proportionately, resulting in a value of approximately 959.

The window for the fire patterns comprises 6 polygonal components, consisting of mainland New Brunswick and the 5 largest islands. Some lakes which should form holes in the mainland component are currently missing; this problem may be remedied in future releases. The window was formed by ‘simplifying’ the map that was originally obtained. The simplification consisted in reducing (using an interactive visual technique) the number of polygon edges in each component. For instance the number of edges in the mainland component was reduced from over 138,000 to 500.

For some purposes it is probably better to use a discretized (mask type) window. See **Examples**.

Because of the coarseness of the coordinates of the original data (1 minute of longitude is approximately 1 kilometer at the latitude of New Brunswick), data entry errors, and the simplification of the observation window, many of the original fire locations appeared to be outside of the window. This problem was addressed by shifting the location of the ‘outsider’ points slightly, or deleting them, as seemed appropriate.

The columns of the data frame comprising the marks of nbfires are:

year This a *factor* with levels 1987, 1989, ..., 2002, 2003. Note that 1988 is not present in the levels.

fire.type A factor with levels forest, grass, dump, and other.

dis.date The discovery date of the fire, which is the nearest possible surrogate for the starting time of the fire. This is an object of class `POSIXct` and gives the starting discovery time of the fire to the nearest minute.

dis.julian The discovery date and time of the fire, expressed in ‘Julian days’, i.e. as a decimal fraction representing the number of days since the beginning of the year (midnight 31 December).

out.date The date on which the fire was judged to be ‘out’. This is an object of class `POSIXct` and gives the ‘out’ time of the fire to the nearest minute.

out.julian The date and time at which the fire was judged to be ‘out’, expressed in Julian days.

cause General cause of the fire. This is a factor with levels unknown, rrds (railroads), misc (miscellaneous), ltning (lightning), for.ind (forest industry), incend (incendiary), rec (recreation), resid (resident), and oth.ind (other industry). Causes unknown, ltning, and incend are supposedly designated as ‘final’ by the New Brunswick Department of Natural Resources, meaning (it seems) “that’s all there is to it”. Other causes are apparently intended to be refined by being combined with “source of ignition”. However cross-tabulating cause with ign.src — see below — reveals that very often these three ‘causes’ are associated with an “ignition source” as well.

ign.src Source of ignition, a factor with levels cigs (cigarette/match/pipe/ashes), burn.no.perm (burning without a permit), burn.w.perm (burning with a permit), presc.burn (prescribed burn), wood.spark (wood spark), mach.spark (machine spark), campfire, chainsaw, machinery, veh.acc (vehicle accident), rail.acc (railroad accident), wheelbox (wheelbox on railcars), hot.flakes (hot flakes off railcar wheels), dump.fire (fire escaping from a dump), ashes (ashes, briquettes, burning garbage, etc.)

fnl.size The final size of the fire (area burned) in hectares, to the nearest 10th hectare.

Note that due to data entry errors some of the “out dates” and “out times” in the original data sets were actually *earlier* than the corresponding “discovery dates” and “discover times”. In such cases all corresponding entries of the marks data frame (i.e. `dis.date`, `dis.julian`, `out.date`, and `out.julian`) were set equal to NA. Also, some of the dates and times were missing (equal to NA) in the original data sets.

The ‘ignition source’ data were given as integer codes in the original data sets. The code book that I obtained gave interpretations for codes 1, 2, ..., 15. However the actually also contained codes of 0, 16, 17, 18, and in one instance 44. These may simply be data entry errors. These uninterpretable values were assigned the level unknown. Many of the years had most, or sometimes all, of the ignition source codes equal to 0 (hence turning out as unknown, and many of the years had many missing values as well. These were also assigned the level unknown. Of the 7108 fires in nbfires, 4354 had an unknown ignition source. This variable is hence unlikely to be very useful.

There are also anomalies between cause and ign.src, e.g. cause being unknown but ign.src being cigs, burn.no.perm, mach.spark, hot.flakes, dump.fire or ashes. Particularly worrisome is the fact that the cause ltning (!!!) is associate with sources of ignition cigs, burn.w.perm, presc.burn, and wood.spark.

Source

The data were kindly provided by the New Brunswick Department of Natural Resources. Special thanks are due to Jefferey Betts for a great deal of assistance.

References

Turner, Rolf. Point patterns of forest fire locations. *Environmental and Ecological Statistics* **16** (2009) 197 – 223, doi:10.1007/s10651-007-0085-1.

Thomson, D. B., Mephan, M. P., and Steeves, R. R. (1977) The stereographic double projection. Technical Report 46, University of New Brunswick, Fredericton, N. B., Canada URL: gge.unb.ca/Pubs/Pubs.html.

Examples

```
## Not run:
data(nbfires)
# Get the year 2000 data.
X <- split(nbfires,"year")
Y.00 <- X[["2000"]]
# Plot all of the year 2000 data, marked by fire type.
plot(Y.00,which.marks="fire.type")
# Cut back to forest and grass fires.
Y.00 <- Y.00[marks(Y.00)$fire.type %in% c("forest","grass")]
# Plot the year 2000 forest and grass fires marked by fire duration time.
stt <- marks(Y.00)$dis.julian
fin <- marks(Y.00)$out.julian
marks(Y.00) <- cbind(marks(Y.00),dur=fin-stt)
plot(Y.00,which.marks="dur")
# Look at just the rectangular subwindow (superimposed on the entire window).
nbw.mask <- as.mask(nbfires$window, dimyx=500)
plot(nbw.mask, col=c("green", "white"))
plot(nbfires$window, border="red", add=TRUE)
plot(Y.00[nbw.rect],use.marks=FALSE,add=TRUE)
plot(nbw.rect,add=TRUE,border="blue")
# Look at the K function for the year 2000 forest and grass fires.
K.00 <- Kest(Y.00)
plot(K.00)

## End(Not run)
```

nearest.raster.point *Find Pixel Nearest to a Given Point***Description**

Given cartesian coordinates, find the nearest pixel.

Usage

```
nearest.raster.point(x,y,w, indices=TRUE)
```

Arguments

x	Numeric vector of <i>x</i> coordinates of any points
y	Numeric vector of <i>y</i> coordinates of any points
w	A window (an object of class "owin") of type "mask" representing a binary pixel image.
indices	Logical flag indicating whether to return the row and column indices, or the actual <i>x</i> , <i>y</i> coordinates.

Details

The argument *w* should be a window (an object of class "owin", see [owin.object](#) for details) of type "mask". This represents a binary pixel image.

The arguments *x* and *y* should be numeric vectors of equal length. They are interpreted as the coordinates of points in space. For each point (*x*[*i*], *y*[*i*]), the function finds the nearest pixel in the grid of pixels for *w*.

If *indices*=TRUE, this function returns a list containing two vectors *rr* and *cc* giving row and column positions (in the image matrix). For the location (*x*[*i*], *y*[*i*]) the nearest pixel is at row *rr*[*i*] and column *cc*[*i*] of the image.

If *indices*=FALSE, the function returns a list containing two vectors *x* and *y* giving the actual coordinates of the pixels.

Value

If *indices*=TRUE, a list containing two vectors *rr* and *cc* giving row and column positions (in the image matrix). If *indices*=FALSE, a list containing vectors *x* and *y* giving actual coordinates of the pixels.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[owin.object](#), [as.mask](#)

Examples

```
w <- owin(c(0,1), c(0,1), mask=matrix(TRUE, 100,100)) # 100 x 100 grid
nearest.raster.point(0.5, 0.3, w)
nearest.raster.point(0.5, 0.3, w, indices=FALSE)
```

nearestsegment

Find Line Segment Nearest to Each Point

Description

Given a point pattern and a line segment pattern, this function finds the nearest line segment for each point.

Usage

```
nearestsegment(X, Y)
```

Arguments

- X A point pattern (object of class "ppp").
- Y A line segment pattern (object of class "psp").

Details

The distance between a point x and a straight line segment y is defined to be the shortest Euclidean distance between x and any location on y . This algorithm first calculates the distance from each point of X to each segment of Y . Then it determines, for each point x in X , which segment of Y is closest. The index of this segment is returned.

Value

Integer vector v (of length equal to the number of points in X) identifying the nearest segment to each point. If $v[i] = j$, then $Y[j]$ is the line segment lying closest to $X[i]$.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[project2segment](#) to project each point of X to a point lying on one of the line segments.

Use [distmap.psp](#) to identify the nearest line segment for each pixel in a grid.

Examples

```
X <- runifpoint(3)
Y <- as.psp(matrix(runif(20), 5, 4), window=owin())
v <- nearestsegment(X,Y)
plot(Y)
plot(X, add=TRUE)
plot(X[1], add=TRUE, col="red")
plot(Y[v[1]], add=TRUE, lwd=2, col="red")
```

nnclean*Nearest Neighbour Clutter Removal*

Description

Detect features in a 2D or 3D spatial point pattern using nearest neighbour clutter removal.

Usage

```
nnclean(X, k, ...)
## S3 method for class 'ppp'
nnclean(X, k, ...
         edge.correct = FALSE, wrap = 0.1,
         convergence = 0.001, plothist = FALSE,
         verbose = TRUE, maxit = 50)
## S3 method for class 'pp3'
nnclean(X, k, ...
         convergence = 0.001, plothist = FALSE,
         verbose = TRUE, maxit = 50)
```

Arguments

X	A two-dimensional spatial point pattern (object of class "ppp") or a three-dimensional point pattern (object of class "pp3").
k	Degree of neighbour: k=1 means nearest neighbour, k=2 means second nearest, etc.
...	Ignored.
edge.correct	Logical flag specifying whether periodic edge correction should be performed (only implemented in 2 dimensions).
wrap	Numeric value specifying the relative size of the margin in which data will be replicated for the periodic edge correction (if edge.correct=TRUE). A fraction of window width and window height.
convergence	Tolerance threshold for testing convergence of EM algorithm.
maxit	Maximum number of iterations for EM algorithm.
plothist	Logical flag specifying whether to plot a diagnostic histogram of the nearest neighbour distances and the fitted distribution.
verbose	Logical flag specifying whether to print progress reports.

Details

Byers and Raftery (1998) developed a technique for recognising features in a spatial point pattern in the presence of random clutter.

For each point in the pattern, the distance to the k th nearest neighbour is computed. Then the E-M algorithm is used to fit a mixture distribution to the nearest neighbour distances. The mixture components represent the feature and the clutter. The mixture model can be used to classify each point as belonging to one or other component.

The function *nnclean* is generic, with methods for two-dimensional point patterns (class "ppp") and three-dimensional point patterns (class "pp3") currently implemented.

The result is a point pattern (2D or 3D) with two additional columns of marks:

class A factor, with levels "noise" and "feature", indicating the maximum likelihood classification of each point.

prob Numeric vector giving the estimated probabilities that each point belongs to a feature.

Value

An object of the same kind as X, obtained by attaching marks to the points of X.

Author(s)

Original by Simon Byers and Adrian Raftery. Adapted for **spatstat** by Adrian Baddeley.

References

Byers, S. and Raftery, A.E. (1998) Nearest-neighbour clutter removal for estimating features in spatial point processes. *Journal of the American Statistical Association* **93**, 577–584.

See Also

[nndist](#), [split.ppp](#), [cut.ppp](#)

Examples

```
data(shapley)
X <- nnClean(shapley, k=17)
plot(X, chars=c(".", "+"), cols=1:2)
Y <- split(X)
plot(Y, chars="+", cex=0.5)
marks(X) <- marks(X)$prob
plot(cut(X, breaks=3), chars=c(".", "+", "+"), cols=1:3)
```

Description

Computes nearest-neighbour correlation indices of a marked point pattern, including the nearest-neighbour mark product index (default case of nncorr), the nearest-neighbour mark index (nnmean), and the nearest-neighbour variogram index (nnvario).

Usage

```
nncorr(X,
       f = function(m1, m2) { m1 * m2 },
       ...,
       use = "all.obs", method = c("pearson", "kendall", "spearman"),
       denominator=NULL)
nnmean(X)
nnvario(X)
```

Arguments

X	The observed point pattern. An object of class "ppp".
f	Function f used in the definition of the nearest neighbour correlation. There is a sensible default that depends on the type of marks of X.
...	Extra arguments passed to f.
use,method	Arguments passed to the standard correlation function <code>cor</code> .
denominator	Internal use only.

Details

The nearest neighbour correlation index \bar{n}_f of a marked point process X is a number measuring the dependence between the mark of a typical point and the mark of its nearest neighbour.

The command `nncorr` computes the nearest neighbour correlation index based on any test function f provided by the user. The default behaviour of `nncorr` is to compute the nearest neighbour mark product index. The commands `nnmean` and `nnvario` are convenient abbreviations for other special choices of f .

In the default case, `nncorr(X)` computes three different versions of the nearest-neighbour correlation index: the unnormalised, normalised, and classical correlations.

unnormalised: The **unnormalised** nearest neighbour correlation (Stoyan and Stoyan, 1994, section 14.7) is defined as

$$\bar{n}_f = E[f(M, M^*)]$$

where $E[\cdot]$ denotes mean value, M is the mark attached to a typical point of the point process, and M^* is the mark attached to its nearest neighbour (i.e. the nearest other point of the point process).

Here f is any function $f(m_1, m_2)$ with two arguments which are possible marks of the pattern, and which returns a nonnegative real value. Common choices of f are: for continuous real-valued marks,

$$f(m_1, m_2) = m_1 m_2$$

for discrete marks (multitype point patterns),

$$f(m_1, m_2) = 1(m_1 = m_2)$$

and for marks taking values in $[0, 2\pi]$,

$$f(m_1, m_2) = \sin(m_1 - m_2)$$

For example, in the second case, the unnormalised nearest neighbour correlation \bar{n}_f equals the proportion of points in the pattern which have the same mark as their nearest neighbour.

Note that \bar{n}_f is not a “correlation” in the usual statistical sense. It can take values greater than 1.

normalised: We can define a **normalised** nearest neighbour correlation by

$$\bar{m}_f = \frac{E[f(M, M^*)]}{E[f(M, M')]} \quad (1)$$

where again M is the mark attached to a typical point, M^* is the mark attached to its nearest neighbour, and M' is an independent copy of M with the same distribution. This normalisation is also not a “correlation” in the usual statistical sense, but is normalised so that the value 1 suggests “lack of correlation”: if the marks attached to the points of X are independent and identically distributed, then $\bar{m}_f = 1$. The interpretation of values larger or smaller than 1 depends on the choice of function f .

classical: Finally if the marks of X are real numbers, we can also compute the **classical** correlation, that is, the correlation coefficient of the two random variables M and M^* . The classical correlation has a value between -1 and 1 . Values close to -1 or 1 indicate strong dependence between the marks.

In the default case where f is not given, `nncorr(X)` computes

- If the marks of X are real numbers, the unnormalised and normalised versions of the nearest-neighbour product index $E[M M^*]$, and the classical correlation between M and M^* .
- If the marks of X are factor valued, the unnormalised and normalised versions of the nearest-neighbour equality index $P[M = M^*]$.

The wrapper functions `nnmean` and `nnvario` compute the correlation indices for two special choices of the function $f(m_1, m_2)$.

- `nnmean` computes the correlation indices for $f(m_1, m_2) = m_1$. The unnormalised index is simply the mean value of the mark of the neighbour of a typical point, $E[M^*]$, while the normalised index is $E[M^*]/E[M]$, the ratio of the mean mark of the neighbour of a typical point to the mean mark of a typical point.
- `nnvario` computes the correlation indices for $f(m_1, m_2) = (1/2)(m_1 - m_2)^2$.

The argument X must be a point pattern (object of class "ppp") and must be a marked point pattern. (The marks may be a data frame, containing several columns of mark variables; each column is treated separately.)

If the argument f is given, it must be a function, accepting two arguments $m1$ and $m2$ which are vectors of equal length containing mark values (of the same type as the marks of X). It must return a vector of numeric values of the same length as $m1$ and $m2$. The values must be non-negative.

The arguments `use` and `method` control the calculation of the classical correlation using `cor`, as explained in the help file for `cor`.

Other arguments may be passed to f through the \dots argument.

This algorithm assumes that X can be treated as a realisation of a stationary (spatially homogeneous) random spatial point process in the plane, observed through a bounded window. The window (which is specified in X as $X>window$) may have arbitrary shape. Biases due to edge effects are treated using the 'border method' edge correction.

Value

Labelled vector of length 2 or 3 containing the unnormalised and normalised nearest neighbour correlations, and the classical correlation if appropriate. Alternatively a matrix with 2 or 3 rows, containing this information for each mark variable.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

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

Examples

```
data(finpines)
nncorr(finpines)
# heights of neighbouring trees are slightly negatively correlated

data(amacrine)
nncorr(amacrine)
# neighbouring cells are usually of different type
```

nncross

Nearest Neighbours Between Two Patterns

Description

Given two point patterns X and Y , finds the nearest neighbour in Y of each point of X . Alternatively Y may be a line segment pattern.

Usage

```
nncross(X, Y, iX=NULL, iY=NULL,
        what = c("dist", "which"),
        ...,
        sortby=c("range", "var", "x", "y"),
        is.sorted.X = FALSE,
        is.sorted.Y = FALSE)
```

Arguments

X	Point pattern (object of class "ppp").
Y	Either a point pattern (object of class "ppp") or a line segment pattern (object of class "psp").
iX , iY	Optional identifiers, applicable only in the case where Y is a point pattern, used to determine whether a point in X is identical to a point in Y . See Details.
$what$	Character string specifying what information should be returned. Either the nearest neighbour distance ("dist"), the identifier of the nearest neighbour ("which"), or both.
$sortby$	Determines which coordinate to use to sort the point patterns. See Details.
$is.sorted.X$, $is.sorted.Y$	Logical values attesting whether the point patterns X and Y have been sorted. See Details.
...	Ignored.

Details

Given two point patterns X and Y this function finds, for each point of X , the nearest point of Y . The distance between these points is also computed.

Alternatively if X is a point pattern and Y is a line segment pattern, the function finds the nearest line segment to each point of X , and computes the distance.

The return value is a data frame, with rows corresponding to the points of X. The first column gives the nearest neighbour distances (i.e. the i th entry is the distance from the i th point of X to the nearest element of Y). The second column gives the indices of the nearest neighbours (i.e. the i th entry is the index of the nearest element in Y.) If `what="dist"` then only the vector of distances is returned. If `what="which"` then only the vector of indices is returned.

Note that this function is not symmetric in X and Y. To find the nearest neighbour in X of each point in Y, where Y is a point pattern, use `nncross(Y, X)`.

The arguments `iX` and `iY` are used when the two point patterns X and Y have some points in common. In this situation `nncross(X, Y)` would return some zero distances. To avoid this, attach a unique integer identifier to each point, such that two points are identical if their identifying numbers are equal. Let `iX` be the vector of identifier values for the points in X, and `iY` the vector of identifiers for points in Y. Then the code will only compare two points if they have different values of the identifier. See the Examples.

Value

By default (if `what=c("dist", "which")`) a data frame with two columns:

<code>dist</code>	Nearest neighbour distance
<code>which</code>	Nearest neighbour index in Y

If `what="dist"`, a vector of nearest neighbour distances.

If `what="which"`, a vector of nearest neighbour indices.

Sorting data and pre-sorted data

For efficiency, the algorithm sorts the point patterns X and Y into increasing order of the x coordinate or increasing order of the the y coordinate. By default (if `sortby="range"`), the sorting will occur on the coordinate that has the larger range of values (according to the frame of the enclosing window of Y). If `sortby = "var"`, sorting will occur on the coordinate that has the greater variance (in the pattern Y). Setting `sortby="x"` or `sortby = "y"` will specify that sorting should occur on the x or y coordinate, respectively.

If the point pattern X is already sorted, then the corresponding argument `is.sorted.X` should be set to TRUE, and `sortby` should be set equal to "x" or "y" to indicate which coordinate is sorted.

Similarly if Y is already sorted, then `is.sorted.Y` should be set to TRUE, and `sortby` should be set equal to "x" or "y" to indicate which coordinate is sorted.

If both X and Y are sorted *on the same coordinate axis* then both `is.sorted.X` and `is.sorted.Y` should be set to TRUE, and `sortby` should be set equal to "x" or "y" to indicate which coordinate is sorted.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>, Rolf Turner <r.turner@auckland.ac.nz>, and Jens Oehlschlaegel

See Also

[nnndist](#) for nearest neighbour distances in a single point pattern.

Examples

```
# two different point patterns
X <- runifpoint(15)
Y <- runifpoint(20)
N <- nncross(X,Y)$which
# note that length(N) = 15
plot(superimpose(X=X, Y=Y), main="nncross", cols=c("red","blue"))
arrows(X$x, X$y, Y[N]$x, Y[N]$y, length=0.15)

# two patterns with some points in common
Z <- runifpoint(50)
X <- Z[1:30]
Y <- Z[20:50]
iX <- 1:30
iY <- 20:50
N <- nncross(X,Y, iX, iY)$which
N <- nncross(X,Y, iX, iY, what="which") #faster
plot(superimpose(X=X, Y=Y), main="nncross", cols=c("red","blue"))
arrows(X$x, X$y, Y[N]$x, Y[N]$y, length=0.15)

# point pattern and line segment pattern
X <- runifpoint(15)
Y <- rpoisline(10)
N <- nncross(X,Y)
```

nndist

Nearest neighbour distances

Description

Computes the distance from each point to its nearest neighbour in a point pattern. Alternatively computes the distance to the second nearest neighbour, or third nearest, etc.

Usage

```
nndist(X, ...)
## S3 method for class 'ppp'
nndist(X, ..., k=1, method="C")
## Default S3 method:
nndist(X, Y=NULL, ..., k=1, method="C")
```

Arguments

X, Y	Arguments specifying the locations of a set of points. For <i>nndist.ppp</i> , the argument X should be a point pattern (object of class "ppp"). For <i>nndist.default</i> , typically X and Y would be numeric vectors of equal length. Alternatively Y may be omitted and X may be a list with two components x and y, or a matrix with two columns.
...	Ignored by <i>nndist.ppp</i> and <i>nndist.default</i> .
k	Integer, or integer vector. The algorithm will compute the distance to the kth nearest neighbour.
method	String specifying which method of calculation to use. Values are "C" and "interpreted".

Details

This function computes the Euclidean distance from each point in a point pattern to its nearest neighbour (the nearest other point of the pattern). If k is specified, it computes the distance to the k th nearest neighbour.

The function `nndist` is generic, with a method for point patterns (objects of class "ppp"), and a default method for coordinate vectors. There is also a method for line segment patterns, `nndist.psp`.

The method for point patterns expects a single point pattern argument X and returns the vector of its nearest neighbour distances.

The default method expects that X and Y will determine the coordinates of a set of points. Typically X and Y would be numeric vectors of equal length. Alternatively Y may be omitted and X may be a list with two components named x and y , or a matrix or data frame with two columns.

The argument k may be a single integer, or an integer vector. If it is a vector, then the k th nearest neighbour distances are computed for each value of k specified in the vector.

The argument `method` is not normally used. It is retained only for checking the validity of the software. If `method = "interpreted"` then the distances are computed using interpreted R code only. If `method="C"` (the default) then C code is used. The C code is faster by two to three orders of magnitude and uses much less memory.

If there is only one point (if x has length 1), then a nearest neighbour distance of `Inf` is returned. If there are no points (if x has length zero) a numeric vector of length zero is returned.

To identify *which* point is the nearest neighbour of a given point, use `nnwhich`.

To use the nearest neighbour distances for statistical inference, it is often advisable to use the edge-corrected empirical distribution, computed by `Gest`.

To find the nearest neighbour distances from one point pattern to another point pattern, use `nncross`.

Value

Numeric vector or matrix containing the nearest neighbour distances for each point.

If $k = 1$ (the default), the return value is a numeric vector v such that $v[i]$ is the nearest neighbour distance for the i th data point.

If k is a single integer, then the return value is a numeric vector v such that $v[i]$ is the k th nearest neighbour distance for the i th data point.

If k is a vector, then the return value is a matrix m such that $m[i, j]$ is the $k[j]$ th nearest neighbour distance for the i th data point.

Warnings

An infinite or NA value is returned if the distance is not defined (e.g. if there is only one point in the point pattern).

Author(s)

Pavel Grabarnik <pavel.grabar@issp.serpukhov.su> and Adrian Baddeley <Adrian.Baddeley@csiro.au>
<http://www.maths.uwa.edu.au/~adrian/>

See Also

`nndist.psp`, `pairdist`, `Gest`, `nnwhich`, `nncross`.

Examples

```

data(cells)
# nearest neighbours
d <- nndist(cells)

# second nearest neighbours
d2 <- nndist(cells, k=2)

# first, second and third nearest
d1to3 <- nndist(cells, k=1:3)

x <- runif(100)
y <- runif(100)
d <- nndist(x, y)

# Stienen diagram
plot(cells %mark% (nndist(cells)/2), markscale=1)

```

nndist.pp3

Nearest neighbour distances in three dimensions

Description

Computes the distance from each point to its nearest neighbour in a three-dimensional point pattern. Alternatively computes the distance to the second nearest neighbour, or third nearest, etc.

Usage

```

## S3 method for class 'pp3'
nndist(X, ..., k=1)

```

Arguments

X	Three-dimensional point pattern (object of class "pp3").
...	Ignored.
k	Integer, or integer vector. The algorithm will compute the distance to the kth nearest neighbour.

Details

This function computes the Euclidean distance from each point in a three-dimensional point pattern to its nearest neighbour (the nearest other point of the pattern). If k is specified, it computes the distance to the kth nearest neighbour.

The function `nndist` is generic; this function `nndist.pp3` is the method for the class "pp3".

The argument k may be a single integer, or an integer vector. If it is a vector, then the kth nearest neighbour distances are computed for each value of k specified in the vector.

If there is only one point (if x has length 1), then a nearest neighbour distance of Inf is returned. If there are no points (if x has length zero) a numeric vector of length zero is returned.

To identify which point is the nearest neighbour of a given point, use [nnwhich](#).

To use the nearest neighbour distances for statistical inference, it is often advisable to use the edge-corrected empirical distribution, computed by [Gest](#).

To find the nearest neighbour distances from one point pattern to another point pattern, use [nncross](#).

Value

Numeric vector or matrix containing the nearest neighbour distances for each point.

If $k = 1$ (the default), the return value is a numeric vector v such that $v[i]$ is the nearest neighbour distance for the i th data point.

If k is a single integer, then the return value is a numeric vector v such that $v[i]$ is the k th nearest neighbour distance for the i th data point.

If k is a vector, then the return value is a matrix m such that $m[i, j]$ is the $k[j]$ th nearest neighbour distance for the i th data point.

Warnings

An infinite or NA value is returned if the distance is not defined (e.g. if there is only one point in the point pattern).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> based on code for two dimensions by Pavel Grabarnik

See Also

[nndist](#), [pairdist](#), [G3est](#), [nnwhich](#)

Examples

```
X <- runifpoint3(40)

# nearest neighbours
d <- nndist(X)

# second nearest neighbours
d2 <- nndist(X, k=2)

# first, second and third nearest
d1to3 <- nndist(X, k=1:3)
```

Description

Computes the distance from each point to its nearest neighbour in a multi-dimensional point pattern. Alternatively computes the distance to the second nearest neighbour, or third nearest, etc.

Usage

```
## S3 method for class 'ppx'
nndist(X, ..., k=1)
```

Arguments

- `x` Multi-dimensional point pattern (object of class "ppx").
- `...` Arguments passed to `coords.ppx` to determine which coordinates should be used.
- `k` Integer, or integer vector. The algorithm will compute the distance to the k th nearest neighbour.

Details

This function computes the Euclidean distance from each point in a multi-dimensional point pattern to its nearest neighbour (the nearest other point of the pattern). If k is specified, it computes the distance to the k th nearest neighbour.

The function `nndist` is generic; this function `nndist.ppx` is the method for the class "ppx".

The argument k may be a single integer, or an integer vector. If it is a vector, then the k th nearest neighbour distances are computed for each value of k specified in the vector.

If there is only one point (if `x` has length 1), then a nearest neighbour distance of `Inf` is returned. If there are no points (if `x` has length zero) a numeric vector of length zero is returned.

To identify *which* point is the nearest neighbour of a given point, use `nnwhich`.

To find the nearest neighbour distances from one point pattern to another point pattern, use `nncross`.

By default, both spatial and temporal coordinates are extracted. To obtain the spatial distance between points in a space-time point pattern, set `temporal=FALSE`.

Value

Numeric vector or matrix containing the nearest neighbour distances for each point.

If $k = 1$ (the default), the return value is a numeric vector `v` such that `v[i]` is the nearest neighbour distance for the i th data point.

If k is a single integer, then the return value is a numeric vector `v` such that `v[i]` is the k th nearest neighbour distance for the i th data point.

If k is a vector, then the return value is a matrix `m` such that `m[i, j]` is the $k[j]$ th nearest neighbour distance for the i th data point.

Warnings

An infinite or NA value is returned if the distance is not defined (e.g. if there is only one point in the point pattern).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

See Also

`nndist`, `pairdist`, `nnwhich`

Examples

```
df <- data.frame(x=runif(5),y=runif(5),z=runif(5),w=runif(5))
X <- ppx(data=df)

# nearest neighbours
d <- nndist(X)

# second nearest neighbours
d2 <- nndist(X, k=2)

# first, second and third nearest
d1to3 <- nndist(X, k=1:3)
```

nndist.psp

Nearest neighbour distances between line segments

Description

Computes the distance from each line segment to its nearest neighbour in a line segment pattern. Alternatively finds the distance to the second nearest, third nearest etc.

Usage

```
## S3 method for class 'psp'
nndist(X, ..., k=1, method="Fortran")
```

Arguments

- X A line segment pattern (object of class "psp").
- ... Ignored.
- k Integer, or integer vector. The algorithm will compute the distance to the kth nearest neighbour.
- method String specifying which method of calculation to use. Values are "Fortran" and "interpreted". Usually not specified.

Details

This is a method for the generic function [nndist](#) for the class "psp".

If $k=1$, this function computes the distance from each line segment to the nearest other line segment in X. In general it computes the distance from each line segment to the kth nearest other line segment. The argument k can also be a vector, and this computation will be performed for each value of k.

Distances are calculated using the Hausdorff metric. The Hausdorff distance between two line segments is the maximum distance from any point on one of the segments to the nearest point on the other segment.

If there are fewer than $\max(k)+1$ line segments in the pattern, some of the nearest neighbour distances will be infinite (Inf).

The argument method is not normally used. It is retained only for checking the validity of the software. If method = "interpreted" then the distances are computed using interpreted R code only. If method="Fortran" (the default) then Fortran code is used. The Fortran code is somewhat faster.

Value

Numeric vector or matrix containing the nearest neighbour distances for each line segment.

If $k = 1$ (the default), the return value is a numeric vector v such that $v[i]$ is the nearest neighbour distance for the i th segment.

If k is a single integer, then the return value is a numeric vector v such that $v[i]$ is the k th nearest neighbour distance for the i th segment.

If k is a vector, then the return value is a matrix m such that $m[i, j]$ is the $k[j]$ th nearest neighbour distance for the i th segment.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[nndist](#), [nndist.ppp](#)

Examples

```
L <- psp(runif(10), runif(10), runif(10), runif(10), owin())
D <- nndist(L)
D <- nndist(L, k=1:3)
```

nnwhich

Nearest neighbour

Description

Finds the nearest neighbour of each point in a point pattern.

Usage

```
nnwhich(X, ...)
## S3 method for class 'ppp'
nnwhich(X, ..., k=1, method="C")
## Default S3 method:
nnwhich(X, Y=NULL, ..., k=1, method="C")
```

Arguments

X, Y	Arguments specifying the locations of a set of points. For <i>nnwhich.ppp</i> , the argument X should be a point pattern (object of class "ppp"). For <i>nnwhich.default</i> , typically X and Y would be numeric vectors of equal length. Alternatively Y may be omitted and X may be a list with two components x and y , or a matrix with two columns.
\dots	Ignored by <i>nnwhich.ppp</i> and <i>nnwhich.default</i> .
k	Integer, or integer vector. The algorithm will compute the distance to the k th nearest neighbour.
<i>method</i>	String specifying which method of calculation to use. Values are "C" and "interpreted".

Details

For each point in the given point pattern, this function finds its nearest neighbour (the nearest other point of the pattern). By default it returns a vector giving, for each point, the index of the point's nearest neighbour. If k is specified, the algorithm finds each point's k th nearest neighbour.

The function `nnwhich` is generic, with method for point patterns (objects of class "ppp") and a default method which are described here, as well as a method for three-dimensional point patterns (objects of class "pp3", described in [nnwhich.pp3](#)).

The method `nnwhich.ppp` expects a single point pattern argument X . The default method expects that X and Y will determine the coordinates of a set of points. Typically X and Y would be numeric vectors of equal length. Alternatively Y may be omitted and X may be a list with two components named x and y , or a matrix or data frame with two columns.

The argument k may be a single integer, or an integer vector. If it is a vector, then the k th nearest neighbour distances are computed for each value of k specified in the vector.

If there are no points (if x has length zero) a numeric vector of length zero is returned. If there is only one point (if x has length 1), then the nearest neighbour is undefined, and a value of NA is returned. In general if the number of points is less than or equal to k , then a vector of NA's is returned.

The argument `method` is not normally used. It is retained only for checking the validity of the software. If `method = "interpreted"` then the distances are computed using interpreted R code only. If `method="C"` (the default) then C code is used. The C code is faster by two to three orders of magnitude and uses much less memory.

To evaluate the *distance* between a point and its nearest neighbour, use [nndist](#).

To find the nearest neighbours from one point pattern to another point pattern, use [nncross](#).

Value

Numeric vector or matrix giving, for each point, the index of its nearest neighbour (or k th nearest neighbour).

If $k = 1$ (the default), the return value is a numeric vector v giving the indices of the nearest neighbours (the nearest neighbour of the i th point is the j th point where $j = v[i]$).

If k is a single integer, then the return value is a numeric vector giving the indices of the k th nearest neighbours.

If k is a vector, then the return value is a matrix m such that $m[i, j]$ is the index of the $k[j]$ th nearest neighbour for the i th data point.

Warnings

A value of NA is returned if there is only one point in the point pattern.

Author(s)

Pavel Grabarnik <pavel.grabar@issp.serpukhov.su> and Adrian Baddeley <Adrian.Baddeley@csiro.au>
<http://www.maths.uwa.edu.au/~adrian/>

See Also

[nndist](#), [nncross](#)

Examples

```

data(cells)
plot(cells)
m <- nnwhich(cells)
m2 <- nnwhich(cells, k=2)

# plot nearest neighbour links
b <- cells[m]
arrows(cells$x, cells$y, b$x, b$y, angle=15, length=0.15, col="red")

# find points which are the neighbour of their neighbour
self <- (m[m] == seq(m))
# plot them
A <- cells[self]
B <- cells[m[self]]
plot(cells)
segments(A$x, A$y, B$x, B$y)

```

nnwhich.pp3

Nearest neighbours in three dimensions

Description

Finds the nearest neighbour of each point in a three-dimensional point pattern.

Usage

```
## S3 method for class 'pp3'
nnwhich(X, ..., k=1)
```

Arguments

X	Three-dimensional point pattern (object of class "pp3").
...	Ignored.
k	Integer, or integer vector. The algorithm will compute the distance to the kth nearest neighbour.

Details

For each point in the given three-dimensional point pattern, this function finds its nearest neighbour (the nearest other point of the pattern). By default it returns a vector giving, for each point, the index of the point's nearest neighbour. If k is specified, the algorithm finds each point's kth nearest neighbour.

The function `nnwhich` is generic. This is the method for the class "pp3".

If there are no points in the pattern, a numeric vector of length zero is returned. If there is only one point, then the nearest neighbour is undefined, and a value of NA is returned. In general if the number of points is less than or equal to k, then a vector of NA's is returned.

To evaluate the *distance* between a point and its nearest neighbour, use [nndist](#).

To find the nearest neighbours from one point pattern to another point pattern, use [nncross](#).

Value

Numeric vector or matrix giving, for each point, the index of its nearest neighbour (or kth nearest neighbour).

If $k = 1$ (the default), the return value is a numeric vector v giving the indices of the nearest neighbours (the nearest neighbour of the i th point is the j th point where $j = v[i]$).

If k is a single integer, then the return value is a numeric vector giving the indices of the k th nearest neighbours.

If k is a vector, then the return value is a matrix m such that $m[i, j]$ is the index of the $k[j]$ th nearest neighbour for the i th data point.

Warnings

A value of NA is returned if there is only one point in the point pattern.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> based on two-dimensional code by Pavel Grabarnik

See Also

[nnwhich](#), [nndist](#), [nncross](#)

Examples

```
X <- runifpoint3(30)
m <- nnwhich(X)
m2 <- nnwhich(X, k=2)
```

nnwhich.ppx

Nearest Neighbours in Any Dimensions

Description

Finds the nearest neighbour of each point in a multi-dimensional point pattern.

Usage

```
## S3 method for class 'ppx'
nnwhich(X, ..., k=1)
```

Arguments

- X Multi-dimensional point pattern (object of class "ppx").
- \dots Arguments passed to [coords.ppx](#) to determine which coordinates should be used.
- k Integer, or integer vector. The algorithm will compute the distance to the k th nearest neighbour.

Details

For each point in the given multi-dimensional point pattern, this function finds its nearest neighbour (the nearest other point of the pattern). By default it returns a vector giving, for each point, the index of the point's nearest neighbour. If k is specified, the algorithm finds each point's k th nearest neighbour.

The function `nnwhich` is generic. This is the method for the class "`ppx`".

If there are no points in the pattern, a numeric vector of length zero is returned. If there is only one point, then the nearest neighbour is undefined, and a value of `NA` is returned. In general if the number of points is less than or equal to k , then a vector of `NA`'s is returned.

To evaluate the *distance* between a point and its nearest neighbour, use `nndist`.

To find the nearest neighbours from one point pattern to another point pattern, use `nncross`.

By default, both spatial and temporal coordinates are extracted. To obtain the spatial distance between points in a space-time point pattern, set `temporal=FALSE`.

Value

Numeric vector or matrix giving, for each point, the index of its nearest neighbour (or k th nearest neighbour).

If $k = 1$ (the default), the return value is a numeric vector v giving the indices of the nearest neighbours (the nearest neighbour of the i th point is the j th point where $j = v[i]$).

If k is a single integer, then the return value is a numeric vector giving the indices of the k th nearest neighbours.

If k is a vector, then the return value is a matrix m such that $m[i, j]$ is the index of the $k[j]$ th nearest neighbour for the i th data point.

Warnings

A value of `NA` is returned if there is only one point in the point pattern.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

See Also

`nnwhich`, `nndist`, `nncross`

Examples

```
df <- data.frame(x=runif(5),y=runif(5),z=runif(5),w=runif(5))
X <- ppx(data=df)
m <- nnwhich(X)
m2 <- nnwhich(X, k=2)
```

nptfun*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

```
nptfun(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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> Ege Rubak and Jesper Moller.

See Also

[psst](#)

Examples

```
data(cells)
fit0 <- ppm(cells, ~1)
v <- psst(fit0, nptfun)
```

<code>npoints</code>	<i>Number of Points in a Point Pattern</i>
----------------------	--

Description

Returns the number of points in a point pattern of any kind.

Usage

```
npoints(x)
## S3 method for class 'ppp'
npoints(x)
## S3 method for class 'pp3'
npoints(x)
## S3 method for class 'ppx'
npoints(x)
```

Arguments

- | | |
|----------------|---|
| <code>x</code> | A point pattern (object of class "ppp", "pp3", "ppx" or some other suitable class). |
|----------------|---|

Details

This function returns the number of points in a point pattern. The function `npoints` is generic with methods for the classes "ppp", "pp3", "ppx" and possibly other classes.

Value

Integer.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`ppp.object`, `print.pp3`, `print.ppx`.

Examples

```
data(cells)
npoints(cells)
```

nsegments*Number of Line Segments in a Line Segment Pattern*

Description

Returns the number of line segments in a line segment pattern.

Usage

```
nsegments(x)
## S3 method for class 'psp'
nsegments(x)
```

Arguments

x A line segment pattern, i.e. an object of class psp.

Details

This function is generic, but there is at present only one method, that for class psp.

Value

Integer.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[npoints\(\)](#), [psp.object\(\)](#)

Examples

```
data(copper)
nsegments(copper$Lines)
nsegments(copper$SouthLines)
```

nztrees

*New Zealand Trees Point Pattern***Description**

The data give the locations of trees in a forest plot.

They were collected by Mark and Esler (1970) and were extracted and analysed by Ripley (1981, pp. 169–175). They represent the positions of 86 trees in a forest plot approximately 140 by 85 feet.

Ripley discarded from his analysis the eight trees at the right-hand edge of the plot (which appear to be part of a planted border) and trimmed the window by a 5-foot margin accordingly.

Usage

```
data(nztrees)
```

Format

An object of class "ppp" representing the point pattern of tree locations. The Cartesian coordinates are in feet.

See [ppp.object](#) for details of the format of a point pattern object.

Note

To trim a 5-foot margin off the window, type `nzsub <- nztrees[, owin(c(0,148),c(0,95))]`

Source

Mark and Esler (1970), Ripley (1981).

References

- Ripley, B.D. (1981) *Spatial statistics*. John Wiley and Sons.
 Mark, A.F. and Esler, A.E. (1970) An assessment of the point-centred quarter method of plotless sampling in some New Zealand forests. *Proceedings of the New Zealand Ecological Society* **17**, 106–110.

opening

*Morphological Opening***Description**

Perform morphological opening of a window, a line segment pattern or a point pattern.

Usage

```
opening(w, r, ...)
## S3 method for class 'owin'
opening(w, r, ..., polygonal=NULL)
## S3 method for class 'ppp'
opening(w, r, ...)
## S3 method for class 'psp'
opening(w, r, ...)
```

Arguments

w	A window (object of class "owin" or a line segment pattern (object of class "psp") or a point pattern (object of class "ppp").
r	positive number: the radius of the opening.
...	extra arguments passed to <code>as.mask</code> controlling the pixel resolution, if a pixel approximation is used
polygonal	Logical flag indicating whether to compute a polygonal approximation to the erosion (polygonal=TRUE) or a pixel grid approximation (polygonal=FALSE).

Details

The morphological opening (Serra, 1982) of a set W by a distance $r > 0$ is the subset of points in W that can be separated from the boundary of W by a circle of radius r . That is, a point x belongs to the opening if it is possible to draw a circle of radius r (not necessarily centred on x) that has x on the inside and the boundary of W on the outside. The opened set is a subset of W .

For a small radius r , the opening operation has the effect of smoothing out irregularities in the boundary of W . For larger radii, the opening operation removes promontories in the boundary. For very large radii, the opened set is empty.

The algorithm applies `erosion` followed by `dilation`.

Value

If $r > 0$, an object of class "owin" representing the opened region. If $r=0$, the result is identical to w .

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

Serra, J. (1982) Image analysis and mathematical morphology. Academic Press.

See Also

`closing` for the opposite operation.
`dilation`, `erosion` for the basic operations.
`owin`, `as.owin` for information about windows.

Examples

```
data(letterR)
v <- opening(letterR, 0.3, dimyx=256)
plot(v, main="opening")
plot(letterR, add=TRUE)
```

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.
- Ord, J.K. (1977) Contribution to the discussion of Ripley (1977).
- Ord, J.K. (1978) How many trees in a forest? *Mathematical Scientist* **3**, 23–33.
- Ripley, B.D. (1977) Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, Series B*, **39**, 172 – 212.

See Also

[ppm](#), [ppm.object](#), [OrdThresh](#)

ord.family

Ord Interaction Process Family

Description

An object describing the family of all Ord interaction point processes

Details**Advanced Use Only!**

This structure would not normally be touched by the user. It describes the family of point process models introduced by Ord (1977).

If you need to create a specific Ord-type model for use in analysis, use the function [OrdThresh](#) or [Ord](#).

Anyway, `ord.family` is an object of class "isf" containing a function `ord.family$eval` for evaluating the sufficient statistics of any Ord type point process model taking an exponential family form.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.
- Ord, J.K. (1977) Contribution to the discussion of Ripley (1977).
- Ord, J.K. (1978) How many trees in a forest? *Mathematical Scientist* **3**, 23–33.
- Ripley, B.D. (1977) Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, Series B*, **39**, 172 – 212.

See Also

[pairwise.family](#), [pairsat.family](#), [Poisson](#), [Pairwise](#), [PairPiece](#), [Strauss](#), [StraussHard](#), [Softcore](#), [Geyer](#), [SatPiece](#), [Saturated](#), [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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.
- Ord, J.K. (1977) Contribution to the discussion of Ripley (1977).
- Ord, J.K. (1978) How many trees in a forest? *Mathematical Scientist* **3**, 23–33.
- Ripley, B.D. (1977) Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, Series B*, **39**, 172 – 212.

See Also

[ppm](#), [ppm.object](#)

osteo*Osteocyte Lacunae Data: Replicated Three-Dimensional Point Patterns*

Description

These data give the three-dimensional locations of osteocyte lacunae observed in rectangular volumes of solid bone using a confocal microscope.

There were four samples of bone, and ten regions were mapped in each bone, yielding 40 spatial point patterns. The data can be regarded as replicated observations of a three-dimensional point process, nested within bone samples.

Usage

```
data(osteo)
```

Format

A [hyperframe](#) with the following columns:

<code>id</code>	character string identifier of bone sample
<code>shortid</code>	last numeral in <code>id</code>
<code>brick</code>	serial number (1 to 10) of sampling volume within this bone sample
<code>pts</code>	three dimensional point pattern (class <code>pp3</code>)
<code>depth</code>	the depth of the brick in microns

Details

These data are three-dimensional point patterns representing the positions of *osteocyte lacunae*, holes in bone which were occupied by osteocytes (bone-building cells) during life.

Observations were made on four different skulls of Macaque monkeys using a three-dimensional microscope. From each skull, observations were collected in 10 separate sampling volumes. In all, there are 40 three-dimensional point patterns in the dataset.

The data were collected in 1984 by A. Baddeley, A. Boyde, C.V. Howard and S. Reid (see references) using the tandem-scanning reflected light microscope (TSRLM) at University College London. This was one of the first optical confocal microscopes available.

Each point pattern dataset gives the (x, y, z) coordinates (in microns) of all points visible in a three-dimensional rectangular box (“brick”) of dimensions $81 \times 100 \times d$ microns, where d varies. The z coordinate is depth into the bone (depth of the focal plane of the confocal microscope); the (x, y) plane is parallel to the exterior surface of the bone; the relative orientation of the x and y axes is not important.

The bone samples were three intact skulls and one skull cap, all originally identified as belonging to the macaque monkey *Macaca fascicularis*, from the collection of the Department of Anatomy, University of London. Later analysis (Baddeley et al, 1993) suggested that the skull cap, given here as the first animal, was a different subspecies, and this was confirmed by anatomical inspection.

Sampling Procedure

The following extract from Baddeley et al (1987) describes the sampling procedure.

The parietal bones of three fully articulated adult Macaque monkey (*Macaca fascicularis*) skulls from the collection of University College London were used. The right parietal bone was examined, in each case, approximately 1 cm lateral to the sagittal suture and 2 cm posterior to the coronal suture. The skulls were mounted on plasticine on a moving stage placed beneath the TSRLM. Immersion oil was applied and a $\times 60$, NA 1.0 oil immersion objective lens (Lomo) was focussed at 10 microns below the cranial surface. The TV image was produced by a Panasonic WB 1850/B camera on a Sony PVM 90CE TV monitor.

A graduated rectangular counting frame 90×110 mm (representing 82×100 microns in real units) was marked on a Perspex overlay and fixed to the screen. The area of tissue seen within the frame defined a subfield: a guard area of 10 mm width was visible on all sides of the frame. Ten subfields were examined, arranged approximately in a rectangular grid pattern, with at least one field width separating each pair of fields. The initial field position was determined randomly by applying a randomly-generated coordinate shift to the moving stage. Subsequent fields were attained using the coarse controls of the microscope stage, in accordance with the rectangular grid pattern.

For each subfield, the focal plane was racked down from its initial 10 micron depth until all visible osteocyte lacunae had been examined. This depth d was recorded. The 3-dimensional sampling volume was therefore a rectangular box of dimensions $82 \times 100 \times d$ microns, called a "brick". For each visible lacuna, the fine focus racking control was adjusted until maximum brightness was obtained. The depth of the focal plane was then recorded as the \$z\$ coordinate of the "centre point" of the lacuna. Without moving the focal plane, the x and y coordinates of the centre of the lacunar image were read off the graduated counting frame. This required a subjective judgement of the position of the centre of the 2-dimensional image. Profiles were approximately elliptical and the centre was considered to be well-defined. Accuracy of the recording procedure was tested by independent repetition (by the same operator and by different operators) and found to be reproducible to plus or minus 2 mm on the screen.

A lacuna was counted only if its (x, y) coordinates lay inside the 90×110 mm counting frame.

Source

Adrian Baddeley.

References

- Baddeley, A.J., Howard, C.V., Boyde, A. and Reid, S.A. (1987) Three dimensional analysis of the spatial distribution of particles using the tandem-scanning reflected light microscope. *Acta Stereologica* **6** (supplement II) 87–100.
- Baddeley, A.J., Moyeed, R.A., Howard, C.V. and Boyde, A. (1993) Analysis of a three-dimensional point pattern with replication. *Applied Statistics* **42** (1993) 641–668.
- Howard, C.V. and Reid, S. and Baddeley, A.J. and Boyde, A. (1985) Unbiased estimation of particle density in the tandem-scanning reflected light microscope. *Journal of Microscopy* **138** 203–212.

Examples

```
data(osteо)
osteо
## Not run:
plot(osteо$pts[[1]], main="animal 1, brick 1")
ape1 <- osteо[osteо$shortid==4, ]
plot(ape1, tick.marks=FALSE)
with(osteо, summary(pts)$intensity)
plot(with(ape1, K3est(pts)))

## End(Not run)
```

`owin`*Create a Window*

Description

Creates an object of class "owin" representing an observation window in the two-dimensional plane

Usage

```
owin(xrange=c(0,1), yrangle=c(0,1), ..., poly=NULL, mask=NULL,  
unitname=NULL, xy=NULL)
```

Arguments

<code>xrange</code>	<i>x</i> coordinate limits of enclosing box
<code>yrangle</code>	<i>y</i> coordinate limits of enclosing box
<code>...</code>	Ignored.
<code>poly</code>	Optional. Polygonal boundary of window. Incompatible with <code>mask</code> .
<code>mask</code>	Optional. Logical matrix giving binary image of window. Incompatible with <code>poly</code> .
<code>unitname</code>	Optional. Name of unit of length. Either a single character string, or a vector of two character strings giving the singular and plural forms, respectively.
<code>xy</code>	Optional. List with components <code>x</code> and <code>y</code> specifying the pixel coordinates for <code>mask</code> .

Details

In the **spatstat** library, a point pattern dataset must include information about the window of observation. This is represented by an object of class "owin". See [owin.object](#) for an overview.

To create a window in its own right, users would normally invoke `owin`, although sometimes `as.owin` may be convenient.

A window may be rectangular, polygonal, or a mask (a binary image).

- **rectangular windows:** If only `xrange` and `yrangle` are given, then the window will be rectangular, with its *x* and *y* coordinate dimensions given by these two arguments (which must be vectors of length 2). If no arguments are given at all, the default is the unit square with `xrange=c(0,1)` and `yrangle=c(0,1)`.
- **polygonal windows:** If `poly` is given, then the window will be polygonal.
 - *single polygon:* If `poly` is a matrix or data frame with two columns, or a structure with two component vectors `x` and `y` of equal length, then these values are interpreted as the cartesian coordinates of the vertices of a polygon circumscribing the window. The vertices must be listed *anticlockwise*. No vertex should be repeated (i.e. do not repeat the first vertex).
 - *multiple polygons or holes:* If `poly` is a list, each entry `poly[[i]]` of which is a matrix or data frame with two columns or a structure with two component vectors `x` and `y` of equal length, then the successive list members `poly[[i]]` are interpreted as separate polygons which together make up the boundary of the window. The vertices of each polygon must be listed *anticlockwise* if the polygon is part of the external boundary, but *clockwise* if the polygon is the boundary of a hole in the window. Again, do not repeat any vertex.

- **binary masks:** If `mask` is given, then the window will be a binary image. The argument `mask` should be a logical matrix such that `mask[i,j]` is TRUE if the point $(x[j], y[i])$ belongs to the window, and FALSE if it does not. Note carefully that rows of `mask` correspond to the y coordinate, and columns to the x coordinate. Here x and y are vectors of x and y coordinates equally spaced over `xrange` and `yrange` respectively. The pixel coordinate vectors x and y may be specified explicitly using the argument `xy`, which should be a list containing components x and y . Alternatively there is a sensible default.

To create a window which is mathematically defined by inequalities in the Cartesian coordinates, use `raster.x()` and `raster.y()` as in the examples below.

Functions `square` and `disc` will create square and circular windows, respectively.

Value

An object of class "owin" describing a window in the two-dimensional plane.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`square`, `disc`, `owin.object`, `as.owin`, `complement.owin`, `ppp.object`, `ppp`

Examples

```
w <- owin()
w <- owin(c(0,1), c(0,1))
# the unit square

w <- owin(c(10,20), c(10,30), unitname=c("foot","feet"))
# a rectangle of dimensions 10 x 20 feet
# with lower left corner at (10,10)

# polygon (diamond shape)
w <- owin(poly=list(x=c(0.5,1,0.5,0),y=c(0,1,2,1)))
w <- owin(c(0,1), c(0,2), poly=list(x=c(0.5,1,0.5,0),y=c(0,1,2,1)))

# polygon with hole
ho <- owin(poly=list(list(x=c(0,1,1,0), y=c(0,0,1,1)),
list(x=c(0.6,0.4,0.4,0.6), y=c(0.2,0.2,0.4,0.4)))))

w <- owin(c(-1,1), c(-1,1), mask=matrix(TRUE, 100,100))
# 100 x 100 image, all TRUE
X <- raster.x(w)
Y <- raster.y(w)
wm <- owin(w$xrange, w$yrange, mask=(X^2 + Y^2 <= 1))
# discrete approximation to the unit disc

## Not run:
plot(c(0,1),c(0,1),type="n")
bdry <- locator()
# click the vertices of a polygon (anticlockwise)

## End(Not run)
```

```
w <- owin(poly=bdry)
## Not run: plot(w)

## Not run:
im <- as.logical(matrix(scan("myfile"), nrow=128, ncol=128))
# read in an arbitrary 128 x 128 digital image from text file
rim <- im[, 128:1]
# Assuming it was given in row-major order in the file
# i.e. scanning left-to-right in rows from top-to-bottom,
# the use of matrix() has effectively transposed rows & columns,
# so to convert it to our format just reverse the column order.
w <- owin(mask=rim)
plot(w)
# display it to check!

## End(Not run)
```

owin.object*Class owin***Description**

A class `owin` to define the “observation window” of a point pattern

Details

In the **spatstat** library, a point pattern dataset must include information about the window or region in which the pattern was observed. A window is described by an object of class “`owin`”. Windows of arbitrary shape are supported.

An object of class “`owin`” has one of three types:

- “rectangle”: a rectangle in the two-dimensional plane with edges parallel to the axes
- “polygonal”: a region whose boundary is a polygon or several polygons. The region may have holes and may consist of several polygonal components.
- “mask”: a binary image (a logical matrix) set to TRUE for pixels inside the window and FALSE outside the window.

Objects of class “`owin`” may be created by the function `owin` and converted from other types of data by the function `as.owin`.

They may be manipulated by the functions `as.rectangle`, `as.mask`, `complement.owin`, `rotate`, `shift`, `affine`, `erosion`, `dilation`, `opening` and `closing`.

Geometrical calculations available for windows include `area.owin`, `perimeter`, `diameter.owin`, `bounding.box`, `eroded.areas`, `bdist.points`, `bdist.pixels`, and `even.breaks.owin`. The mapping between continuous coordinates and pixel raster indices is facilitated by the functions `raster.x`, `raster.y` and `nearest.raster.point`.

There is a plot method for window objects, `plot.owin`. This may be useful if you wish to plot a point pattern’s window without the points for graphical purposes.

There are also methods for `summary` and `print`.

Warnings

In a window of type "mask", the row index corresponds to increasing y coordinate, and the column index corresponds to increasing x coordinate.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

`owin`, `as.owin`, `as.rectangle`, `as.mask`, `summary.owin`, `print.owin`, `complement.owin`, `erosion`,
`dilation`, `opening`, `closing`, `affine.owin`, `shift.owin`, `rotate.owin`, `raster.x`, `raster.y`,
`nearest.raster.point`, `plot.owin`, `area.owin`, `bounding.box`, `diameter`, `eroded.areas`, `bdist.points`,
`bdist.pixels`

Examples

```
w <- owin()
w <- owin(c(0,1), c(0,1))
# the unit square

w <- owin(c(0,1), c(0,2))
## Not run:
plot(w)
# plots edges of a box 1 unit x 2 units
v <- locator()
# click on points in the plot window
# to be the vertices of a polygon
# traversed in anticlockwise order
u <- owin(c(0,1), c(0,2), poly=v)
plot(u)
# plots polygonal boundary using polygon()
plot(as.mask(u, eps=0.02))
# plots discrete pixel approximation to polygon

## End(Not run)
```

Description

Computes the matrix of distances between all pairs of 'things' in a dataset

Usage

```
pairdist(X, ...)
```

Arguments

- | | |
|-----|---|
| X | Object specifying the locations of a set of 'things' (such as a set of points or a set of line segments). |
| ... | Further arguments depending on the method. |

Details

Given a dataset X and Y (representing either a point pattern or a line segment pattern) pairdist computes the distance between each pair of ‘things’ in the dataset, and returns a matrix containing these distances.

The function pairdist is generic, with methods for point patterns (objects of class "ppp"), line segment patterns (objects of class "psp") and a default method. See the documentation for [pairdist.ppp](#), [pairdist.psp](#) or [pairdist.default](#) for details.

Value

A square matrix whose [i , j] entry is the distance between the ‘things’ numbered i and j.

Author(s)

Pavel Grabarnik <pavel.grabar@issp.serpukhov.su> and Adrian Baddeley <Adrian.Baddeley@csiro.au>
<http://www.maths.uwa.edu.au/~adrian/>

See Also

[pairdist.ppp](#), [pairdist.psp](#), [pairdist.default](#), [crossdist](#), [nndist](#), [Kest](#)

pairdist.default *Pairwise distances*

Description

Computes the matrix of distances between all pairs of points in a set of points

Usage

```
## Default S3 method:  
pairdist(X, Y=NULL, ..., period=NULL, method="C", squared=FALSE)
```

Arguments

X, Y	Arguments specifying the coordinates of a set of points. Typically X and Y would be numeric vectors of equal length. Alternatively Y may be omitted and X may be a list with two components x and y, or a matrix with two columns.
...	Ignored.
period	Optional. Dimensions for periodic edge correction.
method	String specifying which method of calculation to use. Values are "C" and "interpreted". Usually not specified.
squared	Logical. If squared=TRUE, the squared distances are returned instead (this computation is faster).

Details

Given the coordinates of a set of points, this function computes the Euclidean distances between all pairs of points, and returns the matrix of distances. It is a method for the generic function *pairdist*.

The arguments *X* and *Y* must determine the coordinates of a set of points. Typically *X* and *Y* would be numeric vectors of equal length. Alternatively *Y* may be omitted and *X* may be a list with two components named *x* and *y*, or a matrix or data frame with two columns.

Alternatively if *period* is given, then the distances will be computed in the ‘periodic’ sense (also known as ‘torus’ distance). The points will be treated as if they are in a rectangle of width *period*[1] and height *period*[2]. Opposite edges of the rectangle are regarded as equivalent.

If *squared*=TRUE then the *squared* Euclidean distances d^2 are returned, instead of the Euclidean distances d . The squared distances are faster to calculate, and are sufficient for many purposes (such as finding the nearest neighbour of a point).

The argument *method* is not normally used. It is retained only for checking the validity of the software. If *method* = “interpreted” then the distances are computed using interpreted R code only. If *method*=“C” (the default) then C code is used. The C code is somewhat faster.

Value

A square matrix whose [i,j] entry is the distance between the points numbered i and j.

Author(s)

Pavel Grabarnik <pavel.grabar@issp.serpukhov.su> and Adrian Baddeley <Adrian.Baddeley@csiro.au>
<http://www.maths.uwa.edu.au/~adrian/>

See Also

[crossdist](#), [nndist](#), [Kest](#)

Examples

```
x <- runif(100)
y <- runif(100)
d <- pairdist(x, y)
d <- pairdist(cbind(x,y))
d <- pairdist(x, y, period=c(1,1))
d <- pairdist(x, y, squared=TRUE)
```

pairdist.lpp

Pairwise shortest-path distances between points on a linear network

Description

Given a pattern of points on a linear network, compute the matrix of distances between all pairs of points, measuring distance by the shortest path in the network.

Usage

```
## S3 method for class 'lpp'
pairdist(X, ..., method="C")
```

Arguments

X	Point pattern on linear network (object of class "lpp").
method	Optional string determining the method of calculation. Either "interpreted" or "C".
...	Ignored.

Details

Given a pattern of points on a linear network, this function computes the matrix of distances between all pairs of points, measuring distance by the shortest path in the network.

If `method="C"` the distances are computed using code in the C language. If `method="interpreted"` then the computation is performed using interpreted R code. The R code is much slower, but is provided for checking purposes.

Value

A symmetric matrix.

Author(s)

Ang Qi Wei <aqw07398@hotmail.com> and Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

See Also

[lpp](#)

Examples

```
example(lpp)
pairdist(X)
```

`pairdist.pp3`

Pairwise distances in Three Dimensions

Description

Computes the matrix of distances between all pairs of points in a three-dimensional point pattern.

Usage

```
## S3 method for class 'pp3'
pairdist(X, ..., periodic=FALSE, squared=FALSE)
```

Arguments

X	A point pattern (object of class "pp3").
...	Ignored.
periodic	Logical. Specifies whether to apply a periodic edge correction.
squared	Logical. If <code>squared=TRUE</code> , the squared distances are returned instead (this computation is faster).

Details

This is a method for the generic function *pairdist*.

Given a three-dimensional point pattern *X* (an object of class "pp3"), this function computes the Euclidean distances between all pairs of points in *X*, and returns the matrix of distances.

Alternatively if *periodic*=TRUE and the window containing *X* is a box, then the distances will be computed in the 'periodic' sense (also known as 'torus' distance): opposite faces of the box are regarded as equivalent. This is meaningless if the window is not a box.

If *squared*=TRUE then the *squared* Euclidean distances d^2 are returned, instead of the Euclidean distances *d*. The squared distances are faster to calculate, and are sufficient for many purposes (such as finding the nearest neighbour of a point).

Value

A square matrix whose [i,j] entry is the distance between the points numbered i and j.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> based on two-dimensional code by Pavel Grabarnik.

See Also

pairdist, crossdist, nndist, K3est

Examples

```
X <- runifpoint3(20)
d <- pairdist(X)
d <- pairdist(X, periodic=TRUE)
d <- pairdist(X, squared=TRUE)
```

pairdist.ppp

Pairwise distances

Description

Computes the matrix of distances between all pairs of points in a point pattern.

Usage

```
## S3 method for class 'ppp'
pairdist(X, ..., periodic=FALSE, method="C", squared=FALSE)
```

Arguments

<i>X</i>	A point pattern (object of class "ppp").
...	Ignored.
<i>periodic</i>	Logical. Specifies whether to apply a periodic edge correction.
<i>method</i>	String specifying which method of calculation to use. Values are "C" and "interpreted". Usually not specified.
<i>squared</i>	Logical. If <i>squared</i> =TRUE, the squared distances are returned instead (this computation is faster).

Details

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

Given a point pattern X (an object of class "ppx"), this function computes the Euclidean distances between all pairs of points in X , and returns the matrix of distances.

Alternatively if `periodic=TRUE` and the window containing X is a rectangle, then the distances will be computed in the 'periodic' sense (also known as 'torus' distance): opposite edges of the rectangle are regarded as equivalent. This is meaningless if the window is not a rectangle.

If `squared=TRUE` then the *squared* Euclidean distances d^2 are returned, instead of the Euclidean distances d . The squared distances are faster to calculate, and are sufficient for many purposes (such as finding the nearest neighbour of a point).

The argument `method` is not normally used. It is retained only for checking the validity of the software. If `method = "interpreted"` then the distances are computed using interpreted R code only. If `method="C"` (the default) then C code is used. The C code is somewhat faster.

Value

A square matrix whose $[i, j]$ entry is the distance between the points numbered i and j .

Author(s)

Pavel Grabarnik <pavel.grabar@issp.serpukhov.su> and Adrian Baddeley <Adrian.Baddeley@csiro.au>
<http://www.maths.uwa.edu.au/~adrian/>

See Also

`pairdist`, `pairdist.default`, `pairdist.psp`, `crossdist`, `nndist`, `Kest`

Examples

```
data(cells)
d <- pairdist(cells)
d <- pairdist(cells, periodic=TRUE)
d <- pairdist(cells, squared=TRUE)
```

`pairdist.ppx`

Pairwise Distances in Any Dimensions

Description

Computes the matrix of distances between all pairs of points in a multi-dimensional point pattern.

Usage

```
## S3 method for class 'ppx'
pairdist(X, ...)
```

Arguments

- `X` A point pattern (object of class "ppx").
- `...` Arguments passed to `coords.ppx` to determine which coordinates should be used.

Details

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

Given a multi-dimensional point pattern `X` (an object of class "ppx"), this function computes the Euclidean distances between all pairs of points in `X`, and returns the matrix of distances.

By default, both spatial and temporal coordinates are extracted. To obtain the spatial distance between points in a space-time point pattern, set `temporal=FALSE`.

Value

A square matrix whose [i,j] entry is the distance between the points numbered i and j.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

See Also

`pairdist`, `crossdist`, `nndist`

Examples

```
df <- data.frame(x=runif(4),y=runif(4),z=runif(4),w=runif(4))
X <- ppx(data=df)
pairdist(X)
```

`pairdist.psp`

Pairwise distances between line segments

Description

Computes the matrix of distances between all pairs of line segments in a line segment pattern.

Usage

```
## S3 method for class 'psp'
pairdist(X, ..., method="Fortran", type="Hausdorff")
```

Arguments

<code>X</code>	A line segment pattern (object of class "psp").
<code>...</code>	Ignored.
<code>method</code>	String specifying which method of calculation to use. Values are "Fortran" and "interpreted". Usually not specified.
<code>type</code>	Type of distance to be computed. Options are "Hausdorff" and "separation". Partial matching is used.

Details

This function computes the distance between each pair of line segments in X , and returns the matrix of distances.

This is a method for the generic function `pairdist` for the class "psp".

The distances between line segments are measured in one of two ways:

- if `type="Hausdorff"`, distances are computed in the Hausdorff metric. The Hausdorff distance between two line segments is the *maximum* distance from any point on one of the segments to the nearest point on the other segment.
- if `type="separation"`, distances are computed as the *minimum* distance from a point on one line segment to a point on the other line segment. For example, line segments which cross over each other have separation zero.

The argument `method` is not normally used. It is retained only for checking the validity of the software. If `method = "interpreted"` then the distances are computed using interpreted R code only. If `method="Fortran"` (the default) then Fortran code is used. The Fortran code is somewhat faster.

Value

A square matrix whose $[i, j]$ entry is the distance between the line segments numbered i and j .

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`crossdist`, `nndist`, `pairdist.ppp`

Examples

```
L <- psp(runif(10), runif(10), runif(10), runif(10), owin())
D <- pairdist(L)
S <- pairdist(L, type="sep")
```

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

<code>r</code>	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

$$f(x_1, \dots, x_n) = \alpha \prod_i b(x_i) \prod_{i < j} h(x_i, x_j)$$

where x_1, \dots, 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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

Takacs, R. (1986) Estimator for the pair potential of a Gibbsian point process. *Statistics* **17**, 429–433.

See Also

`ppm`, `pairwise.family`, `ppm.object`, `Strauss` `rmh.ppm`

Examples

```

PairPiece(c(0.1,0.2))
# prints a sensible description of itself
data(cells)

## Not run:
ppm(cells, ~1, PairPiece(r = c(0.05, 0.1, 0.2)))
# fit a stationary piecewise constant pairwise interaction process

## End(Not run)

ppm(cells, ~polynom(x,y,3), PairPiece(c(0.05, 0.1)))
# nonstationary process with log-cubic polynomial trend

```

pairs.im

Scatterplot Matrix for Pixel Images

Description

Produces a scatterplot matrix of the pixel values in two or more pixel images.

Usage

```
## S3 method for class 'im'
pairs(..., plot=TRUE)
```

Arguments

- | | |
|------|---|
| ... | Any number of arguments, each of which is either a pixel image (object of class "im") or a named argument to be passed to pairs.default . |
| plot | Logical. If TRUE, the scatterplot matrix is plotted. |

Details

This is a method for the generic function [pairs](#) for the class of pixel images.

It produces a square array of plot panels, in which each panel shows a scatterplot of the pixel values of one image against the corresponding pixel values of another image.

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

First the pixel image domains are intersected, and converted to a common pixel resolution. Then the corresponding pixel values of each image are extracted. Then [pairs.default](#) is called to plot the scatterplot matrix.

Any arguments in ... which are not pixel images will be passed to [pairs.default](#) to control the plot.

Value

Invisible. A [data.frame](#) containing the corresponding pixel values for each image. The return value also belongs to the class [plotpairsim](#) which has a plot method, so that it can be re-plotted.

Note

To control the appearance of the individual scatterplot panels, see [pairs.default](#), [points](#) or [par](#). To control the plotting symbol for the points in the scatterplot, use the arguments `pch`, `col`, `bg` as described under [points](#) (because the default panel plotter is the function [points](#)). To suppress the tick marks on the plot axes, type `par(xaxt="n", yaxt="n")` before calling [pairs](#).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[pairs](#), [pairs.default](#), [plot.im](#), [im](#), [par](#)

Examples

```
X <- density(rpoispp(30))
Y <- density(rpoispp(40))
Z <- density(rpoispp(30))
pairs(X,Y,Z)
```

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.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

Geyer, C.J. (1999) Likelihood Inference for Spatial Point Processes. Chapter 3 in O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. Van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*, Chapman and Hall / CRC, Monographs on Statistics and Applied Probability, number 80. Pages 79–140.

See Also

[Geyer](#) to create the Geyer saturation process.

[SatPiece](#) to create a saturated process with piecewise constant pair potential.

[Saturated](#) to create a more general saturation model.

Other families: [inforder.family](#), [ord.family](#), [pairwise.family](#).

Pairwise

Generic Pairwise Interaction model

Description

Creates an instance of a pairwise interaction point process model which can then be fitted to point pattern data.

Usage

```
Pairwise(pot, name, par, parnames, printfun)
```

Arguments

pot	An R language function giving the user-supplied pairwise interaction potential.
name	Character string.
par	List of numerical values for irregular parameters
parnames	Vector of names of irregular parameters
printfun	Do not specify this argument: for internal use only.

Details

This code constructs a member of the pairwise interaction family [pairwise.family](#) with arbitrary pairwise interaction potential given by the user.

Each pair of points in the point pattern contributes a factor $h(d)$ to the probability density, where d is the distance between the two points. The factor term $h(d)$ is

$$h(d) = \exp(-\theta \text{pot}(d))$$

provided $\text{pot}(d)$ 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 $\text{pot}(d) = -\infty$. 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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

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

Examples

```
#This is the same as StraussHard(r=0.7,h=0.05)
strpot <- function(d,par) {
  r <- par$r
  h <- par$h
  value <- (d <= r)
  value[d < h] <- -Inf
  value
}
mySH <- Pairwise(strpot, "StraussHard process", list(r=0.7,h=0.05),
  c("interaction distance r", "hard core distance h"))
data(cells)
ppm(cells, ~ 1, mySH, correction="isotropic")

# Fiksel (1984) double exponential interaction
# see Stoyan, Kendall, Mecke 1987 p 161

fikspot <- function(d, par) {
  r <- par$r
  h <- par$h
  zeta <- par$zeta
  value <- exp(-zeta * d)
  value[d < h] <- -Inf
  value[d > r] <- 0
  value
}
Fiksel <- Pairwise(fikspot, "Fiksel double exponential process",
  list(r=3.5, h=1, zeta=1),
  c("interaction distance r",
    "hard core distance h",
    "exponential coefficient zeta"))

data(spruces)
fit <- ppm(unmark(spruces), ~1, Fiksel, rbord=3.5)
fit
plot(fitin(fit), xlim=c(0,4))
coef(fit)
# corresponding values obtained by Fiksel (1984) were -1.9 and -6.0
```

pairwise.family

*Pairwise Interaction Process Family***Description**

An object describing the family of all pairwise interaction Gibbs point processes.

Details**Advanced Use Only!**

This structure would not normally be touched by the user. It describes the pairwise interaction family of point process models.

If you need to create a specific pairwise interaction model for use in modelling, use the function [Pairwise](#) or one of the existing functions listed below.

Anyway, `pairwise.family` is an object of class "isf" containing a function `pairwise.family$eval` for evaluating the sufficient statistics of any pairwise interaction point process model taking an exponential family form.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

Other families: [pairsat.family](#), [ord.family](#), [inforder.family](#).

Pairwise interactions: [Poisson](#), [Pairwise](#), [PairPiece](#), [Fiksel](#), [Hardcore](#), [LennardJones](#), [MultiHard](#), [MultiStrauss](#), [MultiStraussHard](#), [Strauss](#), [StraussHard](#), [Softcore](#).

Other interactions: [AreaInter](#), [Geyer](#), [Saturated](#), [Ord](#), [OrdThresh](#).

pcf

*Pair Correlation Function***Description**

Estimate the pair correlation function.

Usage

```
pcf(X, ...)
```

Arguments

- | | |
|------------------|---|
| <code>X</code> | Either the observed data point pattern, or an estimate of its K function, or an array of multitype K functions (see Details). |
| <code>...</code> | Other arguments passed to the appropriate method. |

Details

The pair correlation function of a stationary point process is

$$g(r) = \frac{K'(r)}{2\pi r}$$

where $K'(r)$ is the derivative of $K(r)$, the reduced second moment function (aka “Ripley’s K function”) of the point process. See [Kest](#) for information about $K(r)$. For a stationary Poisson process, the pair correlation function is identically equal to 1. Values $g(r) < 1$ suggest inhibition between points; values greater than 1 suggest clustering.

We also apply the same definition to other variants of the classical K function, such as the multitype K functions (see [Kcross](#), [Kdot](#)) and the inhomogeneous K function (see [Kinhom](#)). For all these variants, the benchmark value of $K(r) = \pi r^2$ corresponds to $g(r) = 1$.

This routine computes an estimate of $g(r)$ either directly from a point pattern, or indirectly from an estimate of $K(r)$ or one of its variants.

This function is generic, with methods for the classes “`ppp`”, “`fv`” and “`fasp`”.

If X is a point pattern (object of class “`ppp`”) then the pair correlation function is estimated using a traditional kernel smoothing method (Stoyan and Stoyan, 1994). See [pcf.ppp](#) for details.

If X is a function value table (object of class “`fv`”), then it is assumed to contain estimates of the K function or one of its variants (typically obtained from [Kest](#) or [Kinhom](#)). This routine computes an estimate of $g(r)$ using smoothing splines to approximate the derivative. See [pcf.fv](#) for details.

If X is a function value array (object of class “`fasp`”), then it is assumed to contain estimates of several K functions (typically obtained from [Kmulti](#) or [alltypes](#)). This routine computes an estimate of $g(r)$ for each cell in the array, using smoothing splines to approximate the derivatives. See [pcf.fasp](#) for details.

Value

Either a function value table (object of class “`fv`”, see [fv.object](#)) representing a pair correlation function, or a function array (object of class “`fasp`”, see [fasp.object](#)) representing an array of pair correlation functions.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

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

See Also

[pcf.ppp](#), [pcf.fv](#), [pcf.fasp](#), [Kest](#), [Kinhom](#), [Kcross](#), [Kdot](#), [Kmulti](#), [alltypes](#)

Examples

```
# ppp object
data(simdat)

p <- pcf(simdat)
```

```

plot(p)

# fv object
K <- Kest(simdat)
p2 <- pcf(K, spar=0.8, method="b")
plot(p2)

# multitype pattern; fasp object
data(betacells)

betaK <- alltypes(betacells, "K")
betap <- pcf(betaK, spar=1, method="b")
plot(betap)

```

pcf.fasp*Pair Correlation Function obtained from array of K functions***Description**

Estimates the (bivariate) pair correlation functions of a point pattern, given an array of (bivariate) K functions.

Usage

```
## S3 method for class 'fasp'
pcf(X, ..., method="c")
```

Arguments

X	An array of multitype K functions (object of class "fasp").
...	Arguments controlling the smoothing spline function <code>smooth.spline</code> .
method	Letter "a", "b", "c" or "d" indicating the method for deriving the pair correlation function from the K function.

Details

The pair correlation function of a stationary point process is

$$g(r) = \frac{K'(r)}{2\pi r}$$

where $K'(r)$ is the derivative of $K(r)$, the reduced second moment function (aka "Ripley's K function") of the point process. See [Kest](#) for information about $K(r)$. For a stationary Poisson process, the pair correlation function is identically equal to 1. Values $g(r) < 1$ suggest inhibition between points; values greater than 1 suggest clustering.

We also apply the same definition to other variants of the classical K function, such as the multitype K functions (see [Kcross](#), [Kdot](#)) and the inhomogeneous K function (see [Kinhom](#)). For all these variants, the benchmark value of $K(r) = \pi r^2$ corresponds to $g(r) = 1$.

This routine computes an estimate of $g(r)$ from an array of estimates of $K(r)$ or its variants, using smoothing splines to approximate the derivatives. It is a method for the generic function [pcf](#).

The argument X should be a function array (object of class "fasp", see `fasp.object`) containing several estimates of K functions. This should have been obtained from `alltypes` with the argument `fun="K"`.

The smoothing spline operations are performed by `smooth.spline` and `predict.smooth.spline` from the `modreg` library. Four numerical methods are available:

- "**a**" apply smoothing to $K(r)$, estimate its derivative, and plug in to the formula above;
- "**b**" apply smoothing to $Y(r) = \frac{K(r)}{2\pi r}$ constraining $Y(0) = 0$, estimate the derivative of Y , and solve;
- "**c**" apply smoothing to $Z(r) = \frac{K(r)}{\pi r^2}$ constraining $Z(0) = 1$, estimate its derivative, and solve.
- "**d**" apply smoothing to $V(r) = \sqrt{K(r)}$, estimate its derivative, and solve.

Method "c" seems to be the best at suppressing variability for small values of r . However it effectively constrains $g(0) = 1$. If the point pattern seems to have inhibition at small distances, you may wish to experiment with method "b" which effectively constrains $g(0) = 0$. Method "a" seems comparatively unreliable.

Useful arguments to control the splines include the smoothing tradeoff parameter `spar` and the degrees of freedom `df`. See `smooth.spline` for details.

Value

A function array (object of class "fasp", see `fasp.object`) representing an array of pair correlation functions. This can be thought of as a matrix Y each of whose entries $Y[i, j]$ is a function value table (class "fv") representing the pair correlation function between points of type i and points of type j .

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Stoyan, D, Kendall, W.S. and Mecke, J. (1995) *Stochastic geometry and its applications*. 2nd edition. Springer Verlag.
 Stoyan, D. and Stoyan, H. (1994) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

See Also

`Kest`, `Kinhom`, `Kcross`, `Kdot`, `Kmulti`, `alltypes`, `smooth.spline`, `predict.smooth.spline`

Examples

```
# multitype point pattern
data(betacells)

KK <- alltypes(betacells, "K")
p <- pcf.fasp(KK, spar=0.5, method="b")
plot(p)
# short range inhibition between all types
# strong inhibition between "on" and "off"
```

pcf.fvPair Correlation Function obtained from K Function

Description

Estimates the pair correlation function of a point pattern, given an estimate of the K function.

Usage

```
## S3 method for class 'fv'
pcf(X, ..., method="c")
```

Arguments

X	An estimate of the K function or one of its variants. An object of class "fv".
...	Arguments controlling the smoothing spline function <code>smooth.spline</code> .
method	Letter "a", "b", "c" or "d" indicating the method for deriving the pair correlation function from the K function.

Details

The pair correlation function of a stationary point process is

$$g(r) = \frac{K'(r)}{2\pi r}$$

where $K'(r)$ is the derivative of $K(r)$, the reduced second moment function (aka "Ripley's K function") of the point process. See [Kest](#) for information about $K(r)$. For a stationary Poisson process, the pair correlation function is identically equal to 1. Values $g(r) < 1$ suggest inhibition between points; values greater than 1 suggest clustering.

We also apply the same definition to other variants of the classical K function, such as the multitype K functions (see [Kcross](#), [Kdot](#)) and the inhomogeneous K function (see [Kinhom](#)). For all these variants, the benchmark value of $K(r) = \pi r^2$ corresponds to $g(r) = 1$.

This routine computes an estimate of $g(r)$ from an estimate of $K(r)$ or its variants, using smoothing splines to approximate the derivative. It is a method for the generic function [pcf](#) for the class "fv".

The argument X should be an estimated K function, given as a function value table (object of class "fv", see [fv.object](#)). This object should be the value returned by [Kest](#), [Kcross](#), [Kmulti](#) or [Kinhom](#).

The smoothing spline operations are performed by [smooth.spline](#) and [predict.smooth.spline](#) from the [modreg](#) library. Four numerical methods are available:

- "a" apply smoothing to $K(r)$, estimate its derivative, and plug in to the formula above;
- "b" apply smoothing to $Y(r) = \frac{K(r)}{2\pi r}$ constraining $Y(0) = 0$, estimate the derivative of Y , and solve;
- "c" apply smoothing to $Z(r) = \frac{K(r)}{\pi r^2}$ constraining $Z(0) = 1$, estimate its derivative, and solve.
- "d" apply smoothing to $V(r) = \sqrt{K(r)}$, estimate its derivative, and solve.

Method "c" seems to be the best at suppressing variability for small values of r . However it effectively constrains $g(0) = 1$. If the point pattern seems to have inhibition at small distances, you may wish to experiment with method "b" which effectively constrains $g(0) = 0$. Method "a" seems comparatively unreliable.

Useful arguments to control the splines include the smoothing tradeoff parameter `spar` and the degrees of freedom `df`. See [smooth.spline](#) for details.

Value

A function value table (object of class "`fv`", see [fv.object](#)) representing a pair correlation function.

Essentially a data frame containing (at least) the variables

<code>r</code>	the vector of values of the argument r at which the pair correlation function $g(r)$ has been estimated
<code>pcf</code>	vector of values of $g(r)$

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Stoyan, D, Kendall, W.S. and Mecke, J. (1995) *Stochastic geometry and its applications*. 2nd edition. Springer Verlag.
 Stoyan, D. and Stoyan, H. (1994) Fractals, random shapes and point fields: methods of geometrical statistics. John Wiley and Sons.

See Also

[pcf](#), [pcf.ppp](#), [Kest](#), [Kinhom](#), [Kcross](#), [Kdot](#), [Kmulti](#), [alltypes](#), [smooth.spline](#), [predict.smooth.spline](#)

Examples

```
# univariate point pattern
data(simdat)

K <- Kest(simdat)
p <- pcf.fv(K, spar=0.5, method="b")
plot(p, main="pair correlation function for simdat")
# indicates inhibition at distances r < 0.3
```

Description

Estimates the pair correlation function of a point pattern using kernel methods.

Usage

```
## S3 method for class 'ppp'
pcf(X, ..., r = NULL, kernel="epanechnikov", bw=NULL, stoyan=0.15,
     correction=c("translate", "Ripley"))
```

Arguments

X	A point pattern (object of class "ppp").
r	Vector of values for the argument r at which $g(r)$ should be evaluated. There is a sensible default.
kernel	Choice of smoothing kernel, passed to <code>density</code> .
bw	Bandwidth for smoothing kernel, passed to <code>density</code> .
...	Other arguments passed to the kernel density estimation function <code>density</code> .
stoyan	Bandwidth coefficient; see Details.
correction	Choice of edge correction.

Details

The pair correlation function $g(r)$ is a summary of the dependence between points in a spatial point process. The best intuitive interpretation is the following: the probability $p(r)$ of finding two points at locations x and y separated by a distance r is equal to

$$p(r) = \lambda^2 g(r) dx dy$$

where λ is the intensity of the point process. For a completely random (uniform Poisson) process, $p(r) = \lambda^2$ so $g(r) = 1$.

Formally, the pair correlation function of a stationary point process is defined by

$$g(r) = \frac{K'(r)}{2\pi r}$$

where $K'(r)$ is the derivative of $K(r)$, the reduced second moment function (aka “Ripley’s K function”) of the point process. See [Kest](#) for information about $K(r)$.

For a stationary Poisson process, the pair correlation function is identically equal to 1. Values $g(r) < 1$ suggest inhibition between points; values greater than 1 suggest clustering.

This routine computes an estimate of $g(r)$ by the kernel smoothing method (Stoyan and Stoyan (1994), pages 284–285). By default, their recommendations are followed exactly.

If `correction="translate"` then the translation correction is used. The estimate is given in equation (15.15), page 284 of Stoyan and Stoyan (1994).

If `correction="Ripley"` then Ripley’s isotropic edge correction is used; the estimate is given in equation (15.18), page 285 of Stoyan and Stoyan (1994).

If `correction=c("translate", "Ripley")` then both estimates will be computed.

The choice of smoothing kernel is controlled by the argument `kernel` which is passed to [density](#). The default is the Epanechnikov kernel, recommended by Stoyan and Stoyan (1994, page 285).

The bandwidth of the smoothing kernel can be controlled by the argument `bw`. Its precise interpretation is explained in the documentation for [density](#). For the Epanechnikov kernel, the argument `bw` is equivalent to $h/\sqrt{5}$.

Stoyan and Stoyan (1994, page 285) recommend using the Epanechnikov kernel with support $[-h, h]$ chosen by the rule of thumb $h = c/\sqrt{\lambda}$, where λ is the (estimated) intensity of the point

process, and c is a constant in the range from 0.1 to 0.2. See equation (15.16). If `bw` is missing, then this rule of thumb will be applied. The argument `sstoyan` determines the value of c .

The argument `r` is the vector of values for the distance r at which $g(r)$ should be evaluated. There is a sensible default. If it is specified, `r` must be a vector of increasing numbers starting from `r[1] = 0`, and `max(r)` must not exceed half the diameter of the window.

To compute a confidence band for the true value of the pair correlation function, use [lohboot](#).

Value

A function value table (object of class "fv"). Essentially a data frame containing the variables

<code>r</code>	the vector of values of the argument r at which the pair correlation function $g(r)$ has been estimated
<code>theo</code>	vector of values equal to 1, the theoretical value of $g(r)$ for the Poisson process
<code>trans</code>	vector of values of $g(r)$ estimated by translation correction
<code>iso</code>	vector of values of $g(r)$ estimated by Ripley isotropic correction

as required.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

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

See Also

[Kest](#), [pcf](#), [density](#), [lohboot](#).

Examples

```
data(simdat)

p <- pcf(simdat)
plot(p, main="pair correlation function for simdat")
# indicates inhibition at distances r < 0.3
```

Description

Estimates the pair correlation function from a three-dimensional point pattern.

Usage

```
pcf3est(X, ..., rmax = NULL, nrval = 128, correction = c("translation",
"isotropic"), delta=NULL, adjust=1, biascorrect=TRUE)
```

Arguments

X	Three-dimensional point pattern (object of class "pp3").
...	Ignored.
rmax	Optional. Maximum value of argument r for which $g_3(r)$ will be estimated.
n rval	Optional. Number of values of r for which $g_3(r)$ will be estimated.
correction	Optional. Character vector specifying the edge correction(s) to be applied. See Details.
delta	Optional. Half-width of the Epanechnikov smoothing kernel.
adjust	Optional. Adjustment factor for the default value of delta.
biascorrect	Logical value. Whether to correct for underestimation due to truncation of the kernel near $r = 0$.

Details

For a stationary point process Φ in three-dimensional space, the pair correlation function is

$$g_3(r) = \frac{K'_3(r)}{4\pi r^2}$$

where K'_3 is the derivative of the three-dimensional K -function (see [K3est](#)).

The three-dimensional point pattern X is assumed to be a partial realisation of a stationary point process Φ . The distance between each pair of distinct points is computed. Kernel smoothing is applied to these distance values (weighted by an edge correction factor) and the result is renormalised to give the estimate of $g_3(r)$.

The available edge corrections are:

"translation": the Ohser translation correction estimator (Ohser, 1983; Baddeley et al, 1993)

"isotropic": the three-dimensional counterpart of Ripley's isotropic edge correction (Ripley, 1977; Baddeley et al, 1993).

Kernel smoothing is performed using the Epanechnikov kernel with half-width delta. If delta is missing, the default is to use the rule-of-thumb $\delta = 0.26/\lambda^{1/3}$ where $\lambda = n/v$ is the estimated intensity, computed from the number n of data points and the volume v of the enclosing box. This default value of delta is multiplied by the factor adjust.

The smoothing estimate of the pair correlation $g_3(r)$ is typically an underestimate when r is small, due to truncation of the kernel at $r = 0$. If biascorrect=TRUE, the smoothed estimate is approximately adjusted for this bias. This is advisable whenever the dataset contains a sufficiently large number of points.

Value

A function value table (object of class "fv") that can be plotted, printed or coerced to a data frame containing the function values.

Additionally the value of delta is returned as an attribute of this object.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rana Moyeed.

References

- Baddeley, A.J., Moyeed, R.A., Howard, C.V. and Boyde, A. (1993) Analysis of a three-dimensional point pattern with replication. *Applied Statistics* **42**, 641–668.
- Ohser, J. (1983) On estimators for the reduced second moment measure of point processes. *Mathematische Operationsforschung und Statistik, series Statistics*, **14**, 63 – 71.
- Ripley, B.D. (1977) Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, Series B*, **39**, 172 – 212.

See Also

[K3est](#), [pcf](#)

Examples

```
X <- rpoispp3(250)
Z <- pcf3est(X)
Zbias <- pcf3est(X, biascorrect=FALSE)
if(interactive()) {
  opa <- par(mfrow=c(1,2))
  plot(Z, ylim.covers=c(0, 1.2))
  plot(Zbias, ylim.covers=c(0, 1.2))
  par(opa)
}
attr(Z, "delta")
```

pcfcross

Multitype pair correlation function (cross-type)

Description

Calculates an estimate of the cross-type pair correlation function for a multitype point pattern.

Usage

```
pcfcross(X, i, j, ...)
```

Arguments

- | | |
|----------|--|
| X | The observed point pattern, from which an estimate of the cross-type pair correlation function $g_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor). |
| i | The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of <code>marks(X)</code> . |
| j | The type (mark value) of the points in X to which distances are measured. A character string (or something that will be converted to a character string). Defaults to the second level of <code>marks(X)</code> . |
| ... | Arguments passed to pcf.ppp . |

Details

The cross-type pair correlation function is a generalisation of the pair correlation function [pcf](#) to multitype point patterns.

For two locations x and y separated by a distance r , the probability $p(r)$ of finding a point of type i at location x and a point of type j at location y is

$$p(r) = \lambda_i \lambda_j g_{i,j}(r) dx dy$$

where λ_i is the intensity of the points of type i . For a completely random Poisson marked point process, $p(r) = \lambda_i \lambda_j$ so $g_{i,j}(r) = 1$. Indeed for any marked point pattern in which the points of type i are independent of the points of type j , the theoretical value of the cross-type pair correlation is $g_{i,j}(r) = 1$.

For a stationary multitype point process, the cross-type pair correlation function between marks i and j is formally defined as

$$g_{i,j}(r) = \frac{K'_{i,j}(r)}{2\pi r}$$

where $K'_{i,j}$ is the derivative of the cross-type K function $K_{i,j}(r)$ of the point process. See [Kest](#) for information about $K(r)$.

The command [pcfcross](#) computes a kernel estimate of the cross-type pair correlation function between marks i and j . It uses [pcf.ppp](#) to compute kernel estimates of the pair correlation functions for several unmarked point patterns, and uses the bilinear properties of second moments to obtain the cross-type pair correlation.

See [pcf.ppp](#) for a list of arguments that control the kernel estimation.

The companion function [pcfdot](#) computes the corresponding analogue of [Kdot](#).

Value

An object of class "fv", see [fv.object](#), which can be plotted directly using [plot.fv](#).

Essentially a data frame containing columns

- | | |
|------|---|
| r | the vector of values of the argument r at which the function $g_{i,j}$ has been estimated |
| theo | the theoretical value $g_{i,j}(r) = 1$ for independent marks. |

together with columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $g_{i,j}$ obtained by the edge corrections named.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

Mark connection function [markconnect](#).

Multitype pair correlation [pcfdot](#).

Pair correlation [pcf](#), [pcf.ppp](#).

[Kcross](#)

Examples

```
data(amacrine)
p <- pcfcross(amacrine, "off", "on")
p <- pcfcross(amacrine, "off", "on", stoyan=0.1)
plot(p)
```

pcfcross.inhom

Inhomogeneous Multitype Pair Correlation Function (Cross-Type)

Description

Estimates the inhomogeneous cross-type pair correlation function for a multitype point pattern.

Usage

```
pcfcross.inhom(X, i, j, lambdaI = NULL, lambdaJ = NULL, ...,
                r = NULL, breaks = NULL,
                kernel="epanechnikov", bw=NULL, stoyan=0.15,
                correction = c("isotropic", "Ripley", "translate"),
                sigma = NULL, varcov = NULL)
```

Arguments

X	The observed point pattern, from which an estimate of the inhomogeneous cross-type pair correlation function $g_{ij}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor).
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of <code>marks(X)</code> .
j	The type (mark value) of the points in X to which distances are measured. A character string (or something that will be converted to a character string). Defaults to the second level of <code>marks(X)</code> .
lambdaI	Optional. Values of the estimated intensity function of the points of type i. Either a vector giving the intensity values at the points of type i, a pixel image (object of class "im") giving the intensity values at all locations, or a function(x,y) which can be evaluated to give the intensity value at any location.
lambdaJ	Optional. Values of the estimated intensity function of the points of type j. A numeric vector, pixel image or function(x,y).
r	Vector of values for the argument r at which $g_{ij}(r)$ should be evaluated. There is a sensible default.
breaks	Optional. An alternative to the argument r . Not normally invoked by the user.
kernel	Choice of smoothing kernel, passed to <code>density.default</code> .
bw	Bandwidth for smoothing kernel, passed to <code>density.default</code> .
...	Other arguments passed to the kernel density estimation function <code>density.default</code> .
stoyan	Bandwidth coefficient; see Details.
correction	Choice of edge correction.
sigma, varcov	Optional arguments passed to <code>density.ppp</code> to control the smoothing bandwidth, when <code>lambdaI</code> or <code>lambdaJ</code> is estimated by kernel smoothing.

Details

The inhomogeneous cross-type pair correlation function $g_{ij}(r)$ is a summary of the dependence between two types of points in a multitype spatial point process that does not have a uniform density of points.

The best intuitive interpretation is the following: the probability $p(r)$ of finding two points, of types i and j respectively, at locations x and y separated by a distance r is equal to

$$p(r) = \lambda_i(x)\lambda_j(y)g(r) dx dy$$

where λ_i is the intensity function of the process of points of type i . For a multitype Poisson point process, this probability is $p(r) = \lambda_i(x)\lambda_j(y)$ so $g_{ij}(r) = 1$.

The command `pcfcross.inhom` estimates the inhomogeneous pair correlation using a modified version of the algorithm in `pcf.ppp`.

If the arguments `lambdaI` and `lambdaJ` are missing or null, they are estimated from `X` by kernel smoothing using a leave-one-out estimator.

Value

A function value table (object of class "fv"). Essentially a data frame containing the variables

<code>r</code>	the vector of values of the argument r at which the inhomogeneous cross-type pair correlation function $g_{ij}(r)$ has been estimated
<code>theo</code>	vector of values equal to 1, the theoretical value of $g_{ij}(r)$ for the Poisson process
<code>trans</code>	vector of values of $g_{ij}(r)$ estimated by translation correction
<code>iso</code>	vector of values of $g_{ij}(r)$ estimated by Ripley isotropic correction

as required.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`pcf.ppp`, `pcfinhom`, `pcfcross`, `pcfdot.inhom`

Examples

```
data(amacrine)
plot(pcfcross.inhom(amacrine, "on", "off", stoyan=0.1),
     legendpos="bottom")
```

pcfdot*Multitype pair correlation function (i-to-any)*

Description

Calculates an estimate of the multitype pair correlation function (from points of type i to points of any type) for a multitype point pattern.

Usage

```
pcfdot(X, i, ...)
```

Arguments

X	The observed point pattern, from which an estimate of the dot-type pair correlation function $g_{i\bullet}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor).
i	The type (mark value) of the points in X from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of marks(X).
...	Arguments passed to pcf.ppp .

Details

This is a generalisation of the pair correlation function [pcf](#) to multitype point patterns.

For two locations x and y separated by a nonzero distance r , the probability $p(r)$ of finding a point of type i at location x and a point of any type at location y is

$$p(r) = \lambda_i \lambda g_{i\bullet}(r) dx dy$$

where λ is the intensity of all points, and λ_i is the intensity of the points of type i . For a completely random Poisson marked point process, $p(r) = \lambda_i \lambda$ so $g_{i\bullet}(r) = 1$.

For a stationary multitype point process, the type- i -to-any-type pair correlation function between marks i and j is formally defined as

$$g_{i\bullet}(r) = \frac{K'_{i\bullet}(r)}{2\pi r}$$

where $K'_{i\bullet}$ is the derivative of the type- i -to-any-type K function $K_{i\bullet}(r)$ of the point process. See [Kdot](#) for information about $K_{i\bullet}(r)$.

The command [pcfdot](#) computes a kernel estimate of the multitype pair correlation function from points of type i to points of any type. It uses [pcf.ppp](#) to compute kernel estimates of the pair correlation functions for several unmarked point patterns, and uses the bilinear properties of second moments to obtain the multitype pair correlation.

See [pcf.ppp](#) for a list of arguments that control the kernel estimation.

The companion function [pcfcross](#) computes the corresponding analogue of [Kcross](#).

Value

An object of class "fv", see [fv.object](#), which can be plotted directly using [plot.fv](#).

Essentially a data frame containing columns

- | | |
|------|--|
| r | the vector of values of the argument r at which the function $g_{i\bullet}$ has been estimated |
| theo | the theoretical value $g_{i\bullet}(r) = 1$ for independent marks. |

together with columns named "border", "bord.modif", "iso" and/or "trans", according to the selected edge corrections. These columns contain estimates of the function $g_{i,j}$ obtained by the edge corrections named.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

Mark connection function [markconnect](#).

Multitype pair correlation [pcfcross](#).

Pair correlation [pcf,pcf.ppp](#).

[Kdot](#)

Examples

```
data(amacrine)
p <- pcfdot(amacrine, "on")
p <- pcfdot(amacrine, "on", stoyan=0.1)
plot(p)
```

pcfdot.inhom

Inhomogeneous Multitype Pair Correlation Function (Type-i-To-Any-Type)

Description

Estimates the inhomogeneous multitype pair correlation function (from type i to any type) for a multitype point pattern.

Usage

```
pcfdot.inhom(X, i, lambdaI = NULL, lambdadot = NULL, ...,
             r = NULL, breaks = NULL,
             kernel="epanechnikov", bw=NULL, stoyan=0.15,
             correction = c("isotropic", "Ripley", "translate"),
             sigma = NULL, varcov = NULL)
```

Arguments

<i>X</i>	The observed point pattern, from which an estimate of the inhomogeneous multitype pair correlation function $g_{i\bullet}(r)$ will be computed. It must be a multitype point pattern (a marked point pattern whose marks are a factor).
<i>i</i>	The type (mark value) of the points in <i>X</i> from which distances are measured. A character string (or something that will be converted to a character string). Defaults to the first level of <code>marks(X)</code> .
<code>lambdaI</code>	Optional. Values of the estimated intensity function of the points of type <i>i</i> . Either a vector giving the intensity values at the points of type <i>i</i> , a pixel image (object of class "im") giving the intensity values at all locations, or a function(<i>x,y</i>) which can be evaluated to give the intensity value at any location.
<code>lambdadot</code>	Optional. Values of the estimated intensity function of the point pattern <i>X</i> . A numeric vector, pixel image or function(<i>x,y</i>).
<i>r</i>	Vector of values for the argument <i>r</i> at which $g_{i\bullet}(r)$ should be evaluated. There is a sensible default.
<code>breaks</code>	Optional. An alternative to the argument <i>r</i> . Not normally invoked by the user.
<code>kernel</code>	Choice of smoothing kernel, passed to density.default .
<code>bw</code>	Bandwidth for smoothing kernel, passed to density.default .
<code>...</code>	Other arguments passed to the kernel density estimation function density.default .
<code>stoyan</code>	Bandwidth coefficient; see Details.
<code>correction</code>	Choice of edge correction.
<code>sigma, varcov</code>	Optional arguments passed to density.ppp to control the smoothing bandwidth, when <code>lambdaI</code> or <code>lambdadot</code> is estimated by kernel smoothing.

Details

The inhomogeneous multitype (type *i* to any type) pair correlation function $g_{i\bullet}(r)$ is a summary of the dependence between different types of points in a multitype spatial point process that does not have a uniform density of points.

The best intuitive interpretation is the following: the probability $p(r)$ of finding a point of type *i* at location *x* and another point of any type at location *y*, where *x* and *y* are separated by a distance *r*, is equal to

$$p(r) = \lambda_i(x)\lambda(y)g(r) dx dy$$

where λ_i is the intensity function of the process of points of type *i*, and where λ is the intensity function of the points of all types. For a multitype Poisson point process, this probability is $p(r) = \lambda_i(x)\lambda(y)$ so $g_{i\bullet}(r) = 1$.

The command `pcf dot.inhom` estimates the inhomogeneous multitype pair correlation using a modified version of the algorithm in [pcf.ppp](#).

If the arguments `lambdaI` and `lambdadot` are missing or null, they are estimated from *X* by kernel smoothing using a leave-one-out estimator.

Value

A function value table (object of class "fv"). Essentially a data frame containing the variables

<i>r</i>	the vector of values of the argument <i>r</i> at which the inhomogeneous multitype pair correlation function $g_{i\bullet}(r)$ has been estimated
<code>theo</code>	vector of values equal to 1, the theoretical value of $g_{i\bullet}(r)$ for the Poisson process

`trans` vector of values of $g_{i\bullet}(r)$ estimated by translation correction
`iso` vector of values of $g_{i\bullet}(r)$ estimated by Ripley isotropic correction
as required.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[pcf.ppp](#), [pcfinhom](#), [pcfdot](#), [pcfcross.inhom](#)

Examples

```
data(amacrine)
plot(pcfdot.inhom(amacrine, "on", stoyan=0.1), legendpos="bottom")
```

pcfinhom

Inhomogeneous Pair Correlation Function

Description

Estimates the inhomogeneous pair correlation function of a point pattern using kernel methods.

Usage

```
pcfinhom(X, lambda = NULL, ..., r = NULL,
          kernel = "epanechnikov", bw = NULL, stoyan = 0.15,
          correction = c("translate", "Ripley"),
          renormalise = TRUE, normpower=1,
          reciplambda = NULL,
          sigma = NULL, varcov = NULL)
```

Arguments

<code>X</code>	A point pattern (object of class "ppp").
<code>lambda</code>	Optional. Values of the estimated intensity function. Either a vector giving the intensity values at the points of the pattern <code>X</code> , a pixel image (object of class "im") giving the intensity values at all locations, a fitted point process model (object of class "ppm") or a function(<code>x,y</code>) which can be evaluated to give the intensity value at any location.
<code>r</code>	Vector of values for the argument r at which $g(r)$ should be evaluated. There is a sensible default.
<code>kernel</code>	Choice of smoothing kernel, passed to density.default .
<code>bw</code>	Bandwidth for smoothing kernel, passed to density.default .
<code>...</code>	Other arguments passed to the kernel density estimation function density.default .
<code>stoyan</code>	Bandwidth coefficient; see Details.
<code>correction</code>	Choice of edge correction.

<code>renormalise</code>	Logical. Whether to renormalise the estimate. See Details.
<code>normpower</code>	Integer (usually either 1 or 2). Normalisation power. See Details.
<code>reciplambda</code>	Alternative to <code>lambda</code> . Values of the estimated <i>reciprocal</i> $1/\lambda$ of the intensity function. Either a vector giving the reciprocal intensity values at the points of the pattern <code>X</code> , a pixel image (object of class "im") giving the reciprocal intensity values at all locations, or a function(<code>x, y</code>) which can be evaluated to give the reciprocal intensity value at any location.
<code>sigma, varcov</code>	Optional arguments passed to <code>density.ppp</code> to control the smoothing bandwidth, when <code>lambda</code> is estimated by kernel smoothing.

Details

The inhomogeneous pair correlation function $g_{\text{inhom}}(r)$ is a summary of the dependence between points in a spatial point process that does not have a uniform density of points.

The best intuitive interpretation is the following: the probability $p(r)$ of finding two points at locations x and y separated by a distance r is equal to

$$p(r) = \lambda(x)\lambda(y)g(r) dx dy$$

where λ is the intensity function of the point process. For a Poisson point process with intensity function λ , this probability is $p(r) = \lambda(x)\lambda(y)$ so $g_{\text{inhom}}(r) = 1$.

The inhomogeneous pair correlation function is related to the inhomogeneous K function through

$$g_{\text{inhom}}(r) = \frac{K'_{\text{inhom}}(r)}{2\pi r}$$

where $K'_{\text{inhom}}(r)$ is the derivative of $K_{\text{inhom}}(r)$, the inhomogeneous K function. See `Kinhom` for information about $K_{\text{inhom}}(r)$.

The command `pcfinhom` estimates the inhomogeneous pair correlation using a modified version of the algorithm in `pcf.ppp`.

If `renormalise=TRUE` (the default), then the estimates are multiplied by $c \cdot \text{normpower}$ where $c = \text{area}(W)/\sum(1/\lambda(x_i))$. This rescaling reduces the variability and bias of the estimate in small samples and in cases of very strong inhomogeneity. The default value of `normpower` is 1 but the most sensible value is 2, which would correspond to rescaling the `lambda` values so that $\sum(1/\lambda(x_i)) = \text{area}(W)$.

Value

A function value table (object of class "fv"). Essentially a data frame containing the variables

<code>r</code>	the vector of values of the argument r at which the inhomogeneous pair correlation function $g_{\text{inhom}}(r)$ has been estimated
<code>theo</code>	vector of values equal to 1, the theoretical value of $g_{\text{inhom}}(r)$ for the Poisson process
<code>trans</code>	vector of values of $g_{\text{inhom}}(r)$ estimated by translation correction
<code>iso</code>	vector of values of $g_{\text{inhom}}(r)$ estimated by Ripley isotropic correction

as required.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[pcf](#), [pcf.ppp](#), [Kinhom](#)

Examples

```
data(residualspaper)
X <- residualspaper$Fig4b
plot(pcfinhom(X, stoyan=0.2, sigma=0.1))
fit <- ppm(X, ~polynom(x,y,2))
plot(pcfinhom(X, lambda=fit, normpower=2))
```

perimeter

Perimeter Length of Window

Description

Computes the perimeter length of a window

Usage

```
perimeter(w)
```

Arguments

w A window (object of class "owin") or data that can be converted to a window by [as.owin](#).

Details

This function computes the perimeter (length of the boundary) of the window w. If w is a rectangle or a polygonal window, the perimeter is the sum of the lengths of the edges of w. If w is a mask, it is first converted to a polygonal window using [as.polygonal](#), then staircase edges are removed using [simplify.owin](#), and the perimeter of the resulting polygon is computed.

Value

A numeric value giving the perimeter length of the window.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[area.owin](#) [diameter.owin](#), [owin.object](#), [as.owin](#)

Examples

```

perimeter(square(3))
data(letterR)
perimeter(letterR)
if(spatstat.options("gpclib")) {
  if(interactive()) {
    print(perimeter(as.mask(letterR)))
  } else print(perimeter(as.mask(letterR, dimyx=32)))
}

```

periodify

Make Periodic Copies of a Spatial Pattern

Description

Given a spatial pattern (point pattern, line segment pattern, window, etc) make shifted copies of the pattern and optionally combine them to make a periodic pattern.

Usage

```

periodify(X, ...)
## S3 method for class 'ppp'
periodify(X, nx = 1, ny = 1, ...,
          combine=TRUE, warn=TRUE, check=TRUE,
          ix=(-nx):nx, iy=(-ny):ny,
          ixy=expand.grid(ix=ix,iy=iy))
## S3 method for class 'psp'
periodify(X, nx = 1, ny = 1, ...,
          combine=TRUE, warn=TRUE, check=TRUE,
          ix=(-nx):nx, iy=(-ny):ny,
          ixy=expand.grid(ix=ix,iy=iy))
## S3 method for class 'owin'
periodify(X, nx = 1, ny = 1, ...,
          combine=TRUE, warn=TRUE,
          ix=(-nx):nx, iy=(-ny):ny,
          ixy=expand.grid(ix=ix,iy=iy))

```

Arguments

X	An object representing a spatial pattern (point pattern, line segment pattern or window).
nx, ny	Integers. Numbers of additional copies of X in each direction. (Overruled by ix, iy, ixy).
...	Ignored.
combine	Logical flag determining whether the copies should be superimposed to make an object like X (if combine=TRUE) or simply returned as a list of objects (combine=FALSE).
warn	Logical flag determining whether to issue warnings.
check	Logical flag determining whether to check the validity of the combined pattern.
ix, iy	Integer vectors determining the grid positions of the copies of X. (Overruled by ixy).

`ixy` Matrix or data frame with two columns, giving the grid positions of the copies of `X`.

Details

Given a spatial pattern (point pattern, line segment pattern, etc) this function makes a number of shifted copies of the pattern and optionally combines them. The function `periodify` is generic, with methods for various kinds of spatial objects.

The default is to make a 3 by 3 array of copies of `X` and combine them into a single pattern of the same kind as `X`. This can be used (for example) to compute toroidal or periodic edge corrections for various operations on `X`.

If the arguments `ix`, `iy` or `ixy` are specified, then these determine the grid positions of the copies of `X` that will be made. For example `(ix, iy) = (1, 2)` means a copy of `X` shifted by the vector `(ix * w, iy * h)` where `w, h` are the width and height of the bounding rectangle of `X`.

If `combine=TRUE` (the default) the copies of `X` are superimposed to create an object of the same kind as `X`. If `combine=FALSE` the copies of `X` are returned as a list.

Value

If `combine=TRUE`, an object of the same class as `X`. If `combine=FALSE`, a list of objects of the same class as `X`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[shift](#)

Examples

```
data(cells)
plot(periodify(cells))
a <- lapply(periodify(cells$window, combine=FALSE),
            plot, add=TRUE, lty=2)
```

Description

Displays a perspective plot of a pixel image.

Usage

```
## S3 method for class 'im'
persp(x, ..., colmap=NULL)
```

Arguments

- `x` The pixel image to be plotted. An object of class "im" (see [im.object](#)).
- `...` Extra arguments passed to [persp.default](#) to control the display.
- `colmap` Optional data controlling the colour map. See Details.

Details

This is the `persp` method for the class "im".

The pixel image `x` must have real or integer values. These values are treated as heights of a surface, and the surface is displayed as a perspective plot on the current plot device, using equal scales on the `x` and `y` axes.

The optional argument `colmap` gives an easy way to display different altitudes in different colours (if this is what you want).

- If `colmap` is a colour map (object of class "colourmap", created by the function [colourmap](#)) then this colour map will be used to associate altitudes with colours.
- If `colmap` is a character vector, then the range of altitudes in the perspective plot will be divided into `length(colmap)` intervals, and those parts of the surface which lie in a particular altitude range will be assigned the corresponding colour from `colmap`.
- If `colmap` is a list with entries `breaks` and `col`, then `colmap$breaks` determines the break-points of the altitude intervals, and `colmap$col` provides the corresponding colours.

Graphical parameters controlling the perspective plot are passed through the ... arguments directly to the function [persp.default](#). See the examples in [persp.default](#) or in `demo(persp)`.

Value

Returns the 3D transformation matrix returned by [persp.default](#).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[im.object](#), [plot.im](#), [contour.im](#)

Examples

```
# an image
Z <- setcov(owin())
persp(Z, colmap=terrain.colors(128))
co <- colourmap(range=c(0,1), col=rainbow(128))
persp(Z, colmap=co, axes=FALSE, shade=0.3)
```

pixellate*Convert Spatial Object to Pixel Image*

Description

Convert a spatial object to a pixel image by measuring the amount of stuff in each pixel.

Usage

```
pixellate(x, ...)
```

Arguments

- | | |
|-----|--|
| x | Spatial object to be converted. A point pattern (object of class "ppp"), a window (object of class "owin"), a line segment pattern (object of class "psp"), or some other suitable data. |
| ... | Arguments passed to methods. |

Details

The function `pixellate` converts a geometrical object `x` into a pixel image, by measuring the *amount* of `x` that is inside each pixel.

If `x` is a point pattern, `pixellate(x)` counts the number of points of `x` falling in each pixel. If `x` is a window, `pixellate(x)` measures the area of intersection of each pixel with the window.

The function `pixellate` is generic, with methods for point patterns ([pixellate.ppp](#)), windows ([pixellate.owin](#)), and line segment patterns ([pixellate.psp](#)). See the separate documentation for these methods.

The related function [`as.im`](#) also converts `x` into a pixel image, but typically measures only the presence or absence of `x` inside each pixel.

Value

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

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[pixellate.ppp](#), [pixellate.owin](#), [pixellate.psp](#), [as.im](#)

`pixellate.owin`*Convert Window to Pixel Image*

Description

Convert a window to a pixel image by measuring the area of intersection between the window and each pixel in a raster.

Usage

```
## S3 method for class 'owin'  
pixellate(x, W = NULL, ...)
```

Arguments

- `x` Window (object of class "owin") to be converted.
- `W` Optional. Window determining the pixel raster on which the conversion should occur.
- `...` Optional. Extra arguments passed to `as.mask` to determine the pixel raster.

Details

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

It converts a window `x` into a pixel image, by measuring the *amount* of `x` that is inside each pixel.

(The related function `as.im` also converts `x` into a pixel image, but records only the presence or absence of `x` in each pixel.)

The pixel raster for the conversion is determined by the argument `W` and the extra arguments `...`.

- If `W` is given, and it is a binary mask (a window of type "mask") then it determines the pixel raster.
- If `W` is given, but it is not a binary mask (it is a window of another type) then it will be converted to a binary mask using `as.mask(W, ...)`.
- If `W` is not given, it defaults to `as.mask(as.rectangle(x), ...)`

In the second and third cases it would be common to use the argument `dimyx` to control the number of pixels. See the Examples.

The algorithm then computes the area of intersection of each pixel with the window.

The result is a pixel image with pixel entries equal to these intersection areas.

Value

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

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[pixellate.hpp](#), [pixellate](#), [as.im](#)

Examples

```
data(letterR)
plot(pixellate(letterR, dimyx=15))
W <- grow.rectangle(as.rectangle(letterR), 0.2)
plot(pixellate(letterR, W, dimyx=15))
```

pixellate.hpp

Convert Point Pattern to Pixel Image

Description

Converts a point pattern to a pixel image. The value in each pixel is the number of points falling in that pixel, and is typically either 0 or 1.

Usage

```
## S3 method for class 'ppp'
pixellate(x, W=NULL, ..., weights = NULL, padzero=FALSE)

## S3 method for class 'ppp'
as.im(X, ...)
```

Arguments

x, X	Point pattern (object of class "ppp").
...	Arguments passed to as.mask to determine the pixel resolution
W	Optional window mask (object of class "owin") determining the pixel raster.
weights	Optional vector of weights associated with the points.
padzero	Logical flag indicating whether to set pixel values to zero outside the window.

Details

The functions `pixellate.hpp` and `as.im.hpp` convert a spatial point pattern `x` into a pixel image, by counting the number of points (or the total weight of points) falling in each pixel.

Calling `as.im.hpp` is equivalent to calling `pixellate.hpp` with its default arguments. Note that `pixellate.hpp` is more general than `as.im.hpp` (it has additional arguments for greater flexibility).

The functions `as.im.hpp` and `pixellate.hpp` are methods for the generic functions `as.im` and `pixellate` respectively, for the class of point patterns.

The pixel raster (in which points are counted) is determined by the argument `W` if it is present (for `pixellate.hpp` only). In this case `W` should be a binary mask (a window object of class "owin" with type "mask"). Otherwise the pixel raster is determined by extracting the window containing `x` and converting it to a binary pixel mask using `as.mask`. The arguments `...` are passed to `as.mask` to control the pixel resolution.

If `weights` is `NULL`, then for each pixel in the mask, the algorithm counts how many points in `x` fall in the pixel. This count is usually either 0 (for a pixel with no data points in it) or 1 (for a pixel

containing one data point) but may be greater than 1. The result is an image with these counts as its pixel values.

If `weights` is given, it should be a numeric vector of the same length as the number of points in `x`. For each pixel, the algorithm finds the total weight associated with points in `x` that fall in the given pixel. The result is an image with these total weights as its pixel values.

By default (if `zeropad=FALSE`) the resulting pixel image has the same spatial domain as the window of the point pattern `x`. If `zeropad=TRUE` then the resulting pixel image has a rectangular domain; pixels outside the original window are assigned the value zero.

Value

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

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[pixellate](#), [im](#), [as.im](#), [density.ppp](#), [smooth.ppp](#).

Examples

```
data(humberside)
plot(pixellate(humberside))
```

pixellate.psp

Convert Line Segment Pattern to Pixel Image

Description

Converts a line segment pattern to a pixel image by measuring the length of lines intersecting each pixel.

Usage

```
## S3 method for class 'psp'
pixellate(x, W=NULL, ..., weights = NULL)
```

Arguments

- `x` Line segment pattern (object of class "psp").
- `W` Optional window (object of class "owin") determining the pixel resolution.
- `...` Optional arguments passed to [as.mask](#) to determine the pixel resolution.
- `weights` Optional vector of weights associated with each line segment.

Details

This function converts a line segment pattern to a pixel image by computing, for each pixel, the total length of intersection between the pixel and the line segments.

This is a method for the generic function [pixellate](#) for the class of line segment patterns.

The pixel raster is determined by `W` and the optional arguments If `W` is missing or `NULL`, it defaults to the window containing `x`. Then `W` is converted to a binary pixel mask using [as.mask](#). The arguments ... are passed to [as.mask](#) to control the pixel resolution.

If `weights` are given, then the length of the intersection between line segment `i` and pixel `j` is multiplied by `weights[i]` before the lengths are summed for each pixel.

Value

A pixel image (object of class "im") with numeric values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[pixellate](#), [as.mask](#), [as.mask.psp](#).

Use [as.mask.psp](#) if you only want to know which pixels are intersected by lines.

Examples

```
X <- psp(runif(10),runif(10), runif(10), runif(10), window=owin())
plot(pixelquad(X))
plot(X, add=TRUE)
sum(lengths.psp(X))
sum(pixelquad(X))
```

Description

Makes a quadrature scheme with a dummy point at every pixel of a pixel image.

Usage

```
pixelquad(X, W = as.owin(X))
```

Arguments

<code>X</code>	Point pattern (object of class "ppp") containing the data points for the quadrature scheme.
<code>W</code>	Specifies the pixel grid. A pixel image (object of class "im"), a window (object of class "owin"), or anything that can be converted to a window by as.owin .

Details

This is a method for producing a quadrature scheme for use by `ppm`. It is an alternative to `quadscheme`.

The function `ppm` fits a point process model to an observed point pattern using the Berman-Turner quadrature approximation (Berman and Turner, 1992; Baddeley and Turner, 2000) to the pseudo-likelihood of the model. It requires a quadrature scheme consisting of the original data point pattern, an additional pattern of dummy points, and a vector of quadrature weights for all these points. Such quadrature schemes are represented by objects of class "quad". See `quad.object` for a description of this class.

Given a grid of pixels, this function creates a quadrature scheme in which there is one dummy point at the centre of each pixel. The counting weights are used (the weight attached to each quadrature point is 1 divided by the number of quadrature points falling in the same pixel).

The argument `X` specifies the locations of the data points for the quadrature scheme. Typically this would be a point pattern dataset.

The argument `W` specifies the grid of pixels for the dummy points of the quadrature scheme. It should be a pixel image (object of class "im"), a window (object of class "owin"), or anything that can be converted to a window by `as.owin`. If `W` is a pixel image or a binary mask (a window of type "mask") then the pixel grid of `W` will be used. If `W` is a rectangular or polygonal window, then it will first be converted to a binary mask using `as.mask` at the default pixel resolution.

Value

An object of class "quad" describing the quadrature scheme (data points, dummy points, and quadrature weights) suitable as the argument `Q` of the function `ppm()` for fitting a point process model.

The quadrature scheme can be inspected using the `print` and `plot` methods for objects of class "quad".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`quadscheme`, `quad.object`, `ppm`

Examples

```
W <- owin(c(0,1),c(0,1))
X <- runifpoint(42, W)
W <- as.mask(W,dimyx=128)
pixelquad(X,W)
```

<code>plot.bermantest</code>	<i>Plot Result of Berman Test</i>
------------------------------	-----------------------------------

Description

Plot the result of Berman's test of goodness-of-fit

Usage

```
## S3 method for class 'bermantest'
plot(x, ...,
      lwd=par("lwd"), col=par("col"), lty=par("lty"),
      lwd0=lwd, col0=col, lty0=lty)
```

Arguments

<code>x</code>	Object to be plotted. An object of class "bermantest" produced by bermantest .
<code>...</code>	extra arguments that will be passed to the plotting function plot.ecdf .
<code>col, lwd, lty</code>	The width, colour and type of lines used to plot the empirical distribution.
<code>col0, lwd0, lty0</code>	The width, colour and type of lines used to plot the predicted distribution.

Details

This is the plot method for the class "bermantest". An object of this class represents the outcome of Berman's test of goodness-of-fit of a spatial Poisson point process model, computed by [bermantest](#).

For the $Z1$ test (i.e. if `x` was computed using `bermantest(, which="Z1")`), the plot displays the two cumulative distribution functions that are compared by the test: namely the empirical cumulative distribution function of the covariate at the data points, \hat{F} , and the predicted cumulative distribution function of the covariate under the model, F_0 , both plotted against the value of the covariate. Two vertical lines show the mean values of these two distributions. If the model is correct, the two curves should be close; the test is based on comparing the two vertical lines.

For the $Z2$ test (i.e. if `x` was computed using `bermantest(, which="Z2")`), the plot displays the empirical cumulative distribution function of the values $U_i = F_0(Y_i)$ where Y_i is the value of the covariate at the i -th data point. The diagonal line with equation $y = x$ is also shown. Two vertical lines show the mean of the values U_i and the value $1/2$. If the model is correct, the two curves should be close. The test is based on comparing the two vertical lines.

Value

NULL.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[bermantest](#)

Examples

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

# fit uniform Poisson process
fit0 <- ppm(X, ~1)

# test covariate = x coordinate
xcoord <- function(x,y) { x }

# test wrong model
k <- bermantest(fit0, xcoord, "Z1")

# plot result of test
plot(k, col="red", col0="green")
```

plot.colourmap

Plot a Colour Map

Description

Displays a colour map as a colour ribbon

Usage

```
## S3 method for class 'colourmap'
plot(x, ...,
      main, xlim = NULL, ylim = NULL, vertical = FALSE, axis = TRUE)
```

Arguments

<code>x</code>	Colour map to be plotted. An object of class "colourmap".
<code>...</code>	Graphical arguments passed to image.default .
<code>main</code>	Main title for plot. A character string.
<code>xlim</code>	Optional range of x values for the location of the colour ribbon.
<code>ylim</code>	Optional range of y values for the location of the colour ribbon.
<code>vertical</code>	Logical flag determining whether the colour ribbon is plotted as a horizontal strip (FALSE) or a vertical strip (TRUE).
<code>axis</code>	Logical flag determining whether an axis should be plotted showing the numerical values that are mapped to the colours.

Details

This is the plot method for the class "colourmap". An object of this class (created by the function [colourmap](#)) represents a colour map or colour lookup table associating colours with each data value.

The command `plot.colourmap` displays the colour map as a colour ribbon. This plot can be useful on its own to inspect the colour map.

To annotate an existing plot with an explanatory colour ribbon, specify `add=TRUE` and use the arguments `xlim` and/or `ylim` to control the physical position of the ribbon on the plot.

Value

None.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[colourmap](#)

Examples

```
co <- colourmap(rainbow(100), breaks=seq(-1,1,length=101))
plot(co)
plot(co, vertical=TRUE)
ca <- colourmap(rainbow(8), inputs=letters[1:8])
plot(ca)
```

plot.envelope

Plot a Simulation Envelope

Description

Plot method for the class "envelope".

Usage

```
## S3 method for class 'envelope'
plot(x, ..., shade=c("hi", "lo"))
```

Arguments

- | | |
|-------|---|
| x | An object of class "envelope", containing the variables to be plotted or variables from which the plotting coordinates can be computed. |
| ... | Extra arguments passed to plot.fv . |
| shade | An index that identifies two columns of x. When the corresponding curves are plotted, the region between the curves will be shaded in light grey. |

Details

This is the plot method for the class "envelope" of simulation envelopes. Objects of this class are created by the command [envelope](#).

This plot method is identical to [plot.fv](#) except that its default behaviour is to shade the region between the upper and lower envelopes in a light grey colour.

To suppress the shading and plot the upper and lower envelopes as curves, set shade=NULL.

To change the colour of the shading, use the argument shadecol which is passed to [plot.fv](#).

See [plot.fv](#) for further information on how to control the plot.

Value

Either NULL, or a data frame giving the meaning of the different line types and colours.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[envelope](#), [plot.fv](#)

Examples

```
data(cells)
E <- envelope(cells, Kest, nsim=19)
plot(E)
plot(E, sqrt(./pi) ~ r)
```

plot.fasp

Plot a Function Array

Description

Plots an array of summary functions, usually associated with a point pattern, stored in an object of class "fasp". A method for [plot](#).

Usage

```
## S3 method for class 'fasp'
plot(x, formule=NULL, ...,
      subset=NULL, title=NULL, samex=TRUE,
      banner=TRUE, mar.panel=NULL,
      outerlabels=TRUE, cex.outerlabels=1.25,
      legend=FALSE)
```

Arguments

- | | |
|---|--|
| x
formule
...
subset | An object of class "fasp" representing a function array.
A formula or list of formulae indicating what variables are to be plotted against what variable. Each formula is either an R language formula object, or a string that can be parsed as a formula. If formule is a list, its k^{th} component should be applicable to the $(i, j)^{th}$ plot where $x\$which[i, j]=k$. If the formula is left as NULL, then plot.fasp attempts to use the component default.formula of x . If that component is NULL as well, it gives up.
Arguments passed to plot.fv to control the individual plot panels.
A logical vector, or a vector of indices, or an expression or a character string, or a list of such, indicating a subset of the data to be included in each plot. If subset is a list, its k^{th} component should be applicable to the $(i, j)^{th}$ plot where $x\$which[i, j]=k$. |
|---|--|

title	Overall title for the plot.
samex	Logical flag indicating whether all individual plots should have the same x axis limits. This makes it easier to compare the plots. It can only be set to FALSE if you are using the default plot style (i.e. only when formule is missing).
banner	Logical. If TRUE, the overall title is plotted.
mar.panel	Vector of length 4 giving the value of the graphics parameter mar controlling the size of plot margins for each individual plot panel. See par .
outerlabels	Logical. If TRUE, the row and column names of the array of functions are plotted in the margins of the array of plot panels. If FALSE, each individual plot panel is labelled by its row and column name.
cex.outerlabels	Character expansion factor for row and column labels of array.
legend	Logical flag determining whether to plot a legend in each panel.

Details

An object of class "fasp" represents an array of summary functions, usually associated with a point pattern. See [fasp.object](#) for details. Such an object is created, for example, by [alltypes](#).

The function **plot.fasp** is a method for **plot**. It calls [plot.fv](#) to plot the individual panels.

For information about the interpretation of the arguments **formule** and **subset**, see [plot.fv](#).

Arguments that are often passed through ... include **col** to control the colours of the different lines in a panel, and **lty** and **lwd** to control the line type and line width of the different lines in a panel. The argument **shade** can also be used to display confidence intervals or significance bands as filled grey shading. See [plot.fv](#).

The argument **title**, if present, will determine the overall title of the plot. If it is absent, it defaults to **x\$title**. Titles for the individual plot panels will be taken from **x\$titles**.

Value

None.

Warnings

(Each component of) the **subset** argument may be a logical vector (of the same length as the vectors of data which are extracted from **x**), or a vector of indices, or an **expression** such as **expression(r<=0.2)**, or a text string, such as "r<=0.2".

Attempting a syntax such as **subset = r<=0.2** (without wrapping **r<=0.2** either in quote marks or in **expression()**) will cause this function to fall over.

Variables referred to in any formula must exist in the data frames stored in **x**. What the names of these variables are will of course depend upon the nature of **x**.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[alltypes](#), [plot.fv](#), [fasp.object](#)

Examples

```

## Not run:
# Bramble Canes data.
data(bramblecanes)

X.G <- alltypes(bramblecanes, "G", dataname="Bramblecanes", verb=TRUE)
plot(X.G)
plot(X.G,subset="r<=0.2")
plot(X.G,formule=asin(sqrt(cbind(km,theo))) ~ asin(sqrt(theo)))
plot(X.G,fo=cbind(km,theo) - theo~r,subset="r<=0.2")

# Simulated data.
pp <- runifpoint(350, owin(c(0,1),c(0,1)))
pp <- pp %mark% factor(c(rep(1,50),rep(2,100),rep(3,200)))
X.K <- alltypes(pp, "K", verb=TRUE,dataname="Fake Data")
plot(X.K,fo=cbind(border,theo)~theo,subset="theo<=0.75")

## End(Not run)

```

plot.fv

Plot Function Values

Description

Plot method for the class "fv".

Usage

```

## S3 method for class 'fv'
plot(x, fmla, ..., subset=NULL, lty=NULL, col=NULL, lwd=NULL,
      xlim=NULL, ylim=NULL, xlab=NULL, ylab=NULL, ylim.covers=NULL,
      legend=!add, legendpos="topleft", legendavoid=missing(legendpos),
      legendmath=TRUE, legendargs=list(),
      shade=NULL, shadecol="grey", add=FALSE, limitsonly=FALSE)

```

Arguments

x	An object of class "fv", containing the variables to be plotted or variables from which the plotting coordinates can be computed.
fmla	an R language formula determining which variables or expressions are plotted. Either a formula object, or a string that can be parsed as a formula. See Details.
subset	(optional) subset of rows of the data frame that will be plotted.
lty	(optional) numeric vector of values of the graphical parameter lty controlling the line style of each plot.
col	(optional) numeric vector of values of the graphical parameter col controlling the colour of each plot.
lwd	(optional) numeric vector of values of the graphical parameter lwd controlling the line width of each plot.
xlim	(optional) range of x axis
ylim	(optional) range of y axis

xlab	(optional) label for x axis
ylab	(optional) label for y axis
...	Extra arguments passed to <code>plot.default</code> .
ylim.covers	Optional vector of y values that must be included in the y axis. For example <code>ylim.covers=0</code> will ensure that the y axis includes the origin.
legend	Logical flag or <code>NULL</code> . If <code>legend=TRUE</code> , the algorithm plots a legend in the top left corner of the plot, explaining the meaning of the different line types and colours.
legendpos	The position of the legend. Either a character string keyword (see legend for keyword options) or a pair of coordinates in the format <code>list(x,y)</code> . Alternatively if <code>legendpos="f1oat"</code> , a location will be selected inside the plot region, avoiding the graphics.
legendavoid	Whether to avoid collisions between the legend and the graphics. Logical value. If <code>TRUE</code> , the code will check for collisions between the legend box and the graphics, and will override <code>legendpos</code> if a collision occurs. If <code>FALSE</code> , the value of <code>legendpos</code> is always respected.
legendmath	Logical. If <code>TRUE</code> , the legend will display the mathematical notation for each curve. If <code>FALSE</code> , the legend text is the identifier (column name) for each curve.
legendargs	Named list containing additional arguments to be passed to legend controlling the appearance of the legend.
shade	An index that identifies two columns of x . When the corresponding curves are plotted, the region between the curves will be shaded in light grey. Often used for displaying simulation envelopes, by setting <code>shade=c("hi", "lo")</code> .
shadecol	The colour to be used in the shade plot. A character string or an integer specifying a colour.
add	Logical. Whether the plot should be added to an existing plot
limitsonly	Logical. If <code>FALSE</code> , plotting is performed normally. If <code>TRUE</code> , no plotting is performed at all; just the x and y limits of the plot are computed and returned.

Details

This is the `plot` method for the class "`fv`".

The use of the argument `fmla` is like `plot.formula`, but offers some extra functionality.

The left and right hand sides of `fmla` are evaluated, and the results are plotted against each other (the left side on the y axis against the right side on the x axis).

The left and right hand sides of `fmla` may be the names of columns of the data frame x , or expressions involving these names. If a variable in `fmla` is not the name of a column of x , the algorithm will search for an object of this name in the environment where `plot.fv` was called, and then in the enclosing environment, and so on.

Multiple curves may be specified by a single formula of the form `cbind(y1,y2,...,yn) ~ x`, where x, y_1, y_2, \dots, y_n are expressions involving the variables in the data frame. Each of the variables y_1, y_2, \dots, y_n in turn will be plotted against x . See the examples.

Convenient abbreviations which can be used in the formula are

- the symbol `.` which represents all the variables in the data frame (other than the function argument itself);
- the symbol `.x` which represents the function argument;
- the symbol `.y` which represents the recommended value of the function.

The value returned by this plot function indicates the meaning of the line types and colours in the plot. It can be used to make a suitable legend for the plot if you want to do this by hand. See the examples.

The argument shade can be used to display critical bands or confidence intervals. If it is not NULL, then it should be a subset index for the columns of x , that identifies exactly 2 columns. When the corresponding curves are plotted, the region between the curves will be shaded in light grey. See the Examples.

The default values of lty, col and lwd can be changed using `spatstat.options("plot.fv")`.

Use type = "n" to create the plot region and draw the axes without plotting any data.

Use limitsonly=TRUE to suppress all plotting and just compute the x and y limits. This can be used to calculate common x and y scales for several plots.

Value

Either NULL, or a data frame giving the meaning of the different line types and colours.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`fv.object`, `Kest`

Examples

```

K <- Kest(cells)
# K is an object of class "fv"

plot(K, iso ~ r)           # plots iso against r

plot(K, sqrt(iso/pi) ~ r)  # plots sqrt(iso/r) against r

plot(K, cbind(iso,theo) ~ r) # plots iso against r AND theo against r

plot(K, . ~ r)              # plots all available estimates of K against r

plot(K, sqrt(. / pi) ~ r)   # plots all estimates of L-function
                            # L(r) = sqrt(K(r) / pi)

plot(K, cbind(iso,theo) ~ r, col=c(2,3))
                            # plots iso against r in colour 2
                            # and theo against r in colour 3

plot(K, iso ~ r, subset=quote(r < 0.2))
                            # plots iso against r for r < 10

# Can't remember the names of the columns? No problem..
plot(K, sqrt(. / pi) ~ .x)

# making a legend by hand
v <- plot(K, . ~ r, legend=FALSE)
legend("topleft", legend=v$meaning, lty=v$lty, col=v$col)

```

```

# significance bands
KE <- envelope(cells, Kest, nsim=19)
plot(KE, shade=c("hi", "lo"))

# how to display two functions on a common scale
Kr <- Kest(redwood)
a <- plot(K, limitsonly=TRUE)
b <- plot(Kr, limitsonly=TRUE)
xlim <- range(a$xlim, b$xlim)
ylim <- range(a$ylim, b$ylim)
opa <- par(mfrow=c(1,2))
plot(K, xlim=xlim, ylim=ylim)
plot(Kr, xlim=xlim, ylim=ylim)
par(opa)

```

plot.hyperframe*Plot Entries in a Hyperframe***Description**

Plots the entries in a hyperframe, in a series of panels, one panel for each row of the hyperframe.

Usage

```

## S3 method for class 'hyperframe'
plot(x, e, ..., main, arrange=TRUE,
      nrows=NULL, ncols=NULL,
      parargs=list(mar=c(1,1,3,1) * marsize),
      marsize=0.1)

```

Arguments

- x** Data to be plotted. A hyperframe (object of class "hyperframe", see [hyperframe](#)).
- e** How to plot each row. Optional. R language expression that will be evaluated in each row of the hyperframe to generate the plots.
- ...** Extra arguments controlling the plot (when e is missing).
- main** Overall title for the array of plots.
- arrange** Logical flag indicating whether to plot the objects side-by-side on a single page (`arrange=TRUE`) or plot them individually in a succession of frames (`arrange=FALSE`).
- nrows,ncols** Optional. The number of rows/columns in the plot layout (assuming `arrange=TRUE`). You can specify either or both of these numbers.
- parargs** Optional list of arguments passed to `par` before plotting each panel. Can be used to control margin sizes, etc.
- marsize** Optional scale parameter controlling the sizes of margins between the panels.

Details

This is the `plot` method for the class "hyperframe".

The argument `x` must be a hyperframe (like a data frame, except that the entries can be objects of any class; see [hyperframe](#)).

This function generates a series of plots, one plot for each row of the hyperframe. If `arrange=TRUE` (the default), then these plots are arranged in a neat array of panels within a single plot frame. If `arrange=FALSE`, the plots are simply executed one after another.

Exactly what is plotted, and how it is plotted, depends on the argument `e`. The default (if `e` is missing) is to plot only the first column of `x`. Each entry in the first column is plotted using the generic `plot` command, together with any extra arguments given in

If `e` is present, it should be an R language expression involving the column names of `x`. The expression will be evaluated once for each row of `x`. It will be evaluated in an environment where each column name of `x` is interpreted as meaning the object in that column in the current row. See the Examples.

Value

`NULL`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[hyperframe](#), [with.hyperframe](#)

Examples

```
H <- hyperframe(id=1:10)
H$X <- with(H, rpoispp(100))
H$D <- with(H, distmap(X))
# points only
plot(H[, "X"])
plot(H, plot(X, main=id))
# points superimposed on images
plot(H, {plot(D, main=id); plot(X, add=TRUE)})
```

Description

Plot a pixel image.

Usage

```
## S3 method for class 'im'
plot(x, ..., col=NULL, valuesAreColours=NULL,
      ribbon=TRUE,
      ribside=c("right", "left", "bottom", "top"),
      ribsep=0.15, ribwid=0.05, ribn=1024,
      ribscale=1, ribargs=list())
```

Arguments

<code>x</code>	The pixel image to be plotted. An object of class "im" (see im.object).
<code>...</code>	Extra arguments passed to image.default to control the plot. See Details.
<code>col</code>	Colours for displaying the pixel values. Either a character vector of colour values, or an object of class colourmap .
<code>valuesAreColours</code>	Logical value. If TRUE, the pixel values of <code>x</code> are to be interpreted as colour values.
<code>ribbon</code>	Logical flag indicating whether to display a ribbon showing the colour map.
<code>ribside</code>	Character string indicating where to display the ribbon relative to the main image.
<code>ribsep</code>	Factor controlling the space between the ribbon and the image.
<code>ribwid</code>	Factor controlling the width of the ribbon.
<code>ribn</code>	Number of different values to display in the ribbon.
<code>ribscale</code>	Rescaling factor for tick marks. The values on the numerical scale printed beside the ribbon will be multiplied by this rescaling factor.
<code>ribargs</code>	List of additional arguments passed to image.default and axis to control the display of the ribbon and its scale axis. These may override the ... arguments.

Details

This is the plot method for the class "im". [It is also the [image](#) method for "im".]

The pixel image `x` is displayed on the current plot device, using equal scales on the `x` and `y` axes.

If `ribbon=TRUE`, a legend will be plotted. The legend consists of a colour ribbon and an axis with tick-marks, showing the correspondence between the pixel values and the colour map.

By default, the ribbon is placed at the right of the main image. This can be changed using the argument `ribside`.

Arguments `ribsep`, `ribwid`, `ribn` control the appearance of the ribbon. The width of the ribbon is `ribwid` times the size of the pixel image, where 'size' means the larger of the width and the height. The distance separating the ribbon and the image is `ribsep` times the size of the pixel image. The ribbon contains `ribn` different numerical values, evenly spaced between the minimum and maximum pixel values in the image `x`, rendered according to the chosen colour map.

Arguments `ribscale`, `ribargs` control the annotation of the colour ribbon. To plot the colour ribbon without the axis and tick-marks, use `ribargs=list(axes=FALSE)`.

Normally the pixel values are displayed using the colours given in the argument `col`. This may be either an explicit colour map (an object of class "colourmap", created by the function [colourmap](#)) or a character vector or integer vector that specifies a set of colours.

If `col` is an explicit colour map (an object of class "colourmap") then the same colour always represents the same numeric value. For example this ensures that when we plot different images, the colour maps are consistent.

If `col` is a character vector or integer vector that specifies colours, then the colour mapping will be stretched to match the range of pixel values in the image `x`. In this case, the mapping of pixel values to colours is determined as follows.

logical-valued images: the values FALSE and TRUE are mapped to the colours `col[1]` and `col[2]` respectively. The vector `col` should have length 2.

factor-valued images: the factor levels `levels(x)` are mapped to the entries of `col` in order. The vector `col` should have the same length as `levels(x)`.

numeric-valued images: By default, the range of pixel values in `x` is divided into `n = length(col)` equal subintervals, which are mapped to the colours in `col`. (If `col` was not specified, it defaults to a vector of 255 colours.)

Alternatively if the argument `zlim` is given, it should be a vector of length 2 specifying an interval of real numbers. This interval will be used instead of the range of pixel values. The interval from `zlim[1]` to `zlim[2]` will be mapped to the colours in `col`. This facility enables the user to plot several images using a consistent colour map.

Alternatively if the argument `breaks` is given, then this specifies the endpoints of the subintervals that are mapped to each colour. This is incompatible with `zlim`.

The arguments `col` and `zlim` or `breaks` are passed to the function `image.default`. For examples of the use of these arguments, see `image.default`.

Other graphical parameters controlling the display of both the pixel image and the ribbon can be passed through the ... arguments to the function `image.default`. A parameter is handled only if it is one of the following:

- a formal argument of `image.default` that is operative when `add=TRUE`.
- one of the parameters "main", "asp", "sub", "axes", "ann", "cex", "font", "cex.axis", "cex.lab" described in `par`.
- the argument `box`, a logical value specifying whether a box should be drawn.

By default, images are plotted using image rasters rather than polygons, by setting `useRaster=TRUE` in `image.default`.

Alternatively, the pixel values could be directly interpretable as colour values in R. That is, the pixel values could be character strings that represent colours, or values of a factor whose levels are character strings representing colours.

- If `valuesAreColours=TRUE`, then the pixel values will be interpreted as colour values and displayed using these colours.
- If `valuesAreColours=FALSE`, then the pixel values will *not* be interpreted as colour values, even if they could be.
- If `valuesAreColours=NULL`, the algorithm will guess what it should do. If the argument `col` is given, the pixel values will *not* be interpreted as colour values. Otherwise, if all the pixel values are strings that represent colours, then they will be interpreted and displayed as colours.

If pixel values are interpreted as colours, the arguments `col` and `ribbon` will be ignored, and a ribbon will not be plotted.

Value

The colour map used. An object of class "colourmap".

Image Rendering Errors and Problems

The help for `image.default` explains that errors may occur, or images may be rendered incorrectly, on some devices, depending on the availability of colours and other device-specific constraints.

An error may occur on some graphics devices if the image is very large. If this happens, try setting `useRaster=FALSE` in the call to `plot.im`.

The error message `useRaster=TRUE` can only be used with a regular grid means that the x and y coordinates of the pixels in the image are not perfectly equally spaced, due to numerical rounding. This occurs with some images created by earlier versions of `spatstat`. To repair the coordinates in an image `X`, type `X <- as.im(X)`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`im.object`, `colourmap`, `contour.im`, `persp.im`, `image.default`, `spatstat.options`

Examples

```
# an image
Z <- setcov(owin())
plot(Z)
plot(Z, ribside="bottom")
# stretchable colour map
plot(Z, col=terrain.colors(128), axes=FALSE)
# fixed colour map
tc <- colourmap(rainbow(128), breaks=seq(-1,2,length=129))
plot(Z, col=tc)
# tweaking the plot
plot(Z, main="La vie en bleu", col.main="blue", cex.main=1.5,
      box=FALSE,
      ribargs=list(col.axis="blue", col.ticks="blue", cex.axis=0.75))
```

`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, ...)
```

Arguments

- | | |
|----------------|--|
| <code>x</code> | Influence measure (object of class "influence.ppm") computed by <code>influence.ppm</code> . |
| ... | Arguments passed to <code>plot.ppp</code> to control the plotting. |

Details

This is the plot method for objects of class "influence.ppm". These objects are computed by the command [influence.ppm](#).

The display shows circles centred at the data points with radii proportional to the influence values.

Value

None.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

References

Baddeley, A. and Chang, Y.M. and Song, Y. (2011) Leverage and influence diagnostics for spatial point process models. *Scandinavian Journal of Statistics*, in press.

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"))
```

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.

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`.

By default, this command will first plot the fitted intensity of the model, using `plot.ppm`, and then plot the empirical and fitted summary statistics, using `plot.fv`.

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".

The option `what="intensity"` will be ignored if the model is stationary.

Value

Null.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`kppm`, `plot.ppm`, `plot.minconfit`

Examples

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

`plot.kstest`

Plot a Spatial Kolmogorov-Smirnov Test

Description

Plot the result of a spatial Kolmogorov-Smirnov test

Usage

```
## S3 method for class 'kstest'
plot(x, ...,
      lwd=par("lwd"), col=par("col"), lty=par("lty"),
      lwd0=lwd, col0=col, lty0=lty)
```

Arguments

- `x` Object to be plotted. An object of class "kstest" produced by a method for `kstest`.
- `...` extra arguments that will be passed to the plotting function `plot.default`.
- `col, lwd, lty` The width, colour and type of lines used to plot the empirical distribution.
- `col0, lwd0, lty0` The width, colour and type of lines used to plot the predicted distribution.

Details

This is the plot method for the class "kstest". An object of this class represents the outcome of a spatial Kolmogorov-Smirnov test, computed by [kstest](#).

The plot displays the two cumulative distribution functions that are compared by the test: namely the empirical cumulative distribution function of the covariate at the data points, and the predicted cumulative distribution function of the covariate under the model, both plotted against the value of the covariate.

Value

NULL.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[kstest](#)

Examples

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

# fit uniform Poisson process
fit0 <- ppm(X, ~1)

# test covariate = x coordinate
xcoord <- function(x,y) { x }

# test wrong model
k <- kstest(fit0, xcoord)

# plot result of test
plot(k, lwd0=3)
```

Description

Generates a layered plot. The plot method for objects of class "layered".

Usage

```
## S3 method for class 'layered'
plot(x, ..., which = NULL, plotargs = NULL)
```

Arguments

- x An object of class "layered" created by the function [layered](#).
- ... Arguments to be passed to the `plot` method for *every* layer.
- which Subset index specifying which layers should be plotted.
- plotargs Arguments to be passed to the `plot` methods for individual layers. A list of lists of arguments of the form `name=value`.

Details

Layering is a simple mechanism for controlling a high-level plot that is composed of several successive plots, for example, a background and a foreground plot. The layering mechanism makes it easier to plot, to switch on or off the plotting of each individual layer, and to control the plotting arguments that are passed to each layer.

The layers of data to be plotted should first be converted into a single object of class "layered" using the function [layered](#). Then the layers can be plotted using the method `plot.layered`.

Value

Null.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[layered](#), [plot](#).

Examples

```
data(cells)
D <- distmap(cells)
L <- layered(D, cells)
plot(L)
plot(L, which = 2)
plot(L, plotargs=list(list(ribbon=FALSE), list(pch=3, cols="white")))
```

Description

Plots a leverage function that has been computed by [leverage.ppm](#).

Usage

```
## S3 method for class 'leverage.ppm'
plot(x, ..., showcut=TRUE)
```

Arguments

- x Leverage measure (object of class "leverage.ppm") computed by [leverage.ppm](#).
- ... Arguments passed to [plot.im](#) controlling the image plot.
- showcut Logical. If TRUE, a contour line is plotted at the level equal to the theoretical mean of the leverage.

Details

This is the plot method for objects of class "leverage.ppm". These objects are computed by the command [leverage.ppm](#).

Value

None.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

References

Baddeley, A. and Chang, Y.M. and Song, Y. (2011) Leverage and influence diagnostics for spatial point process models. *Scandinavian Journal of Statistics*, in press.

See Also

[leverage.ppm](#)

Examples

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

plot.linim

Plot Pixel Image on Linear Network

Description

Given a pixel image on a linear network, the pixel values are displayed either as colours or as line widths.

Usage

```
## S3 method for class 'linim'
plot(x, ..., style = c("colour", "width"), scale, adjust = 1)
```

Arguments

<code>x</code>	The pixel image to be plotted. An object of class "linim".
<code>...</code>	Extra graphical parameters, passed to <code>plot.im</code> if <code>style="colour"</code> , or to <code>polygon</code> if <code>style="width"</code> .
<code>style</code>	Character string specifying the type of plot. See Details.
<code>scale</code>	Physical scale factor for representing the pixel values as line widths.
<code>adjust</code>	Adjustment factor for the default scale.

Details

This is the `plot` method for objects of class "linim". Such an object represents a pixel image defined on a linear network.

If `style="colour"` (the default) then the pixel values of `x` are plotted as colours, using `plot.im`.

If `style="width"` then the pixel values of `x` are used to determine the widths of thick lines centred on the line segments of the linear network.

Value

Null.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

References

Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. To appear in *Scandinavian Journal of Statistics*.

See Also

`linim`, `plot.im`, `polygon`

Examples

```
example(linim)
plot(X)
plot(X, style="width")
```

Description

Plots a linear network

Usage

```
## S3 method for class 'linnet'
plot(x, ..., main=NULL, add=FALSE,
      vertices=FALSE, window=FALSE)
```

Arguments

<code>x</code>	Linear network (object of class "linnet").
<code>...</code>	Arguments passed to <code>plot.psp</code> controlling the plot.
<code>main</code>	Main title for plot. Use <code>main=""</code> to suppress it.
<code>add</code>	Logical. If codeTRUE, superimpose the graphics over the current plot. If FALSE, generate a new plot.
<code>vertices</code>	Logical. Whether to plot the vertices as well.
<code>window</code>	Logical. Whether to plot the window containing the linear network.

Details

This is the plot method for class "linnet".

Value

Null.

Author(s)

Ang Qi Wei <aqw07398@hotmail.com> and Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

See Also

[linnet](#)

Examples

```
example(linnet)
plot(letterA)
```

`plot.listof`

Plot a List of Things

Description

Plots a list of things

Usage

```
## S3 method for class 'listof'
plot(x, ..., main, arrange=TRUE,
      nrows=NULL, ncols=NULL, main.panel=NULL, mar.panel=c(2,1,1,2),
      panel.begin=NULL, panel.end=NULL, panel.args=NULL, plotcommand="plot",
      adorn.left=NULL, adorn.right=NULL, adorn.bottom=NULL, adorn.size=0.2)
```

Arguments

<code>x</code>	An object of the class "listof". Essentially a list of objects.
<code>...</code>	Arguments passed to <code>plot</code> when generating each plot panel.
<code>main</code>	Overall heading for the plot.
<code>arrange</code>	Logical flag indicating whether to plot the objects side-by-side on a single page (<code>arrange=TRUE</code>) or plot them individually in a succession of frames (<code>arrange=FALSE</code>).
<code>nrows, ncols</code>	Optional. The number of rows/columns in the plot layout (assuming <code>arrange=TRUE</code>). You can specify either or both of these numbers.
<code>main.panel</code>	Optional. A character string, or a vector of character strings, giving the headings for each of the objects.
<code>mar.panel</code>	Value of the graphics parameter <code>mar</code> controlling the size of the margins outside each plot panel. See the help file for <code>par</code> .
<code>panel.begin, panel.end</code>	Optional. Functions that will be executed before and after each panel is plotted. See Details.
<code>panel.args</code>	Internal use only.
<code>plotcommand</code>	Optional. Character string containing the name of the command that should be executed to plot each panel.
<code>adorn.left, adorn.right, adorn.bottom</code>	Optional. Functions (with no arguments) that will be executed to generate additional plots at the margins (left, right, and/or bottom, respectively) of the array of plots.
<code>adorn.size</code>	Relative width (as a fraction of the other panels' widths) of the margin plots.

Details

This is the `plot` method for the class "listof".

An object of class "listof" (defined in the base R package) represents a list of objects, all belonging to a common class. The base R package defines a method for printing these objects, `print.listof`, but does not define a method for `plot`. So here we have provided a method for `plot`.

In the `spatstat` package, the function `density.splitppp` produces an object of class "listof", essentially a list of pixel images. These images can be plotted in a nice arrangement using `plot.listof`. See the Example.

The arguments `panel.begin` and `panel.end` may be functions that will be executed before and after each panel is plotted. They will be called as `panel.begin(i, y, main=main.panel[i])` and `panel.end(i, y, add=TRUE)`.

Alternatively, `panel.begin` and `panel.end` may be objects of some class that can be plotted with the generic `plot` command. They will be plotted before and after each panel is plotted.

If all entries of `x` are pixel images, the function `image.listof` is called to control the plotting. The arguments `equal.ribbon` and `col` can be used to determine the colour map or maps applied.

Value

Null.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

`print.listof`, `contour.listof`, `image.listof`, `density.splitppp`

Examples

```
# Multitype point pattern
data(amacrine)
plot(D <- density(split(amacrine)))
plot(D, main="", equal.ribbon=TRUE,
      panel.end=function(i,y,...){contour(y, ...)})
```

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, ...)
```

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.

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`.

Value

none.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[msr](#), [smooth.msr](#), [plot.im](#), [plot.ppp](#)

Examples

```
example(msr)
plot(rp)
plot(rs)
```

plot.owin

Plot a Spatial Window

Description

Plot a two-dimensional window of observation for a spatial point pattern

Usage

```
## S3 method for class 'owin'
plot(x, main, add=FALSE, ..., box, edge=0.04,
      hatch=FALSE, angle=45, spacing=diameter(x)/50,
      invert=FALSE)
```

Arguments

- | | |
|----------------|--|
| x | The window to be plotted. An object of class owin , or data which can be converted into this format by as.owin() . |
| main | text to be displayed as a title above the plot. |
| add | logical flag: if TRUE, draw the window in the current plot; if FALSE, generate a new plot. |
| ... | extra arguments passed to the generic plot function. |
| box | logical flag; if TRUE, plot the enclosing rectangular box |
| edge | nonnegative number; the plotting region will have coordinate limits that are 1 + edge times as large as the limits of the rectangular box that encloses the pattern. |
| hatch | logical flag; if TRUE, the interior of the window will be shaded by a grid of parallel lines. |
| angle | orientation of the shading lines (in degrees anticlockwise from the <i>x</i> axis) when hatch =TRUE. |
| spacing | spacing between the shading lines, when hatch =TRUE. |
| invert | logical flag; when the window is a binary pixel mask, the mask colours will be inverted if invert =TRUE. |

Details

This is the plot method for the class `owin`. The action is to plot the boundary of the window on the current plot device, using equal scales on the x and y axes.

If the window `x` is of type "rectangle" or "polygonal", the boundary of the window is plotted as a polygon or series of polygons. If `x` is of type "mask" the discrete raster approximation of the window is displayed as a binary image (white inside the window, black outside).

Graphical parameters controlling the display (e.g. setting the colours) may be passed directly via the `...` arguments, or indirectly reset using `spatstat.options`.

When `x` is of type "rectangle" or "polygonal", it is plotted by the R function `polygon`. To control the appearance (colour, fill density, line density etc) of the polygon plot, determine the required argument of `polygon` and pass it through `...`. For example, to paint the interior of the polygon in red, use the argument `col="red"`. To draw the polygon edges in green, use `border="green"`. To suppress the drawing of polygon edges, use `border=NA`.

When `x` is of type "mask", it is plotted by `image.default`. The appearance of the image plot can be controlled by passing arguments to `image.default` through `...`. The default appearance can also be changed by setting the parameter `par.binary` of `spatstat.options`.

To zoom in (to view only a subset of the window at higher magnification), use the graphical arguments `xlim` and `ylim` to specify the desired rectangular field of view. (The actual field of view may be larger, depending on the graphics device).

Value

none.

Notes on Filled Polygons with Holes

The function `polygon` can only handle polygons without holes. To plot polygons with holes in a solid colour, we have implemented two workarounds.

polypath function: The first workaround uses the relatively new function `polypath` which *does* have the capability to handle polygons with holes. However, not all graphics devices support `polypath`. The older devices `xfig` and `pictex` do not support `polypath`. On a Windows system, the default graphics device windows supports `polypath`. On a Linux system, the default graphics device `X11(type="Xlib")` does *not* support `polypath` but `X11(type="cairo")` does support it. See `X11` and the section on Cairo below.

gpclib package: The other workaround requires use of the `gpclib` package which effects a triangulation of the polygonal window in question which in turn permits the plotting of the window, filled with a solid colour, without filling the holes. Type `licence.polygons()` for information about the licence under which `gpclib` operates. To make use of this workaround you need to have the package `gpclib` installed and to set `spatstat.options(gpclib=TRUE)` (and to be working under circumstances permitted by the `gpclib` licence).

Cairo graphics on a Linux system

Linux systems support the graphics device `X11(type="cairo")` (see `X11`) provided the external library `cairo` is installed on the computer. See <http://www.cairographics.org/download> for instructions on obtaining and installing `cairo`. After having installed `cairo` one needs to re-install R from source so that it has `cairo` capabilities. To check whether your current installation of R has `cairo` capabilities, type (in R) `capabilities()["cairo"]`. The default type for `X11` is controlled by `X11.options`. You may find it convenient to make `cairo` the default, e.g. via your `.Rprofile`. The magic incantation to put into `.Rprofile` is

```
setHook(packageEvent("graphics", "onLoad"),
       function(...) grDevices::X11.options(type="cairo"))
```

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[owin.object](#), [plot.ppp](#), [polygon](#), [image.default](#), [spatstat.options](#)

Examples

```
# rectangular window
data(nztrees)
plot(nztrees>window)
abline(v=148, lty=2)

# polygonal window
data(demopat)
w <- demopat>window
plot(w)
plot(w, col="red", border="green", lwd=2)
plot(w, hatch=TRUE, lwd=2)

# binary mask
we <- as.mask(erosion(w, 400, FALSE))
plot(we)
op <- spatstat.options(par.binary=list(col=grey(c(0.5,1))))
plot(we)
spatstat.options(op)
```

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"), ...)
```

Arguments

<code>x</code>	An object of class <code>plotppm</code> produced by <code>plot.ppm()</code> .
<code>data</code>	The point pattern (an object of class <code>ppp</code>) to which the point process model was fitted (by <code>ppm</code>).
<code>trend</code>	Logical scalar; should the trend component of the fitted model be plotted?
<code>cif</code>	Logical scalar; should the complete conditional intensity of the fitted model be plotted?
<code>se</code>	Logical scalar; should the estimated standard error of the fitted intensity be plotted?
<code>pause</code>	Logical scalar indicating whether to pause with a prompt after each plot. Set <code>pause=FALSE</code> if plotting to a file.
<code>how</code>	Character string or character vector indicating the style or styles of plots to be performed.
<code>...</code>	Extra arguments to the plotting functions <code>persp</code> , <code>image</code> and <code>contour</code> .

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`plot.ppm()`

Examples

```
## Not run:
data(cells)
Q <- quadscheme(cells)
m <- ppm(Q, ~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)

## End(Not run)
```

plot.pp3

Plot a three-dimensional point pattern

Description

Plots a three-dimensional point pattern.

Usage

```
## S3 method for class 'pp3'
plot(x, ...)
```

Arguments

x	Three-dimensional point pattern (object of class "pp3").
...	Arguments passed to scatterplot3d controlling the plot.

Details

This is the plot method for objects of class "pp3".

This function requires the **scatterplot3d** package. The coordinates of the point pattern are passed to the function **scatterplot3d** along with any extra arguments

Value

Null.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[pp3](#)

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[[i]]` 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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`plot.plotppm`, `ppm`, `ppm.object`, `predict.ppm`, `print.ppm`, `persp`, `image`, `contour`, `plot`, `spatstat.options`

Examples

```
## Not run:
data(cells)
m <- ppm(cells, ~1, Strauss(0.05))
pm <- plot(m) # The object ‘‘pm’’ will be plotted as well as saved
# for future plotting.

## End(Not run)
```

plot.ppp

plot a Spatial Point Pattern

Description

Plot a two-dimensional spatial point pattern

Usage

```
## S3 method for class 'ppp'
plot(x, main, ..., chars=NULL, cols=NULL,
use.marks=TRUE, which.marks=1, add=FALSE,
maxsize=NULL, markscale=NULL, zap=0.01)
```

Arguments

x	The spatial point pattern to be plotted. An object of class "ppp", or data which can be converted into this format by as.ppp() .
main	text to be displayed as a title above the plot.
...	extra arguments that will be passed to the plotting functions plot.default , points and/or symbols .
chars	plotting character(s) used to plot points.
cols	the colour(s) used to plot points.
use.marks	logical flag; if TRUE, plot points using a different plotting symbol for each mark; if FALSE, only the locations of the points will be plotted, using points() .
which.marks	Index determining which column of marks to use, if the marks of x are a data frame. A character string or an integer. Defaults to 1 indicating the first column of marks.
add	logical flag; if TRUE, just the points are plotted, over the existing plot. A new plot is not created, and the window is not plotted.
maxsize	maximum size of the circles/squares plotted when x is a marked point pattern with numerical marks. Incompatible with markscale .
markscale	physical scale factor determining the sizes of the circles/squares plotted when x is a marked point pattern with numerical marks. Incompatible with maxsize .
zap	Fraction between 0 and 1. When x is a marked point pattern with numerical marks, zap is the smallest mark value (expressed as a fraction of the maximum possible mark) that will be plotted. Any points which have marks smaller in absolute value than $zap * \max(\text{abs}(\text{marks}(x)))$ will not be plotted.

Details

This is the plot method for point pattern datasets (of class "ppp", see [ppp.object](#)).

First the observation window $x>window$ is plotted. Then the points themselves are plotted, in a fashion that depends on their marks, as follows.

unmarked point pattern: If the point pattern does not have marks, or if $use.marks = FALSE$, then the locations of all points will be plotted using a single plot character

multitype point pattern: If xmarks$ is a factor, then each level of the factor is represented by a different plot character.

continuous marks: If xmarks$ is a numeric vector, the marks are rescaled to the unit interval and each point is represented by a circle with radius proportional to the rescaled mark (if the value is positive) or a square with side proportional to the absolute value of the rescaled mark (if the value is negative).

other kinds of marks: If xmarks$ is neither numeric nor a factor, then each possible mark will be represented by a different plotting character. The default is to represent the i th smallest mark value by $\text{points}(\dots, \text{pch}=i)$.

Plotting of the window $x>window$ is performed by [plot.owin](#). This plot may be modified through the \dots arguments. In particular the extra argument `border` determines the colour of the window.

Plotting of the points themselves is performed by the function [points](#), except for the case of continuous marks, where it is performed by [symbols](#). Their plotting behaviour may be modified through the \dots arguments.

The argument `chars` determines the plotting character or characters used to display the points (in all cases except for the case of continuous marks). For an unmarked point pattern, this should be a single integer or character determining a plotting character (see `par("pch")`). For a multitype point pattern, `chars` should be a vector of integers or characters, of the same length as `levels(x$marks)`, and then the i th level or type will be plotted using character `chars[i]`.

If `chars` is absent, but there is an extra argument `pch`, then this will determine the plotting character for all points.

The argument `cols` determines the colour or colours used to display the points. For an unmarked point pattern, or a marked point pattern with continuous marks, this should be a character string determining a colour. For a multitype point pattern, `cols` should be a character vector, of the same length as `levels(x$marks)`. The i th level or type will be plotted using colour `cols[i]`.

If `cols` is absent, the colour used to plot *all* the points may be determined by the extra argument `fg` (for multitype point patterns) or the extra argument `col` (for all other cases). Note that `col` will also reset the colour of the window.

The arguments `maxsize` and `markscale` incompatible. They control the physical size of the circles and squares which represent the marks in a point pattern with continuous marks. If `markscale` is given, then a mark value of m is plotted as a circle of radius $m * \text{markscale}$ (if m is positive) or a square of side $\text{abs}(m) * \text{markscale}$ (if m is negative). If `maxsize` is given, then the largest mark in absolute value, $m_{\text{max}}=\text{max}(\text{abs}(x$marks))$, will be scaled to have physical size `maxsize`.

The user can set the default values of these plotting parameters using [spatstat.options\("par.points"\)](#).

To zoom in (to view only a subset of the point pattern at higher magnification), use the graphical arguments `xlim` and `ylim` to specify the rectangular field of view.

The value returned by this plot function can be used to make a suitable legend, as shown in the examples.

Value

NULL, or a vector giving the correspondence between mark values and plotting characters.

Removing White Space Around The Plot

A frequently-asked question is: How do I remove the white space around the plot? Currently *plot.ppp* uses the base graphics system of R, so the space around the plot is controlled by parameters to *par*. To reduce the white space, change the parameter *mar*. Typically, *par(mar=rep(0.5, 4))* is adequate, if there are no annotations or titles outside the window.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

iplot, ppp.object, plot, par, points, plot.owin, symbols

Examples

```

data(cells)
plot(cells)

plot(cells, pch=16)

# make the plotting symbols larger (for publication at reduced scale)
plot(cells, cex=2)

# set it in spatstat.options
oldopt <- spatstat.options(par.points=list(cex=2))
plot(cells)
spatstat.options(oldopt)

# multitype
data(lansing)
plot(lansing)

# marked by a real number
data(longleaf)
plot(longleaf)

# just plot the points
plot(longleaf, use.marks=FALSE)
plot(unmark(longleaf)) # equivalent

# controlling COLOURS of points
plot(cells, cols="blue")
plot(lansing, cols=c("black", "yellow", "green",
                     "blue", "red", "pink"))
plot(longleaf, fg="blue")

# make window purple
plot(lansing, border="purple")
# make everything purple
plot(lansing, border="purple", cols="purple", col.main="purple")

# controlling PLOT CHARACTERS
plot(lansing, chars = 11:16)
plot(lansing, chars = c("o", "h", "m", ".", "o", "o"))

```

```

# controlling MARK SCALE
plot(longleaf, markscale=0.1)

# draw circles of DIAMETER equal to nearest neighbour distance
plot(cells %mark% nndist(cells), markscale=1/2)

# making the legend
data(amacrine)
v <- plot(amacrine)
legend(0.2, 1.2, pch=v, legend=names(v))

# point pattern with multiple marks
data(finlpines)
plot(finlpines, which.marks="height")

```

plot.psp*plot a Spatial Line Segment Pattern***Description**

Plot a two-dimensional line segment pattern

Usage

```

## S3 method for class 'psp'
plot(x, ..., add=FALSE, which.marks=1,
      ribbon=TRUE, ribsep=0.15, ribwid=0.05, ribn=1024)

```

Arguments

x	The line segment pattern to be plotted. An object of class "psp", or data which can be converted into this format by as.psp() .
...	extra arguments that will be passed to the plotting functions segments (to plot the segments) and plot.owin (to plot the observation window).
add	Logical. If TRUE, the current plot is not erased; the segments are plotted on top of the current plot, and the window is not plotted.
which.marks	Index determining which column of marks to use, if the marks of x are a data frame. A character string or an integer. Defaults to 1 indicating the first column of marks.
ribbon	Logical flag indicating whether to display a ribbon showing the colour map (in which mark values are associated with colours).
ribsep	Factor controlling the space between the ribbon and the image.
ribwid	Factor controlling the width of the ribbon.
ribn	Number of different values to display in the ribbon.

Details

This is the `plot` method for line segment pattern datasets (of class "`psp`", see [psp.object](#)). It plots both the observation window `x$window` and the line segments themselves.

Plotting of the window `x$window` is performed by [plot.owin](#). This plot may be modified through the `...` arguments.

Plotting of the segments themselves is performed by the standard R function [segments](#). Its plotting behaviour may also be modified through the `...` arguments.

For a *marked* line segment pattern (i.e. if `marks(x)` is not `NULL`) the line segments are plotted in colours determined by the mark values. If `marks(x)` is a data frame, the default is to use the first column of `marks(x)` to determine the colours. To specify another column, use the argument `which.marks`. The colour map (associating mark values with colours) will be displayed as a vertical colour ribbon to the right of the plot, if `ribbon=TRUE`.

Value

`NULL`

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[psp.object](#), [plot](#), [par](#), [plot.owin](#), [symbols](#)

Examples

```
X <- psp(runif(20), runif(20), runif(20), runif(20), window=owin())
plot(X)
plot(X, lwd=3)
lettuce <- sample(letters[1:4], 20, replace=TRUE)
marks(X) <- data.frame(A=1:20, B=factor(lettuce))
plot(X)
plot(X, which.marks="B")
```

`plot.quad`

plot a Spatial Quadrature Scheme

Description

Plot a two-dimensional spatial quadrature scheme.

Usage

```
## S3 method for class 'quad'
plot(x, ..., main=deparse(substitute(x)), dum=list())
```

Arguments

- x The spatial quadrature scheme to be plotted. An object of class "quad".
... extra arguments controlling the plotting of the data points of the quadrature scheme.
dum list of extra arguments controlling the plotting of the dummy points of the quadrature scheme. See below.
main text to be displayed as a title above the plot.

Details

This is the `plot` method for quadrature schemes (objects of class "quad", see [quad.object](#)).

First the data points of the quadrature scheme are plotted (in their observation window) using `plot.ppp` with any arguments specified in ...

Then the dummy points of the quadrature scheme are plotted using `plot.ppp` with any arguments specified in `dum`.

By default the dummy points are superimposed onto the plot of data points. This can be overridden by including the argument `add=FALSE` in the list `dum` as shown in the examples. In this case the data and dummy point patterns are plotted separately.

See `par` and `plot.ppp` for other possible arguments controlling the plots.

Value

NULL.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[quad.object](#), [plot.ppp](#), `par`

Examples

```
data(nztrees)
Q <- quadscheme(nztrees)

plot(Q, main="NZ trees: quadrature scheme")

oldpar <- par(mfrow=c(2,1))
plot(Q, main="NZ trees", dum=list(add=FALSE))
par(oldpar)
```

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

- | | |
|-------------------|---|
| <code>x</code> | a fitted spatial logistic regression model. An object of class " <code>slrm</code> ". |
| <code>...</code> | Extra arguments passed to <code>plot.im</code> to control the appearance of the plot. |
| <code>type</code> | 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@maths.uwa.edu.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

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

Examples

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

<code>plot.splitppp</code>	<i>Plot a List of Point Patterns</i>
----------------------------	--------------------------------------

Description

Plots a list of point patterns.

Usage

```
## S3 method for class 'splitppp'
plot(x, ..., main, arrange=TRUE,
      nrows=NULL, ncols=NULL, main.panel=NULL, mar.panel=c(2,1,1,2),
      panel.begin=NULL, panel.end=NULL, panel.args=NULL, plotcommand="plot",
      adorn.left=NULL, adorn.right=NULL, adorn.bottom=NULL, adorn.size=0.2)
```

Arguments

<code>x</code>	A named list of point patterns, typically obtained from split.ppp .
<code>...</code>	Arguments passed to plot.ppp which control the appearance of each plot panel.
<code>main</code>	Overall heading for the plot.
<code>arrange</code>	Logical flag indicating whether to plot the point patterns side-by-side on a single page (<code>arrange=TRUE</code>) or plot them individually in a succession of frames (<code>arrange=FALSE</code>).
<code>nrows, ncols</code>	Optional. The number of rows/columns in the plot layout (assuming <code>arrange=TRUE</code>). You can specify either or both of these numbers.
<code>main.panel</code>	Optional. A character string, or a vector of character strings, giving the headings for each of the point patterns.
<code>mar.panel</code>	Optional value of the graphics parameter <code>mar</code> controlling the size of the margins outside each plot panel. See the help file for par .
<code>panel.begin, panel.end</code>	Optional. Functions that will be executed before and after each panel is plotted. See Details.
<code>panel.args</code>	Internal use only.
<code>plotcommand</code>	Optional. Character string containing the name of the command that should be executed to plot each panel.
<code>adorn.left, adorn.right, adorn.bottom</code>	Optional. Functions (with no arguments) that will be executed to generate additional plots at the margins (left, right, and/or bottom, respectively) of the array of plots.
<code>adorn.size</code>	Relative width (as a fraction of the other panels' widths) of the margin plots.

Details

This is the `plot` method for the class "splitppp". It is typically used to plot the result of the function [split.ppp](#) but it may also be used to plot any list of point patterns created by the user.

The argument `x` should be a named list of point patterns (objects of class "ppp", see [ppp.object](#)). Each of these point patterns will be plotted in turn using [plot.ppp](#).

The arguments `panel.begin` and `panel.end` may be functions that will be executed before and after each panel is plotted. They will be called as `panel.begin(i, y, main=main.panel[i])` and `panel.end(i, y, add=TRUE)`.

Alternatively, `panel.begin` and `panel.end` may be objects of some class that can be plotted with the generic `plot` command. They will be plotted before and after each panel is plotted.

Value

Null.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[split.ppp](#), [plot.ppp](#), [ppp.object](#)

Examples

```
# Multitype point pattern
data(amacrine)
plot(split(amacrine))
plot(split(amacrine), main="",
  panel.begin=function(i, y, ...) { plot(density(y), ribbon=FALSE, ...) })
```

plot.tess

Plot a tessellation

Description

Plots a tessellation.

Usage

```
## S3 method for class 'tess'
plot(x, ..., main, add=FALSE, col=NULL)
```

Arguments

<code>x</code>	Tessellation (object of class "tess") to be plotted.
<code>...</code>	Arguments controlling the appearance of the plot.
<code>main</code>	Heading for the plot. A character string.
<code>add</code>	Logical. Determines whether the tessellation plot is added to the existing plot.
<code>col</code>	Colour of the tile boundaries. A character string. Ignored for pixel tessellations.

Details

This is a method for the generic `plot` function for the class "tess" of tessellations (see [tess](#)).

The arguments `...` control the appearance of the plot. They are passed to [segments](#), [plot.owin](#) or [plot.im](#), depending on the type of tessellation.

Value

None.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[tess](#)

Examples

```
A <- tess(xgrid=0:4,ygrid=0:4)
plot(A, col="blue", lwd=2, lty=2)
B <- A[c(1, 2, 5, 7, 9)]
plot(B, hatch=TRUE)
v <- as.im(function(x,y){factor(round(5 * (x^2 + y^2)))}, W=owin())
levels(v) <- letters[seq(length(levels(v)))]
E <- tess(image=v)
plot(E)
```

pointsOnLines

Place Points Evenly Along Specified Lines

Description

Given a line segment pattern, place a series of points at equal distances along each line segment.

Usage

```
pointsOnLines(X, eps = NULL, np = 1000, shortok=TRUE)
```

Arguments

X	A line segment pattern (object of class "psp").
eps	Spacing between successive points.
np	Approximate total number of points (incompatible with eps).
shortok	Logical. If FALSE, very short segments (of length shorter than eps) will not generate any points. If TRUE, a very short segment will be represented by its midpoint.

Details

For each line segment in the pattern X, a succession of points is placed along the line segment. These points are equally spaced at a distance eps, except for the first and last points in the sequence.

The spacing eps is measured in coordinate units of X.

If eps is not given, then it is determined by $\text{eps} = \text{len}/\text{np}$ where len is the total length of the segments in X. The actual number of points will then be slightly larger than np.

Value

A point pattern (object of class "ppp") in the same window as X.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[psp](#), [ppp](#), [runifpointOnLines](#)

Examples

```
X <- psp(runif(20), runif(20), runif(20), runif(20), window=owin())
Y <- pointsOnLines(X, eps=0.05)
plot(X, main="")
plot(Y, add=TRUE, pch="+")
```

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppm](#), [Strauss](#)

Examples

```

data(nztrees)
ppm(nztrees, ~1, Poisson())
# fit the stationary Poisson process to 'nztrees'
# no edge correction needed

data(longleaf)

longadult <- longleaf[longleaf$marks >= 30, ]
longadult <- unmark(longadult)
ppm(longadult, ~ x, Poisson())
# fit the nonstationary Poisson process
# with intensity lambda(x,y) = exp( a + bx)

data(lansing)
# trees marked by species

ppm(lansing, ~ marks, Poisson())
# fit stationary marked Poisson process
# with different intensity for each species

## Not run:
ppm(lansing, ~ marks * polynom(x,y,3), Poisson())

## End(Not run)
# fit nonstationary marked Poisson process
# with different log-cubic trend for each species

```

ponderosa

Ponderosa Pine Tree Point Pattern

Description

The data record the locations of 108 Ponderosa Pine (*Pinus ponderosa*) trees in a 120 metre square region in the Klamath National Forest in northern California, published as Figure 2 of Getis and Franklin (1987).

Franklin et al. (1985) determined the locations of approximately 5000 trees from United States Forest Service aerial photographs and digitised them for analysis. Getis and Franklin (1987) selected a 120 metre square subregion that appeared to exhibit clustering. This subregion is the `ponderosa` dataset.

In principle these data are equivalent to Figure 2 of Getis and Franklin (1987) but they are not exactly identical; some of the spatial locations appear to be slightly perturbed.

The data points identified as A, B, C on Figure 2 of Getis and Franklin (1987) correspond to points numbered 42, 7 and 77 in the dataset respectively.

Usage

```
data(ponderosa)
```

Format

Typing `data(ponderosa)` gives access to two objects, `ponderosa` and `ponderosa.extra`.

The dataset `ponderosa` is a spatial point pattern (object of class "ppp") representing the point pattern of tree positions. See [ppp.object](#) for details of the format.

The dataset `ponderosa.extra` is a list containing supplementary data. The entry `id` contains the index numbers of the three special points A, B, C in the point pattern. The entry `plotit` is a function that can be called to produce a nice plot of the point pattern.

Source

Prof. Janet Franklin, University of California, Santa Barbara

References

Franklin, J., Michaelsen, J. and Strahler, A.H. (1985) Spatial analysis of density dependent pattern in coniferous forest stands. *Vegetatio* **64**, 29–36.

Getis, A. and Franklin, J. (1987) Second-order neighbourhood analysis of mapped point patterns. *Ecology* **68**, 473–477.

Examples

```
data(ponderosa)
ponderosa.extra$plotit()
```

`pool`

Pool Data

Description

Pool the data from several objects of the same class.

Usage

```
pool(...)
```

Arguments

...	Objects of the same type.
-----	---------------------------

Details

The function `pool` is generic. There are methods for several classes, listed below.

`pool` is used to combine the data from several objects of the same type, and to compute statistics based on the combined dataset. It may be used to pool the estimates obtained from replicated datasets. It may also be used in high-performance computing applications, when the objects ... have been computed on different processors or in different batch runs, and we wish to combine them.

Value

An object of the same class as the arguments

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[pool.envelope](#), [pool.fasp](#), [pool.rat](#)

pool.envelope

Pool Data from Several Envelopes

Description

Pool the simulation data from several simulation envelopes (objects of class "envelope") and compute a new envelope.

Usage

```
## S3 method for class 'envelope'  
pool(...)
```

Arguments

... Objects of class "envelope".

Details

The function [pool](#) is generic. This is the method for the class "envelope" of simulation envelopes. It is used to combine the simulation data from several simulation envelopes and to compute an envelope based on the combined data.

Each of the arguments ... must be an object of class "envelope" created by running the command [envelope](#) with the argument `savefuns=TRUE`. This ensures that each object contains the simulated data (summary function values for the simulated point patterns) that were used to construct the envelope.

The simulated data are extracted from each object and combined. A new envelope is computed from the combined set of simulations.

Warnings or errors will be issued if the envelope objects ... appear to be incompatible. However, the code is not smart enough to decide whether it is sensible to pool the data.

To modify the envelope parameters or the type of envelope that is computed, first pool the envelope data using [pool.envelope](#), then use [envelope.envelope](#) to modify the envelope parameters.

Value

An object of class "envelope".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[envelope](#), [envelope.envelope](#), [pool](#), [pool.fasp](#)

Examples

```
data(cells)
E1 <- envelope(cells, Kest, nsim=10, savefuns=TRUE)
E2 <- envelope(cells, Kest, nsim=20, savefuns=TRUE)
pool(E1, E2)
```

pool.fasp

Pool Data from Several Function Arrays

Description

Pool the simulation data from several function arrays (objects of class "fasp") and compute a new function array.

Usage

```
## S3 method for class 'fasp'
pool(...)
```

Arguments

... Objects of class "fasp".

Details

The function [pool](#) is generic. This is the method for the class "fasp" of function arrays. It is used to combine the simulation data from several arrays of simulation envelopes and to compute a new array of envelopes based on the combined data.

Each of the arguments ... must be a function array (object of class "fasp") containing simulation envelopes. This is typically created by running the command [alltypes](#) with the arguments `envelope=TRUE` and `savefuns=TRUE`. This ensures that each object is an array of simulation envelopes, and that each envelope contains the simulated data (summary function values) that were used to construct the envelope.

The simulated data are extracted from each object and combined. A new array of envelopes is computed from the combined set of simulations.

Warnings or errors will be issued if the objects ... appear to be incompatible. However, the code is not smart enough to decide whether it is sensible to pool the data.

Value

An object of class "fasp".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[fasp](#), [alltypes](#), [pool.envelope](#), [pool](#)

Examples

```
data(amacrine)
A1 <- alltypes(amacrine,"K",nsim=9,envelope=TRUE,savefuns=TRUE)
A2 <- alltypes(amacrine,"K",nsim=10,envelope=TRUE,savefuns=TRUE)
pool(A1, A2)
```

pool.rat

Pool Data from Several Ratio Objects

Description

Pool the data from several ratio objects (objects of class "rat") and compute a pooled estimate.

Usage

```
## S3 method for class 'rat'
pool(...)
```

Arguments

... Objects of class "rat".

Details

The function `pool` is generic. This is the method for the class "rat" of ratio objects. It is used to combine several estimates of the same quantity when each estimate is a ratio.

Each of the arguments ... must be an object of class "rat" representing a ratio object (basically a numerator and a denominator; see `rat`). We assume that these ratios are all estimates of the same quantity.

If the objects are called R_1, \dots, R_n and if R_i has numerator Y_i and denominator X_i , so that notionally $R_i = Y_i/X_i$, then the pooled estimate is the ratio-of-sums estimator

$$R = \frac{\sum_i Y_i}{\sum_i X_i}.$$

The standard error of R is computed using the delta method as described in Baddeley *et al.* (1993) or Cochran (1977, pp 154, 161).

This calculation is implemented only for certain classes of objects where the arithmetic can be performed.

This calculation is currently implemented only for objects which also belong to the class "fv" (function value tables). For example, if `Kest` is called with argument `ratio=TRUE`, the result is a suitable object (belonging to the classes "rat" and "fv").

Warnings or errors will be issued if the ratio objects ... appear to be incompatible. However, the code is not smart enough to decide whether it is sensible to pool the data.

Value

An object of the same class as the input.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Baddeley, A.J., Moyeed, R.A., Howard, C.V. and Boyde, A. (1993) Analysis of a three-dimensional point pattern with replication. *Applied Statistics* **42**, 641–668.
 Cochran, W.G. (1977) *Sampling techniques*, 3rd edition. New York: John Wiley and Sons.

See Also

[rat](#), [pool](#), [Kest](#)

Examples

```
K1 <- Kest(runifpoint(42), ratio=TRUE, correction="iso")
K2 <- Kest(runifpoint(42), ratio=TRUE, correction="iso")
K3 <- Kest(runifpoint(42), ratio=TRUE, correction="iso")
K <- pool(K1, K2, K3)
plot(K, pooliso ~ r, shade=c("hiiso", "loiso"))
```

Description

Create a three-dimensional point pattern

Usage

```
pp3(x, y, z, ...)
```

Arguments

- | | |
|----------------------|--|
| <code>x, y, z</code> | Numeric vectors of equal length, containing Cartesian coordinates of points in three-dimensional space. |
| <code>...</code> | Arguments passed to as.box3 to determine the three-dimensional box in which the points have been observed. |

Details

An object of class "pp3" represents a pattern of points in three-dimensional space. The points are assumed to have been observed by exhaustively inspecting a three-dimensional rectangular box. The boundaries of the box are included as part of the dataset.

Value

Object of class "pp3" representing a three dimensional point pattern. Also belongs to class "ppx".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[box3](#), [print.pp3](#), [ppx](#)

Examples

```
X <- pp3(runif(10), runif(10), runif(10), box3(c(0,1)))
```

ppm

Fit Point Process Model to Data

Description

Fits a point process model to an observed point pattern

Usage

```
ppm(Q, trend=~1, interaction=Poisson(),
     ...,
     covariates=NULL,
     covfunargs = list(),
     correction="border",
     rbord=reach(interaction),
     use.gam=FALSE,
     method="mpl",
     forcefit=FALSE,
     project=FALSE,
     nd = NULL,
     gcontrol=list(),
     nsim=100, nrmh=1e5, start=NULL, control=list(nrep=nrmh),
     verb=TRUE)
```

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 NULL indicating that a Poisson process (stationary or nonstationary) should be fitted. |
| ... | Ignored. |

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 or single numbers. 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", "translate" and "none", depending on the interaction.
r bord	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 <code>glm</code> instead of <code>gam</code> .
method	The method used to fit the model. Options are "mpl" for the method of Maximum PseudoLikelihood, and "ho" for the Huang-Ogata approximate maximum likelihood method.
forcefit	Logical flag for internal use. If forcefit=FALSE, some trivial models will be fitted by a shortcut. If forcefit=TRUE, the generic fitting method will always be used.
project	Logical. Setting project=TRUE will ensure that the fitted model is always a valid point process by applying <code>project.ppm</code> .
nd	Optional. Integer or pair of integers. The dimension of the grid of points (nd * nd or nd[1] * nd[2]) used to evaluate the integral in the pseudolikelihood.
gcontrol	Optional. List of parameters passed to <code>glm.control</code> (or passed to <code>gam.control</code> if use.gam=TRUE) controlling the model-fitting algorithm.
nsim	Number of simulated realisations to generate (for method="ho")
nrmh	Number of Metropolis-Hastings iterations for each simulated realisation (for method="ho")
start,control	Arguments passed to <code>rmh</code> controlling the behaviour of the Metropolis-Hastings algorithm (for method="ho")
verb	Logical flag indicating whether to print progress reports (for method="ho")

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.

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).

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: [Poisson](#), [AreaInter](#), [BadGey](#), [DiggleGatesStibbard](#), [DiggleGratton](#), [Fiksel](#), [Geyer](#), [Hardcore](#), [LennardJones](#), [MultiStrauss](#), [MultiStraussHard](#), [OrdThresh](#), [Ord](#), [Pairwise](#), [PairPiece](#), [Saturated](#), [SatPiece](#), [Softcore](#), [Strauss](#) and [StraussHard](#). See the examples below.

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 method of maximum pseudolikelihood coincides 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), or a function (which can be evaluated at any location (x, y) to obtain the value of the spatial covariate), or a window (interpreted as a logical variable which is TRUE inside the window and FALSE outside it) or a single number (indicating a covariate that is constant in this dataset). Each entry in the list must be an image (object of class "im", see [im.object](#)), or a function(x , y , ...), or a single number. The software will look up the pixel values of each image at the required locations (quadrature points). In the case of a function(x , y , ...), the arguments x and y are implicit, and any additional arguments ... should be given in `covfunargs`.

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 i th row of `covariates` should contain the values of spatial variables which have been observed at the i th 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.

A quadrature scheme is an object of class “quad” (see [quad.object](#)) which specifies both the data point pattern and the dummy points for the quadrature scheme, as well as the quadrature weights associated with these points. If Q is simply a point pattern (of class “ppp”, see [ppp.object](#)) then it is interpreted as specifying the data points only; a set of dummy points specified by [default.dummy\(\)](#) is added, and the default weighting rule is invoked to compute the quadrature weights.

Finer quadrature schemes (i.e. those with more dummy points) generally yield a better approximation, at the expense of higher computational load.

An easy way to fit models using a finer quadrature scheme is to let Q be the original point pattern data, and use the argument `nd` to determine the number of dummy points in the quadrature scheme.

Complete control over the quadrature scheme is possible. See [quadscheme](#) for an overview. Use `quadscheme(X, D, method="dirichlet")` to compute quadrature weights based on the Dirichlet tessellation, or `quadscheme(X, D, method="grid")` to compute quadrature weights by counting points in grid squares, where X and D are the patterns of data points and dummy points respectively. Alternatively use [pixelquad](#) to make a quadrature scheme with a dummy point at every pixel in a pixel image.

A practical advantage of quadrature schemes arises when we want to fit a model involving covariates (e.g. soil pH). Suppose we have only been able to observe the covariates at a small number of locations. Suppose `cov.dat` is a data frame containing the values of the covariates at the data points (i.e. `cov.dat[i,]` contains the observations for the i th data point) and `cov.dum` is another data frame (with the same columns as `cov.dat`) containing the covariate values at another set of points whose locations are given by the point pattern Y . Then setting $Q = \text{quadscheme}(X, Y)$ 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: The model may be fitted either by the method of maximum pseudolikelihood (Besag, 1975) or by the approximate maximum likelihood method of Huang and Ogata (1999). Maximum pseudolikelihood is much faster, but has poorer statistical properties.

In either case, the algorithm will begin by fitting the model by maximum pseudolikelihood. By default the algorithm returns the maximum pseudolikelihood fit.

Maximum pseudolikelihood is equivalent to maximum likelihood for Poisson point processes. Note that the method of maximum pseudolikelihood is believed to be inefficient and biased for point processes with strong interpoint interactions. In such cases, the Huang-Ogata approximate maximum likelihood method should be used, although maximum pseudolikelihood may also be used profitably for model selection in the initial phases of modelling.

Huang-Ogata method: If `method="ho"` then the model will be fitted using the Huang-Ogata (1999) approximate maximum likelihood method. First the model is fitted by maximum pseudolikelihood as described above, yielding an initial estimate of the parameter vector θ_0 . From this initial model, `nsim` simulated realisations are generated. The score and Fisher information of the model at $\theta = \theta_0$ are estimated from the simulated realisations. Then one step of the Fisher scoring algorithm is taken, yielding an updated estimate θ_1 . The corresponding model is returned.

Simulated realisations are generated using [rmh](#). The iterative behaviour of the Metropolis-Hastings algorithm is controlled by the arguments `start` and `control` which are passed to [rmh](#).

As a shortcut, the argument `nrmh` determines the number of Metropolis-Hastings iterations run to produce one simulated realisation (if `control` is absent). Also if `start` is absent or equal to `NULL`, it defaults to `list(n.start=N)` where N is the number of points in the data point pattern.

Edge correction Edge correction should be applied to the sufficient statistics of the model, to reduce bias. The argument `correction` is the name of an edge correction method. The default `correction="border"` specifies the border correction, in which the quadrature window (the domain of integration of the pseudolikelihood) is obtained by trimming off a margin of width `rbord` from the observation window of the data pattern. Not all edge corrections are implemented (or implementable) for arbitrary windows. Other options depend on the argument `interaction`, but these generally include `correction="periodic"` (the periodic or toroidal edge correction in which opposite edges of a rectangular window are identified) and `correction="translate"` (the translation correction, see Baddeley 1998 and Baddeley and Turner 2000). For pairwise interaction models there is also Ripley's isotropic correction, identified by `correction="isotropic"` or "Ripley".

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. One useful technique is maximum profile pseudolikelihood, which is implemented in the command [profilepl](#).

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 `project=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 `project=TRUE`. See also the functions `valid.ppm` and `project.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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Baddeley, A. and Turner, R. Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42** (2000) 283–322.
- Berman, M. and Turner, T.R. Approximating point process likelihoods with GLIM. *Applied Statistics* **41** (1992) 31–38.
- 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.

See Also

[ppm.object](#) for details of how to print, plot and manipulate a fitted model.

[ppp](#) and [quadscheme](#) for constructing data.

Interactions: [Poisson](#), [AreaInter](#), [BadGey](#), [DiggleGatesStibbard](#), [DiggleGratton](#), [Geyer](#), [Fiksel](#), [Hardcore](#), [LennardJones](#), [MultiStrauss](#), [MultiStraussHard](#), [OrdThresh](#), [Ord](#), [Pairwise](#), [PairPiece](#), [Saturated](#), [SatPiece](#), [Softcore](#), [Strauss](#) and [StraussHard](#).

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.

Examples

```

data(nztrees)
ppm(nztrees)
# fit the stationary Poisson process
# to point pattern 'nztrees'

## Not run:
Q <- quadscheme(nztrees)
ppm(Q)
# equivalent.

## End(Not run)

ppm(nztrees, nd=128)

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)

## Not run:
library(splines)
ppm(nztrees, ~ bs(x,df=3))

## End(Not run)
#       WARNING: do not use predict.ppm() on this result
# 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.

# Huang-Ogata fit:
## Not run: ppm(nztrees, ~1, Strauss(r=10), method="ho")



# COVARIATES
#
X <- rpoispp(42)
weirdfunction <- function(x,y){ 10 * x^2 + 5 * sin(10 * y) }
#
# (a) covariate values as function
ppm(X, ~ y + Z, covariates=list(Z=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, covariates=data.frame(Z=Zvalues))
# Note Q not X

# COVARIATE FUNCTION WITH EXTRA ARGUMENTS
#
f <- function(x,y,a){ y - a }
ppm(X, ~x + f, covariates=list(f=f), covfunargs=list(a=1/2))

## MULTITYPE POINT PROCESSES ###
data(lansing)
# Multitype point pattern --- trees marked by species


# fit stationary marked Poisson process
# with different intensity for each species
## Not run: ppm(lansing, ~ marks, Poisson())


# fit nonstationary marked Poisson process
# with different log-cubic trend for each species
## Not run: ppm(lansing, ~ marks * polynom(x,y,3), Poisson())

```

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 <i>pseudo</i> 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
bermantest	bermantest.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
kstest	kstest.ppm	Kolmogorov-Smirnov test
quadrat.test	quadrat.test.ppm	Quadrat counting test

<code>reach</code>	<code>reach.ppm</code>	Interaction range of model
<code>rmhmodel</code>	<code>rmhmodel.ppm</code>	Model in a form that can be simulated
<code>rmh</code>	<code>rmh.ppm</code>	Perform simulation
<code>unitname</code>	<code>unitname.ppm</code>	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:

<code>coef</code>	the fitted regular parameters (as returned by <code>glm</code>)
<code>trend</code>	the trend formula or <code>NULL</code>
<code>interaction</code>	the point process interaction family (an object of class "interact") or <code>NULL</code>
<code>Q</code>	the quadrature scheme used
<code>maxlogpl</code>	the maximised value of log pseudolikelihood
<code>correction</code>	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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`ppm`, `coef.ppm`, `fitted.ppm`, `print.ppm`, `predict.ppm`, `plot.ppm`.

Examples

```

data(cells)
fit <- ppm(cells, ~ x, Strauss(0.1), correction="periodic")
fit
coef(fit)
## Not run:
pred <- predict(fit)

## End(Not run)
pred <- predict(fit, ngrid=20, type="trend")
## Not run:
plot(fit)

## End(Not run)

```

ppp*Create a Point Pattern*

Description

Creates an object of class "ppp" representing a point pattern dataset in the two-dimensional plane.

Usage

```
ppp(x, y, ..., window, marks, check=TRUE)
```

Arguments

x	Vector of x coordinates of data points
y	Vector of y coordinates of data points
window	window of observation, an object of class "owin"
...	arguments passed to owin to create the window, if window is missing
marks	(optional) mark values for the points. A vector or data frame.
check	Logical flag indicating whether to check that all the (x, y) points lie inside the specified window. Do not set this to FALSE unless you are sure that this check is unnecessary.

Details

In the **spatstat** library, a point pattern dataset is described by an object of class "ppp". This function creates such objects.

The vectors x and y must be numeric vectors of equal length. They are interpreted as the cartesian coordinates of the points in the pattern.

A point pattern dataset is assumed to have been observed within a specific region of the plane called the observation window. An object of class "ppp" representing a point pattern contains information specifying the observation window. This window must always be specified when creating a point pattern dataset; there is intentionally no default action of "guessing" the window dimensions from the data points alone.

You can specify the observation window in several (mutually exclusive) ways:

- xrange, yrangle specify a rectangle with these dimensions;
- poly specifies a polygonal boundary. If the boundary is a single polygon then poly must be a list with components x, y giving the coordinates of the vertices. If the boundary consists of several disjoint polygons then poly must be a list of such lists so that poly[[i]]\$x gives the x coordinates of the vertices of the i th boundary polygon.
- mask specifies a binary pixel image with entries that are TRUE if the corresponding pixel is inside the window.
- window is an object of class "owin" (see [owin.object](#)) specifying the window.

The arguments xrange, yrangle or poly or mask are passed to the window creator function [owin](#) for interpretation. See [owin](#) for further details.

The argument window, if given, must be an object of class "owin". It is a full description of the window geometry, and could have been obtained from [owin](#) or [as.owin](#), or by just extracting the

observation window of another point pattern, or by manipulating such windows. See [owin](#) or the Examples below.

The points with coordinates *x* and *y* **must** lie inside the specified window, in order to define a valid object of this class. Any points which do not lie inside the window will be removed from the point pattern, and a warning will be issued. See the section on Rejected Points.

The name of the unit of length for the *x* and *y* coordinates can be specified in the dataset, using the argument *unitname*, which is passed to [owin](#). See the examples below, or the help file for [owin](#).

The optional argument *marks* is given if the point pattern is marked, i.e. if each data point carries additional information. For example, points which are classified into two or more different types, or colours, may be regarded as having a mark which identifies which colour they are. Data recording the locations and heights of trees in a forest can be regarded as a marked point pattern where the mark is the tree height.

The argument *marks* can be either

- a vector, of the same length as *x* and *y*, which is interpreted so that *marks*[*i*] is the mark attached to the point (*x*[*i*],*y*[*i*]). If the mark is a real number then *marks* should be a numeric vector, while if the mark takes only a finite number of possible values (e.g. colours or types) then *marks* should be a factor.
- a data frame, with the number of rows equal to the number of points in the point pattern. The *i*th row of the data frame is interpreted as containing the mark values for the *i*th point in the point pattern. The columns of the data frame correspond to different mark variables (e.g. tree species and tree diameter).

See [ppp.object](#) for a description of the class "ppp".

Users would normally invoke *ppp* to create a point pattern, but the functions [as.ppp](#) and [scanpp](#) may sometimes be convenient.

Value

An object of class "ppp" describing a point pattern in the two-dimensional plane (see [ppp.object](#)).

Rejected points

The points with coordinates *x* and *y* **must** lie inside the specified window, in order to define a valid object of class "ppp". Any points which do not lie inside the window will be removed from the point pattern, and a warning will be issued.

The rejected points are still accessible: they are stored as an attribute of the point pattern called "rejects" (which is an object of class "ppp" containing the rejected points in a large window). However, rejected points in a point pattern will be ignored by all other functions except [plot.ppp](#).

To remove the rejected points altogether, use [as.ppp](#). To include the rejected points, you will need to find a larger window that contains them, and use this larger window in a call to *ppp*.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppp.object](#), [as.ppp](#), [owin.object](#), [owin](#), [as.owin](#)

Examples

```

# some arbitrary coordinates in [0,1]
x <- runif(20)
y <- runif(20)

# the following are equivalent
X <- ppp(x, y, c(0,1), c(0,1))
X <- ppp(x, y)
X <- ppp(x, y, window=owin(c(0,1),c(0,1)))

# specify that the coordinates are given in metres
X <- ppp(x, y, c(0,1), c(0,1), unitname=c("metre","metres"))

## Not run: plot(X)

# marks
m <- sample(1:2, 20, replace=TRUE)
m <- factor(m, levels=1:2)
X <- ppp(x, y, c(0,1), c(0,1), marks=m)
## Not run: plot(X)

# polygonal window
X <- ppp(x, y, poly=list(x=c(0,10,0), y=c(0,0,10)))
## Not run: plot(X)

# copy the window from another pattern
data(cells)
X <- ppp(x, y, window=cells>window)

```

ppp.object

Class of Point Patterns

Description

A class "ppp" to represent a two-dimensional point pattern. Includes information about the window in which the pattern was observed. Optionally includes marks.

Details

This class represents a two-dimensional point pattern dataset. It specifies

- the locations of the points
- the window in which the pattern was observed
- optionally, “marks” attached to each point (extra information such as a type label).

If X is an object of type ppp, it contains the following elements:

x	vector of x coordinates of data points
y	vector of y coordinates of data points
n	number of points
window	window of observation (an object of class owin)
marks	optional vector or data frame of marks

Users are strongly advised not to manipulate these entries directly.

Objects of class "ppp" may be created by the function `ppp` and converted from other types of data by the function `as.ppp`. Note that you must always specify the window of observation; there is intentionally no default action of "guessing" the window dimensions from the data points alone.

Standard point pattern datasets provided with the package include `amacrine`, `betacells`, `bramblecanes`, `cells`, `demopat`, `ganglia`, `lansing`, `longleaf`, `nztrees`, `redwood`, `simdat` and `swedishpines`. Use `data(xxx)` to access the dataset `xxx`.

Point patterns may be scanned from your own data files by `scanpp` or by using `read.table` and `as.ppp`.

They may be manipulated by the functions `[.ppp` and `superimpose`.

Point pattern objects can be plotted just by typing `plot(X)` which invokes the `plot` method for point pattern objects, `plot.ppp`. See `plot.ppp` for further information.

There are also methods for `summary` and `print` for point patterns. Use `summary(X)` to see a useful description of the data.

Patterns may be generated at random by `runifpoint`, `rpoispp`, `rMaternI`, `rMaternII`, `rSSI`, `rNeymanScott`, `rMatClust`, and `rThomas`.

Most functions which are intended to operate on a window (of class `owin`) will, if presented with a `ppp` object instead, automatically extract the window information from the point pattern.

Warnings

The internal representation of marks is likely to change in the next release of this package.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`owin`, `ppp`, `as.ppp`, `[.ppp`

Examples

```
x <- runif(100)
y <- runif(100)
X <- ppp(x, y, c(0,1),c(0,1))
X
## Not run: plot(X)
mar <- sample(1:3, 100, replace=TRUE)
mm <- ppp(x, y, c(0,1), c(0,1), marks=mar)
## Not run: plot(mm)
# points with mark equal to 2
ss <- mm[ mm$marks == 2 , ]
## Not run: plot(ss)
# left half of pattern 'mm'
lu <- owin(c(0,0.5),c(0,1))
mmleft <- mm[ , lu]
## Not run: plot(mmleft)
## Not run:
# input data from file
qq <- scanpp("my.table", unit.square())
```

```

# interactively build a point pattern
plot(unit.square())
X <- as.ppp(locator(10), unit.square())
plot(X)

## End(Not run)

```

pppdist*Distance Between Two Point Patterns***Description**

Given two point patterns, find the distance between them based on optimal point matching.

Usage

```
pppdist(X, Y, type = "spa", cutoff = 1, q = 1, matching = TRUE,
        ccode = TRUE, precision = NULL, approximation = 10,
        show.rprimal = FALSE, timelag = 0)
```

Arguments

X, Y	Two point patterns (objects of class "ppp").
type	A character string giving the type of distance to be computed. One of "spa" (default), "ace" or "mat", indicating whether the algorithm should find the optimal matching based on "subpattern assignment", "assignment only if cardinalities are equal" or "mass transfer". See details below.
cutoff	The value > 0 at which interpoint distances are cut off.
q	The order of the average that is applied to the interpoint distances. May be Inf , in which case the maximum of the interpoint distances is taken.
matching	Logical. Whether to return the optimal matching or only the associated distance.
ccode	Logical. If <code>FALSE</code> , R code is used which allows for higher precision, but is much slower.
precision	Index controlling accuracy of algorithm. The q -th powers of interpoint distances will be rounded to the nearest multiple of $10^{(-\text{precision})}$. There is a sensible default which depends on <code>ccode</code> .
approximation	If $q = \text{Inf}$, compute distance based on the optimal matching for the corresponding distance of order approximation. Can be Inf , but this makes computations extremely slow.
show.rprimal	Logical. Whether to display a plot showing the iterative solution of the restricted primal problem.
timelag	Time lag, in seconds, between successive displays of the iterative solution of the restricted primal problem.

Details

Computes the distance between point patterns X and Y based on finding the matching between them which minimizes the average of the distances between matched points (if $q=1$), the maximum distance between matched points (if $q=\text{Inf}$), and in general the q -th order average (i.e. the $1/q$ th power of the sum of the q th powers) of the distances between matched points. Distances between matched points are Euclidean distances cut off at the value of cutoff.

The parameter type controls the behaviour of the algorithm if the cardinalities of the point patterns are different. For the type "spa" (subpattern assignment) the subpattern of the point pattern with the larger cardinality n that is closest to the point pattern with the smaller cardinality m is determined; then the q -th order average is taken over n values: the m distances of matched points and $n - m$ "penalty distances" of value cutoff for the unmatched points. For the type "ace" (assignment only if cardinalities equal) the matching is empty and the distance returned is equal to cutoff if the cardinalities differ. For the type "mat" (mass transfer) each point pattern is assumed to have total mass m (= the smaller cardinality) distributed evenly among its points; the algorithm finds then the "mass transfer plan" that minimizes the q -th order weighted average of the distances, where the weights are given by the transferred mass divided by m . The result is a fractional matching (each match of two points has a weight in $(0, 1]$) with the minimized quantity as the associated distance.

The computations for all three types rely heavily on a specialized primal-dual algorithm (described in Luenberger (2003), Section 5.9) for Hitchcock's problem of optimal transport of a product from a number of suppliers to a number of (e.g. vending) locations. The C implementation used by default can handle patterns with a few hundreds of points, but should not be used with thousands of points. By setting `show.rprimal = TRUE`, some insight in the working of the algorithm can be gained.

For moderate and large values of q there can be numerical issues based on the fact that the q -th powers of distances are taken and some positive values enter the optimization algorithm as zeroes because they are too small in comparison with the larger values. In this case the number of zeroes introduced is given in a warning message, and it is possible then that the matching obtained is not optimal and the associated distance is only a strict upper bound of the true distance. As a general guideline (which can be very wrong in special situations) a small number of zeroes (up to about 50 percent of the smaller point pattern cardinality m) usually still results in the right matching, and the number can even be quite a bit higher and usually still provides a highly accurate upper bound for the distance. These numerical problems can be reduced by enforcing (much slower) R code via the argument `ccode = FALSE`.

For $q = \text{Inf}$ there is no fast algorithm available, which is why approximation is normally used: for finding the optimal matching, q is set to the value of approximation. The resulting distance is still given as the maximum rather than the q -th order average in the corresponding distance computation. If `approximation = Inf`, approximation is suppressed and a very inefficient exhaustive search for the best matching is performed.

The value of precision should normally not be supplied by the user. If `ccode = TRUE`, this value is preset to the highest exponent of 10 that the C code still can handle (usually 9). If `ccode = FALSE`, the value is preset according to q (usually 15 if q is small), which can sometimes be changed to obtain less severe warning messages.

Value

Normally an object of class `pppmatching` that contains detailed information about the parameters used and the resulting distance. See [pppmatching.object](#) for details. If `matching = FALSE`, only the numerical value of the distance is returned.

Author(s)

Dominic Schuhmacher <dominic.schuhmacher@stat.unibe.ch> <http://www.dominic.schuhmacher.name>

References

- Hitchcock F.L. (1941) The distribution of a product from several sources to numerous localities. *J. Math. Physics* **20**, 224–230.
- Luenberger D.G. (2003). *Linear and nonlinear programming*. Second edition. Kluwer.
- Schuhmacher, D. and Xia, A. (2008) A new metric between distributions of point processes. *Advances in Applied Probability* **40**, 651–672
- Schuhmacher, D., Vo, B.-T. and Vo, B.-N. (2008) A consistent metric for performance evaluation of multi-object filters. *IEEE Transactions on Signal Processing* **56**, 3447–3457.

See Also

[pppmatching.object](#) [matchingdist](#)

Examples

```
# equal cardinalities
X <- runifpoint(100)
Y <- runifpoint(100)
m <- ppdist(X, Y)
m
## Not run:
plot(m)

## End(Not run)

# differing cardinalities
X <- runifpoint(14)
Y <- runifpoint(10)
m1 <- ppdist(X, Y, type="spa")
m2 <- ppdist(X, Y, type="ace")
m3 <- ppdist(X, Y, type="mat")
summary(m1)
summary(m2)
summary(m3)
## Not run:
m1$matrix
m2$matrix
m3$matrix

## End(Not run)

# q = Inf
X <- runifpoint(10)
Y <- runifpoint(10)
mx1 <- ppdist(X, Y, q=Inf)$matrix
mx2 <- ppdist(X, Y, q=Inf, ccode=FALSE, approximation=50)$matrix
mx3 <- ppdist(X, Y, q=Inf, approximation=Inf)$matrix
((mx1 == mx2) && (mx2 == mx3))
# TRUE if approximations are good
```

pppmatching*Create a Point Matching*

Description

Creates an object of class "pppmatching" representing a matching of two planar point patterns (objects of class "ppp").

Usage

```
pppmatching(X, Y, am, type = NULL, cutoff = NULL, q = NULL,
            mdist = NULL)
```

Arguments

X, Y	Two point patterns (objects of class "ppp").
am	An X\$n by Y\$n matrix with entries ≥ 0 that specifies which points are matched and with what weight; alternatively, an object that can be coerced to this form by <code>as.matrix</code> .
type	A character string giving the type of the matching. One of "spa", "ace" or "mat", or NULL for a generic or unknown matching.
cutoff, q	Numerical values specifying the cutoff value > 0 for interpoint distances and the order $q \in [1, \infty]$ of the average that is applied to them. NULL if not applicable or unknown.
mdist	Numerical value for the distance to be associated with the matching.

Details

The argument `am` is interpreted as a "generalized adjacency matrix": if the $[i, j]$ -th entry is positive, then the i -th point of `X` and the j -th point of `Y` are matched and the value of the entry gives the corresponding weight of the match. For an unweighted matching all the weights should be set to 1. The remaining arguments are optional and allow to save additional information about the matching. See the help files for `pppdist` and `matchingdist` for details on the meaning of these parameters.

Author(s)

Dominic Schuhmacher <dominic.schuhmacher@stat.unibe.ch> <http://www.dominic.schuhmacher.name>

See Also

`pppmatching`, `object`, `matchingdist`

Examples

```
# a random unweighted complete matching
X <- runifpoint(10)
Y <- runifpoint(10)
am <- r2dtable(1, rep(1,10), rep(1,10))[[1]]
# generates a random permutation matrix
m <- ppmatching(X, Y, am)
```

```

summary(m)
m$matrix
## Not run:
plot(m)

## End(Not run)

# a random weighted complete matching
X <- runifpoint(7)
Y <- runifpoint(7)
am <- r2dtable(1, rep(10,7), rep(10,7))[[1]]/10
# generates a random doubly stochastic matrix
m2 <- ppmatching(X, Y, am)
summary(m2)
m2$matrix
## Not run:
# Note: plotting does currently not distinguish
# between different weights
plot(m2)

## End(Not run)

```

pppmatching.object *Class of Point Matchings*

Description

A class "pppmatching" to represent a matching of two planar point patterns. Optionally includes information about the construction of the matching and its associated distance between the point patterns.

Details

This class represents a (possibly weighted and incomplete) matching between two planar point patterns (objects of class "ppp").

A matching can be thought of as a bipartite weighted graph where the vertices are given by the two point patterns and edges of positive weights are drawn each time a point of the first point pattern is "matched" with a point of the second point pattern.

If *m* is an object of type ppmatching, it contains the following elements

pp1, pp2	the two point patterns to be matched (vertices)
matrix	a matrix specifying which points are matched and with what weights (edges)
type	(optional) a character string for the type of the matching (one of "spa", "ace" or "mat")
cutoff	(optional) cutoff value for interpoint distances
q	(optional) the order for taking averages of interpoint distances
distance	(optional) the distance associated with the matching

The element *matrix* is a "generalized adjacency matrix". The numbers of rows and columns match the cardinalities of the first and second point patterns, respectively. The [i, j]-th entry is positive

if the i -th point of X and the j -th point of Y are matched (zero otherwise) and its value then gives the corresponding weight of the match. For an unweighted matching all the weights are set to 1.

The optional elements are for saving details about matchings in the context of optimal point matching techniques. `type` can be one of "spa" (for "subpattern assignment"), "ace" (for "assignment only if cardinalities differ") or "mat" (for "mass transfer"). `cutoff` is a positive numerical value that specifies the maximal interpoint distance and `q` is a value in $[1, \infty]$ that gives the order of the average applied to the interpoint distances. See the help files for `pppdist` and `matchingdist` for detailed information about these elements.

Objects of class "pppmatching" may be created by the function `pppmatching`, and are most commonly obtained as output of the function `pppdist`. There are methods `plot`, `print` and `summary` for this class.

Author(s)

Dominic Schuhmacher <dominic.schuhmacher@stat.unibe.ch> <http://www.dominic.schuhmacher.name>

See Also

`matchingdist` `pppmatching`

Examples

```
# a random complete unweighted matching
X <- runifpoint(10)
Y <- runifpoint(10)
am <- r2dtable(1, rep(1,10), rep(1,10))[[1]]
# generates a random permutation matrix
m <- ppmatching(X, Y, am)
summary(m)
m$matrix
## Not run:
plot(m)

## End(Not run)

# an optimal complete unweighted matching
m2 <- pppdist(X,Y)
summary(m2)
m2$matrix
## Not run:
plot(m2)

## End(Not run)
```

Description

Creates a multidimensional space-time point pattern with any kind of coordinates and marks.

Usage

```
ppx(data, domain=NULL, coord.type=NULL)
```

Arguments

<code>data</code>	The coordinates and marks of the points. A <code>data.frame</code> or <code>hyperframe</code> .
<code>domain</code>	Optional. The space-time domain containing the points. An object in some appropriate format, or <code>NULL</code> .
<code>coord.type</code>	Character vector specifying how each column of <code>data</code> should be interpreted: as a spatial coordinate, a temporal coordinate, a local coordinate or a mark. Entries are partially matched to the values "spatial", "temporal", "local" and "mark".

Details

An object of class "`ppx`" represents a marked point pattern in multidimensional space and/or time. There may be any number of spatial coordinates, any number of temporal coordinates, any number of local coordinates, and any number of mark variables. The individual marks may be atomic (numeric values, factor values, etc) or objects of any kind.

The argument `data` should contain the coordinates and marks of the points. It should be a `data.frame` or more generally a `hyperframe` (see [hyperframe](#)) with one row of data for each point.

Each column of `data` is either a spatial coordinate, a temporal coordinate, a local coordinate, or a mark variable. The argument `coord.type` determines how each column is interpreted. It should be a character vector, of length equal to the number of columns of `data`. It should contain strings that partially match the values "spatial", "temporal", "local" and "mark". (The first letters will be sufficient.)

By default (if `coord.type` is missing or `NULL`), columns of numerical data are assumed to represent spatial coordinates, while other columns are assumed to be marks.

Value

An object of class "`ppx`".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[pp3](#), [print.ppx](#)

Examples

```
df <- data.frame(x=runif(4),y=runif(4),t=runif(4),
                  age=rep(c("old", "new"), 2),
                  size=runif(4))
X <- ppx(data=df, coord.type=c("s","s","t","m","m"))
X

val <- 20 * runif(4)
E <- lapply(val, function(s) { rpoispp(s) })
```

```
hf <- hyperframe(t=val, e=as.listof(E))
Z <- ppx(data=hf, domain=c(0,1))
Z
```

predict.kppm*Prediction from a Fitted Cluster Point Process Model***Description**

Given a fitted cluster point process model, this function computes the fitted intensity.

Usage

```
## S3 method for class 'kppm'
predict(object, ...)
```

Arguments

- | | |
|--------|--|
| object | Fitted cluster point process model. An object of class "kppm". |
| ... | Arguments passed to predict.ppm . |

Details

This is a method for the generic function [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 [predict.ppm](#).

Value

Usually a pixel image (object of class "im"), but see [predict.ppm](#) for details.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[kppm](#), [plot.kppm](#), [vcov.kppm](#), [predict.ppm](#)

Examples

```
data(redwood)
fit <- kppm(redwood, ~x, "Thomas")
predict(fit)
```

`predict.lppm`*Predict Point Process Model on Linear Network*

Description

Given a fitted point process model on a linear network, compute the fitted intensity or conditional intensity of the model.

Usage

```
## S3 method for class 'lppm'  
predict(object, ..., type = "trend", locations = NULL)
```

Arguments

object	The fitted model. An object of class "lppm", see lppm .
type	Type of values to be computed. Either "trend", "cif" or "se".
locations	Optional. Locations at which predictions should be computed. Either a data frame with two columns of coordinates, or a binary image mask.
...	Optional arguments passed to as.mask to determine the pixel resolution (if locations is missing).

Details

This function computes the fitted point process intensity, fitted conditional intensity, or standard error of the fitted intensity, for a point process model on a linear network. It is a method for the generic [predict](#) for the class "lppm".

The argument object should be an object of class "lppm" (produced by [lppm](#)) representing a point process model on a linear network.

Predicted values are computed at the locations given by the argument locations. If this argument is missing, then predicted values are computed at a fine grid of points on the linear network.

- If locations is missing or NULL (the default), the return value is a pixel image (object of class "linim" which inherits class "im") corresponding to a discretisation of the linear network, with numeric pixel values giving the predicted values at each location on the linear network.
- If locations is a data frame, the result is a numeric vector of predicted values at the locations specified by the data frame.
- If locations is a binary mask, the result is a pixel image with predicted values computed at the pixels of the mask.

Value

A pixel image (object of class "linim" which inherits class "im") or a numeric vector, depending on the argument locations. See Details.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

References

- Ang, Q.W. (2010) *Statistical methodology for events on a network*. Master's thesis, School of Mathematics and Statistics, University of Western Australia.
- Ang, Q.W., Baddeley, A. and Nair, G. (2012) Geometrically corrected second-order analysis of events on a linear network, with applications to ecology and criminology. To appear in *Scandinavian Journal of Statistics*.
- McSwiggan, G., Nair, M.G. and Baddeley, A. (2012) Fitting Poisson point process models to events on a linear network. Manuscript in preparation.

See Also

[lpp](#), [linim](#)

Examples

```
example(lpp)
fit <- lppm(X, ~x)
v <- predict(fit, type="trend")
plot(v)
```

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, ngrid=NULL, locations=NULL,
        covariates=NULL, type="trend", X=data.ppm(object),
        ..., 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 <i>y</i> and <i>x</i> directions. (Incompatible with locations)
locations	Optional. Data giving the <i>x</i> , <i>y</i> coordinates (and marks, if required) of locations at which predictions should be computed. Either a point pattern, or a data frame with columns named <i>x</i> and <i>y</i> , or a binary image mask. (Incompatible with ngrid)

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, and "se" for the standard error of the fitted spatial trend.
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.
...	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.
repair	Logical value indicating whether to repair the internal format of object, if it is found to be damaged.

Details

This function computes the spatial trend and the conditional intensity of a fitted spatial point process model, and the standard error of the estimate of spatial trend. See Baddeley and Turner (2000) for explanation and examples.

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 and conditional 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 computed:

- If `type="trend"`:** the "spatial trend" of the fitted model is evaluated at each required spatial location u .
- 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="se"`:** the estimated (asymptotic) standard error of the fitted spatial trend is evaluated at each required spatial location u . This is available only for Poisson point process models.

Note that the "spatial trend" is the same as the intensity function if the fitted model object is a Poisson point process. However, if the model is not a Poisson process, then 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 that is not easy to compute.]

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") 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.

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 i th row of `covariates` is assumed to contain covariate data for the i th 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 $x = \text{as.vector}(\text{raster.x}(locations))$ and $y = \text{as.vector}(\text{raster.y}(locations))$.

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 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: 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"`), or estimates of standard error for the fitted spatial trend (if `type="se"`).

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Baddeley, A. and Turner, R. Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42** (2000) 283–322.
 Berman, M. and Turner, T.R. Approximating point process likelihoods with GLIM. *Applied Statistics* **41** (1992) 31–38.

See Also

`ppm`, `ppm.object`, `plot.ppm`, `print.ppm`, `fitted.ppm`, `spatstat.options`

Examples

```
data(cells)

m <- ppm(cells, ~ polynom(x,y,2), Strauss(0.05))
trend <- predict(m, type="trend")
## Not run:
image(trend)
points(cells)

## End(Not run)
cif <- predict(m, type="cif")
## Not run:
persp(cif)

## End(Not run)
data(japanesepines)
mj <- ppm(japanesepines, ~harmonic(x,y,2))
se <- predict(mj, type="se")

# prediction at arbitrary locations
predict(mj, locations=data.frame(x=0.3, y=0.4))

X <- runifpoint(5, as.owin(japanesepines))
predict(mj, locations=X)
predict(mj, locations=X, type="se")

# multitype
data(amacrine)
ma <- ppm(amacrine, ~marks,
          MultiStrauss(c("off","on"),matrix(0.06, 2, 2)))
Z <- predict(ma)
```

```
Z <- predict(ma, type="cif")
predict(ma, locations=data.frame(x=0.8, y=0.5, marks="on"), type="cif")
```

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@maths.uwa.edu.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[slrm](#)

Examples

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

data(copper)
X <- copper$SouthPoints
Y <- copper$SouthLines
Z <- distmap(Y)
fitc <- slrm(X ~ Z)
pc <- predict(fitc)

Znew <- distmap(copper$Lines)[copper$SouthWindow]
pcnew <- predict(fitc, newdata=list(Z=Znew))
```

print.im

Print Brief Details of an Image

Description

Prints a very brief description of a pixel image object.

Usage

```
## S3 method for class 'im'
print(x, ...)
```

Arguments

x Pixel image (object of class "im").
... Ignored.

Details

A very brief description of the pixel image x is printed.

This is a method for the generic function [print](#).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[print](#), [im.object](#), [summary.im](#)

Examples

```
data(letterR)
U <- as.im(letterR)
U
```

print.owin

Print Brief Details of a Spatial Window

Description

Prints a very brief description of a window object.

Usage

```
## S3 method for class 'owin'
print(x, ...)
```

Arguments

x	Window (object of class "owin").
...	Ignored.

Details

A very brief description of the window x is printed.

This is a method for the generic function [print](#).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[print](#), [print.ppp](#), [summary.owin](#)

Examples

```
owin() # the unit square

data(demopat)
W <- demopat>window
W # just says it is polygonal
as.mask(W) # just says it is a binary image
```

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-fitting 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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

- [ppm.object](#) for details of the class "ppm".
- [ppm](#) for generating these objects.
- [plot.ppm](#), [predict.ppm](#)

Examples

```
## Not run:  
m <- ppm(cells, ~1, Strauss(0.05))  
m  
  
## End(Not run)
```

print.ppp*Print Brief Details of a Point Pattern Dataset*

Description

Prints a very brief description of a point pattern dataset.

Usage

```
## S3 method for class 'ppp'  
print(x, ...)
```

Arguments

x	Point pattern (object of class "ppp").
...	Ignored.

Details

A very brief description of the point pattern x is printed.

This is a method for the generic function [print](#).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[print](#), [print.owin](#), [summary.ppp](#)

Examples

```
data(cells)      # plain vanilla point pattern  
cells  
  
data(lansing)    # multitype point pattern  
lansing  
  
data(longleaf)   # numeric marks  
longleaf  
  
data(demopat)    # weird polygonal window  
demopat
```

print.psp*Print Brief Details of a Line Segment Pattern Dataset*

Description

Prints a very brief description of a line segment pattern dataset.

Usage

```
## S3 method for class 'psp'  
print(x, ...)
```

Arguments

x	Line segment pattern (object of class "psp").
...	Ignored.

Details

A very brief description of the line segment pattern x is printed.

This is a method for the generic function [print](#).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[print](#), [print.owin](#), [summary.psp](#)

Examples

```
a <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())  
a
```

print.quad*Print a Quadrature Scheme*

Description

print method for a quadrature scheme.

Usage

```
## S3 method for class 'quad'  
print(x,...)
```

Arguments

- x A quadrature scheme object, typically obtained from [quadscheme](#). An object of class "quad".
- ... Ignored.

Details

This is the `print` method for the class "quad". It prints simple information about the quadrature scheme.

See [quad.object](#) for details of the class "quad".

Value

none.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[quadscheme](#), [quad.object](#), [plot.quad](#), [summary.quad](#)

Examples

```
data(cells)
Q <- quadscheme(cells)
Q
```

profilepl

Profile Maximum Pseudolikelihood

Description

Fits point process models by profile maximum pseudolikelihood

Usage

```
profilepl(s, f, ..., rbord = NULL, verbose = TRUE)
```

Arguments

- s Data frame containing values of the irregular parameters over which the profile pseudolikelihood 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.
- rbord Radius for border correction (same for all models). If omitted, this will be computed from the interactions.
- verbose Logical flag indicating whether to print progress reports.

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. It uses the method of maximum profile pseudolikelihood.

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 \dots 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.

To apply the method of profile maximum pseudolikelihood, 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 \dots 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 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 function, the best fitting model, and other data. It can be plotted (yielding a plot of the log pseudolikelihood 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` and `print` for this class.

The components of the object include

<code>fit</code>	Best-fitting model
<code>param</code>	The data frame <code>s</code>
<code>iopf</code>	Row index of the best-fitting parameters in <code>s</code>

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

Examples

```

data(cells)

# one irregular parameter
s <- data.frame(r=seq(0.05,0.15, by=0.01))

ps <- profilepl(s, Strauss, cells)
ps
if(interactive()) plot(ps)

# two irregular parameters
s <- expand.grid(r=seq(0.05,0.15, by=0.01),sat=1:3)

pg <- profilepl(s, Geyer, cells)
pg
if(interactive()) plot(pg)
## Not run:
pg$fit

## End(Not run)

# multitype pattern with a common interaction radius
data(betacells)
s <- data.frame(R=seq(65,85,by=5))

MS <- function(R) { MultiStrauss(radii=diag(c(R,R))) }
pm <- profilepl(s, MS, betacells, ~marks)

```

progressreport *Print Progress Reports*

Description

Prints Progress Reports during a loop or iterative calculation.

Usage

```
progressreport(i, n, every = max(1, ceiling(n/100)), nperline =
min(charsperline, every * ceiling(charsperline/(every + 3))),
charsperline = 60, style=spatstat.options("progress"))
```

Arguments

i	Integer. The current iteration number (from 1 to n).
n	Integer. The (maximum) number of iterations to be computed.
every	Optional integer. The number of iterations between successive reports.
nperline	Optional integer. The maximum number of reports to be printed per line of output.
charsperline	Optional integer. The number of characters in a line of output.
style	Character string determining the style of display. See Details.

Details

This is a convenient function for reporting progress during an iterative sequence of calculations or a suite of simulations.

If `style="txtbar"` then `txtProgressBar` is used to represent progress as a bar made of text characters in the R interpreter window.

If `style="tty"`, then progress reports are printed using `cat`. This only seems to work well under Linux.

Value

Null.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

Examples

```
for(i in 1:40) progressreport(i, 40)
```

project.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)
```

Arguments

<code>object</code>	Fitted point process model (object of class "ppm").
<code>...</code>	Ignored.
<code>fatal</code>	Logical value indicating whether to generate an error if the model cannot be projected to a valid model.
<code>trace</code>	Logical value indicating whether to print a trace of the decision process.

Details

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 [project.ppm](#) modifies the model object so that the model is valid.

[project.ppm](#) 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 [project.ppm](#) is the true maximum pseudolikelihood fit to the data.

For large datasets or complex models, the algorithm used in [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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppm](#), [valid.ppm](#), [spatstat.options](#)

Examples

```
fit <- ppm(redwood, ~1, Strauss(0.1))
coef(fit)
fit2 <- project.ppm(fit)
coef(fit2)
```

Description

Given a point pattern and a line segment pattern, this function moves each point to the closest location on a line segment.

Usage

```
project2segment(X, Y)
```

Arguments

- | | |
|---|---|
| X | A point pattern (object of class "ppp"). |
| Y | A line segment pattern (object of class "psp"). |

Details

For each point x in the point pattern X , this function finds the closest line segment y in the line segment pattern Y . It then ‘projects’ the point x onto the line segment y by finding the position z along y which is closest to x . This position z is returned, along with supplementary information.

Value

A list with the following components. Each component has length equal to the number of points in X , and its entries correspond to the points of X .

- | | |
|-------|--|
| Xproj | Point pattern (object of class "ppp" containing the projected points. |
| mapXY | Integer vector identifying the nearest segment to each point. |
| d | Numeric vector of distances from each point of X to the corresponding projected point. |
| tp | Numeric vector giving the scaled parametric coordinate $0 \leq t_p \leq 1$ of the position of the projected point along the segment. |

For example suppose $\text{mapXY}[2] = 5$ and $\text{tp}[2] = 0.33$. Then $Y[5]$ is the line segment lying closest to $X[2]$. The projection of the point $X[2]$ onto the segment $Y[5]$ is the point $Xproj[2]$, which lies one-third of the way between the first and second endpoints of the line segment $Y[5]$.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[nearestsegment](#) for a faster way to determine which segment is closest to each point.

Examples

```
X <- rstrat(square(1), 5)
Y <- as.psp(matrix(runif(20), 5, 4), window=owin())
plot(Y, lwd=3, col="green")
plot(X, add=TRUE, col="red", pch=16)
v <- project2segment(X,Y)
Xproj <- v$Xproj
plot(Xproj, add=TRUE, pch=16)
arrows(X$x, X$y, Xproj$x, Xproj$y, angle=10, length=0.15, col="red")
```

psp*Create a Line Segment Pattern*

Description

Creates an object of class "psp" representing a line segment pattern in the two-dimensional plane.

Usage

```
psp(x0,y0, x1, y1, window, marks=NULL,
     check=spatstat.options("checksegments"))
```

Arguments

x0	Vector of x coordinates of first endpoint of each segment
y0	Vector of y coordinates of first endpoint of each segment
x1	Vector of x coordinates of second endpoint of each segment
y1	Vector of y coordinates of second endpoint of each segment
window	window of observation, an object of class "owin"
marks	(optional) vector or data frame of mark values
check	Logical value indicating whether to check that the line segments lie inside the window.

Details

In the **spatstat** library, a spatial pattern of line segments is described by an object of class "psp". This function creates such objects.

The vectors $x0$, $y0$, $x1$ and $y1$ must be numeric vectors of equal length. They are interpreted as the cartesian coordinates of the endpoints of the line segments.

A line segment pattern is assumed to have been observed within a specific region of the plane called the observation window. An object of class "psp" representing a point pattern contains information specifying the observation window. This window must always be specified when creating a point pattern dataset; there is intentionally no default action of "guessing" the window dimensions from the data points alone.

The argument `window` must be an object of class "owin". It is a full description of the window geometry, and could have been obtained from `owin` or `as.owin`, or by just extracting the observation window of another dataset, or by manipulating such windows. See `owin` or the Examples below.

The optional argument `marks` is given if the line segment pattern is marked, i.e. if each line segment carries additional information. For example, line segments which are classified into two or more different types, or colours, may be regarded as having a mark which identifies which colour they are.

The object `marks` must be a vector of the same length as `x0`, or a data frame with number of rows equal to the length of `x0`. The interpretation is that `marks[i]` or `marks[i,]` is the mark attached to the i th line segment. If the marks are real numbers then `marks` should be a numeric vector, while if the marks takes only a finite number of possible values (e.g. colours or types) then `marks` should be a factor.

See `psp.object` for a description of the class "psp".

Users would normally invoke `psp` to create a line segment pattern, and the function `as.psp` to convert data in another format into a line segment pattern.

Value

An object of class "psp" describing a line segment pattern in the two-dimensional plane (see [psp.object](#)).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[psp.object](#), [as.psp](#), [owin.object](#), [owin](#), [as.owin](#), [marks.psp](#)

Examples

```
X <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
m <- data.frame(A=1:10, B=letters[1:10])
X <- psp(runif(10), runif(10), runif(10), runif(10), window=owin(), marks=m)
```

psp.object

Class of Line Segment Patterns

Description

A class "psp" to represent a spatial pattern of line segments in the plane. Includes information about the window in which the pattern was observed. Optionally includes marks.

Details

An object of this class represents a two-dimensional pattern of line segments. It specifies

- the locations of the line segments (both endpoints)
- the window in which the pattern was observed
- optionally, a “mark” attached to each line segment (extra information such as a type label).

If X is an object of type psp, it contains the following elements:

ends	data frame with entries x0, y0, x1, y1 giving coordinates of segment endpoints
window	window of observation (an object of class owin)
n	number of line segments
marks	optional vector or data frame of marks
markformat	character string specifying the format of the marks; “none”, “vector”, or “dataframe”

Users are strongly advised not to manipulate these entries directly.

Objects of class "psp" may be created by the function [psp](#) and converted from other types of data by the function [as.psp](#). Note that you must always specify the window of observation; there is intentionally no default action of “guessing” the window dimensions from the line segments alone.

Subsets of a line segment pattern may be obtained by the functions [\[.psp](#) and [clip.psp](#).

Line segment pattern objects can be plotted just by typing `plot(X)` which invokes the `plot` method for line segment pattern objects, `plot.psp`. See `plot.psp` for further information.

There are also methods for `summary` and `print` for line segment patterns. Use `summary(X)` to see a useful description of the data.

Utilities for line segment patterns include `midpoints.psp` (to compute the midpoints of each segment), `lengths.psp`, (to compute the length of each segment), `angles.psp`, (to compute the angle of orientation of each segment), and `distmap.psp` to compute the distance map of a line segment pattern.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`psp`, `as.psp`, `[.psp`

Examples

```
# creating
  a <- psp(runif(20),runif(20),runif(20),runif(20), window=owin())
# converting from other formats
  a <- as.psp(matrix(runif(80), ncol=4), window=owin())
  a <- as.psp(data.frame(x0=runif(20), y0=runif(20),
                         x1=runif(20), y1=runif(20)), window=owin())
# clipping
  w <- owin(c(0.1,0.7), c(0.2, 0.8))
  b <- clip.psp(a, w)
  b <- a[w]
# the last two lines are equivalent.
```

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, ...,
      trend = ~1, interaction = Poisson(), rbord = reach(interaction),
      truecoef=NULL, hi.res=NULL, funargs = list(correction="best"))
```

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.
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	Optional. List of additional arguments to be passed to fun.

Details

Let x be a point pattern dataset consisting of points x_1, \dots, 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

$$V(r) = \sum_i \Delta S(x_i, x, r) - \int_W \Delta S(u, x, r) \lambda(u, x) du$$

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> Ege Rubak and Jesper Moller.

References

Baddeley, A., Rubak, E. and Moller, 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

```
data(cells)
fit0 <- ppm(cells, ~1) # uniform Poisson

G0 <- psst(fit0, Gest)
G0
if(interactive()) 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, ...,
      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

<code>object</code>	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").
<code>r</code>	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.
<code>breaks</code>	Optional alternative to <code>r</code> for advanced use.
<code>...</code>	Ignored.
<code>trend, interaction, rbord</code>	Optional. Arguments passed to <code>ppm</code> to fit a point process model to the data, if <code>object</code> is a point pattern. See <code>ppm</code> for details.
<code>ppmcorrection</code>	Optional. Character string specifying the edge correction for the pseudolikelihood to be used in fitting the point process model. Passed to <code>ppm</code> .
<code>correction</code>	Optional. Character string specifying which diagnostic quantities will be computed. Options are "all" and "best". The default is to compute all diagnostic quantities.
<code>truecoef</code>	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 <code>hi.res</code> .
<code>hi.res</code>	Optional. List of parameters passed to <code>quadscheme</code> . 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.
<code>nr</code>	Optional. Number of <code>r</code> values to be used if <code>r</code> is not specified.
<code>ngrid</code>	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, \dots, 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

$$V(r) = \sum_i A(x_i, x, r) - \int_W A(u, x, r) \lambda(u, x) du$$

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> Ege Rubak and Jesper Moller.

References

Baddeley, A., Rubak, E. and Moller, 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

```
pso <- spatstat.options(psstA.ngrid=16,psstA.nr=10)
X <- rStrauss(200,0.1,0.05)
plot(psstA(X))
plot(psstA(X, interaction=Strauss(0.05)))
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, ...,
      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.
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

$$V(r) = \sum_i \Delta S(x_i, x, r) - \int_W \Delta S(u, x, r) \lambda(u, x) du$$

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> Ege Rubak and Jesper Moller.

References

Baddeley, A., Rubak, E. and Moller, 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

```
X <- rStrauss(200,0.1,0.05)
plot(psstG(X))
plot(psstG(X, interaction=Strauss(0.05)))
```

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)
```

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.

<code>control</code>	List of parameters controlling the Metropolis-Hastings algorithm <code>rmh</code> which generates each simulated realisation from the model (unless the model is Poisson). This list becomes the argument <code>control</code> of <code>rmh.default</code> . It overrides <code>nrep</code> .
<code>saveall</code>	Logical flag indicating whether to save all the intermediate calculations.
<code>monochrome</code>	Logical flag indicating whether the plot should be in black and white (<code>monochrome=TRUE</code>), or in colour (<code>monochrome=FALSE</code>).
<code>limcol</code>	String. The colour to be used when plotting the 95-percent limit curves.
<code>maxerr</code>	Maximum number of failures tolerated while generating simulated realisations. See Details.
<code>check</code>	Logical value indicating whether to check the internal format of <code>fit</code> . 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 <code>check=TRUE</code> .
<code>repair</code>	Logical value indicating whether to repair the internal format of <code>fit</code> , if it is found to be damaged.

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.

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

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

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

See Also

[diagnose.ppm](#), [lurking](#), [residuals.ppm](#), [eem](#), [ppm.object](#), [ppm](#), [rmh](#), [rmhcontrol](#)

Examples

```
data(cells)

fit <- ppm(cells, ~1, Poisson())
diagnose.ppm(fit) # no suggestion of departure from stationarity
## Not run: qqplot.ppm(fit, 80) # strong evidence of non-Poisson interaction

## Not run:
diagnose.ppm(fit, type="pearson")
qqplot.ppm(fit, type="pearson")

## End(Not run)
```

```

#####
## oops, I need the plot coordinates
mypreciousdata <- .Last.value
## Not run: mypreciousdata <- qqplot.ppm(fit, type="pearson")

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)
## Not run: 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())
## Not run: 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))
## Not run: qqplot.ppm(fit, 100, expr)

```

Description

A class "quad" to represent a quadrature scheme.

Details

A (finite) quadrature scheme is a list of quadrature points u_j and associated weights w_j which is used to approximate an integral by a finite sum:

$$\int f(x)dx \approx \sum_j f(u_j)w_j$$

Given a point pattern dataset, a *Berman-Turner* quadrature scheme is one which includes all these data points, as well as a nonzero number of other ("dummy") points.

These quadrature schemes are used to approximate the pseudolikelihood of a point process, in the method of Baddeley and Turner (2000) (see Berman and Turner (1992)). Accuracy and computation time both increase with the number of points in the quadrature scheme.

An object of class "quad" represents a Berman-Turner quadrature scheme. It can be passed as an argument to the model-fitting function `ppm`, which requires a quadrature scheme.

An object of this class contains at least the following elements:

- `data`: an object of class "ppp"
giving the locations (and marks) of the data points.
- `dummy`: an object of class "ppp"
giving the locations (and marks) of the dummy points.
- `w`: vector of nonnegative weights for the quadrature points

Users are strongly advised not to manipulate these entries directly.

The domain of quadrature is specified by `dummy$window` while the observation window (if this needs to be specified separately) is taken to be `data$window`.

The weights vector `w` may also have an attribute `attr(w, "zeroes")` equivalent to the logical vector (`w == 0`). If this is absent then all points are known to have positive weights.

To create an object of class "quad", users would typically call the high level function `quadscheme`. (They are actually created by the low level function `quad`.)

Entries are extracted from a "quad" object by the functions `x.quad`, `y.quad`, `w.quad` and `marks.quad`, which extract the `x` coordinates, `y` coordinates, weights, and marks, respectively. The function `n.quad` returns the total number of quadrature points (dummy plus data).

An object of class "quad" can be converted into an ordinary point pattern by the function `union.quad` which simply takes the union of the data and dummy points.

Quadrature schemes can be plotted using `plot.quad` (a method for the generic `plot`).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`quadscheme`, `ppm`

`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)`

Arguments

object	fitted point process model (an object of class "ppm" or "kppm").
drop	Logical value determining whether to delete quadrature 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 `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.

It may happen that some quadrature 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 quadrature 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". 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".

Value

A quadrature scheme (object of class "quad").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppm.object](#), [quad.object](#), [ppm](#)

Examples

```
data(cells)
fit <- ppm(cells, ~1, Strauss(r=0.1))
Q <- quad.ppm(fit)
## Not run: plot(Q)
Q$data$n
Q$dummy$n
```

quadrat.test*Chi-Squared Dispersion Test for Spatial Point Pattern Based on Quadrat Counts*

Description

Performs a chi-squared test of Complete Spatial Randomness for a given point pattern, based on quadrat counts. Alternatively performs a chi-squared goodness-of-fit test of a fitted inhomogeneous Poisson model.

Usage

```
quadrat.test(X, ...)
## S3 method for class 'ppp'
quadrat.test(X, nx=5, ny=nx, ..., xbreaks=NULL, ybreaks=NULL, tess=NULL)
## S3 method for class 'ppm'
quadrat.test(X, nx=5, ny=nx, ..., xbreaks=NULL, ybreaks=NULL, tess=NULL)
## S3 method for class 'quadratcount'
quadrat.test(X, ...)
```

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") to be tested. Alternatively X can be the result of applying quadratcount to a point pattern.
nx, ny	Numbers of quadrats in the <i>x</i> and <i>y</i> directions. Incompatible with xbreaks and ybreaks.
...	Ignored.
xbreaks	Optional. Numeric vector giving the <i>x</i> coordinates of the boundaries of the quadrats. Incompatible with nx.
ybreaks	Optional. Numeric vector giving the <i>y</i> coordinates of the boundaries of the quadrats. Incompatible with ny.
tess	Tessellation (object of class "tess") determining the quadrats. Incompatible with nx, ny, xbreaks, ybreaks.

Details

These functions perform χ^2 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") 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 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 we perform the χ^2 test of goodness-of-fit to the quadrat counts.

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 "quadrat.test", and there is a plot method for this class. See the examples.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[quadrat.test.splitppp](#), [quadratcount](#), [quadrats](#), [quadratresample](#), [chisq.test](#), [kstest](#).

To test a Poisson point process model against a specific alternative, use [anova.ppm](#).

Examples

```

data(simdat)
quadrat.test(simdat)
quadrat.test(simdat, 4, 3)

# quadrat counts
qS <- quadratcount(simdat, 4, 3)
quadrat.test(qS)

# fitted model: inhomogeneous Poisson
fitx <- ppm(simdat, ~x, Poisson())
quadrat.test(fitx)

te <- quadrat.test(simdat, 4)
residuals(te) # Pearson residuals

plot(te)

plot(simdat, pch="+", cols="green", lwd=2)
plot(te, add=TRUE, col="red", cex=1.4, lty=2, lwd=3)

```

```

sublab <- eval(substitute(expression(p[chi^2]==z),
                           list(z=signif(te$p.value,3))))
title(sub=sublab, cex.sub=3)

# quadrats of irregular shape
B <- dirichlet(runifpoint(6, simdat$window))
qB <- quadrat.test(simdat, tess=B)
plot(simdat, main="quadrat.test(simdat, tess=B)", pch="+")
plot(qB, add=TRUE, col="red", lwd=2, cex=1.2)

```

quadrat.test.splitppp Chi-Squared Test of CSR for Split Point Pattern

Description

Performs a chi-squared test of Complete Spatial Randomness for each of the component patterns in a split point pattern.

Usage

```
## S3 method for class 'splitppp'
quadrat.test(X, ...)
```

Arguments

X	A split point pattern (object of class "splitppp"), each component of which will be subjected to the goodness-of-fit test.
...	Arguments passed to quadrat.test.ppp .

Details

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

If X is a split point pattern, then for each of the component point patterns (taken separately) we test the null hypotheses of Complete Spatial Randomness. The method `quadrat.test.ppp` is applied to each component point pattern.

The return value is a list of objects, each giving the result of one of the tests.

Value

A list of objects, each giving the result of one of the hypothesis tests. Each component object is of class "htest" and "quadrat.test". The list itself is of class "listof" so that it can be printed and plotted.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[quadrat.test](#), [quadratcount](#), [quadrats](#), [quadratresample](#), [chisq.test](#), [kstest](#).

To test a Poisson point process model against a specific alternative, use [anova.ppm](#).

Examples

```
data(humberside)
qH <- quadrat.test(split(humberside), 2, 3)
plot(qH)
qH
```

quadratcount

Quadrat counting for a point pattern

Description

Divides window into quadrats and counts the numbers of points in each quadrat.

Usage

```
quadratcount(X, ...)
## S3 method for class 'ppp'
quadratcount(X, nx=5, ny=nx, ...,
             xbreaks=NULL, ybreaks=NULL, tess=NULL)
## S3 method for class 'splitppp'
quadratcount(X, ...)
```

Arguments

X	A point pattern (object of class "ppp") or a split point pattern (object of class "splitppp").
nx, ny	Numbers of rectangular quadrats in the <i>x</i> and <i>y</i> directions. Incompatible with xbreaks and ybreaks.
...	Additional arguments passed to quadratcount.ppp.
xbreaks	Numeric vector giving the <i>x</i> coordinates of the boundaries of the rectangular quadrats. Incompatible with nx.
ybreaks	Numeric vector giving the <i>y</i> coordinates of the boundaries of the rectangular quadrats. Incompatible with ny.
tess	Tessellation (object of class "tess") determining the quadrats. Incompatible with nx, ny, xbreaks, ybreaks.

Details

Quadrat counting is an elementary technique for analysing spatial point patterns. See Diggle (2003).

If X is a point pattern, then by default, the window containing the point pattern X is divided into an $nx \times ny$ grid of rectangular tiles or ‘quadrats’. (If the window is not a rectangle, then these tiles are intersected with the window.) The number of points of X falling in each quadrat is counted. These numbers are returned as a contingency table.

If *xbreaks* is given, it should be a numeric vector giving the *x* coordinates of the quadrat boundaries. If it is not given, it defaults to a sequence of *nx*+1 values equally spaced over the range of *x* coordinates in the window *X\$window*.

Similarly if *ybreaks* is given, it should be a numeric vector giving the *y* coordinates of the quadrat boundaries. It defaults to a vector of *ny*+1 values equally spaced over the range of *y* coordinates in the window. The lengths of *xbreaks* and *ybreaks* may be different.

Alternatively, quadrats of any shape may be used. The argument *tess* can be a tessellation (object of class "tess") whose tiles will serve as the quadrats.

The algorithm counts the number of points of *X* falling in each quadrat, and returns these counts as a contingency table.

The return value is a table which can be printed neatly. The return value is also a member of the special class "quadratcount". Plotting the object will display the quadrats, annotated by their counts. See the examples.

If *X* is a split point pattern (object of class "splitppp" then quadrat counting will be performed on each of the components point patterns, and the resulting contingency tables will be returned in a list. This list can be printed or plotted.

Marks attached to the points are ignored by *quadratcount.ppp*. To obtain a separate contingency table for each type of point in a multitype point pattern, first separate the different points using [split.ppp](#), then apply *quadratcount.splitppp*. See the Examples.

Value

The value of *quadratcount.ppp* is a contingency table containing the number of points in each quadrat. The table is also an object of the special class "quadratcount" and there is a plot method for this class.

The value of *quadratcount.splitppp* is a list of such contingency tables, each containing the quadrat counts for one of the component point patterns in *X*. This list also has the class "listof" which has print and plot methods.

Note

To perform a chi-squared test based on the quadrat counts, use [quadrat.test](#).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Diggle, P.J. *Statistical analysis of spatial point patterns*. Academic Press, 2003.
- Stoyan, D. and Stoyan, H. (1994) *Fractals, random shapes and point fields: methods of geometrical statistics*. John Wiley and Sons.

See Also

[quadrat.test](#), [quadrats](#), [quadratresample](#), [miplot](#)

Examples

```
X <- runifpoint(50)
quadratcount(X)
quadratcount(X, 4, 5)
quadratcount(X, xbreaks=c(0, 0.3, 1), ybreaks=c(0, 0.4, 0.8, 1))
qX <- quadratcount(X, 4, 5)

# plotting:
plot(X, pch="+")
plot(qX, add=TRUE, col="red", cex=1.5, lty=2)

# irregular window
data(humberside)
plot(humberside)
qH <- quadratcount(humberside, 2, 3)
plot(qH, add=TRUE, col="blue", cex=1.5, lwd=2)

# multitype - split
plot(quadratcount(split(humberside), 2, 3))

# quadrats determined by tessellation:
B <- dirichlet(runifpoint(6))
qX <- quadratcount(X, tess=B)
plot(X, pch="+")
plot(qX, add=TRUE, col="red", cex=1.5, lty=2)
```

quadratresample

Resample a Point Pattern by Resampling Quadrats

Description

Given a point pattern dataset, create a resampled point pattern by dividing the window into rectangular quadrats and randomly resampling the list of quadrats.

Usage

```
quadratresample(X, nx, ny=nx, ...,
                 replace = FALSE, nsamples = 1,
                 verbose = (nsamples > 1))
```

Arguments

<code>X</code>	A point pattern dataset (object of class "ppp").
<code>nx, ny</code>	Numbers of quadrats in the x and y directions.
<code>...</code>	Ignored.
<code>replace</code>	Logical value. Specifies whether quadrats should be sampled with or without replacement.
<code>nsamples</code>	Number of randomised point patterns to be generated.
<code>verbose</code>	Logical value indicating whether to print progress reports.

Details

This command implements a very simple bootstrap resampling procedure for spatial point patterns X .

The dataset X must be a point pattern (object of class "ppp") and its observation window must be a rectangle.

The window is first divided into $N = nx * ny$ rectangular tiles (quadrats) of equal size and shape. To generate one resampled point pattern, a random sample of N quadrats is selected from the list of N quadrats, with replacement (if `replace=TRUE`) or without replacement (if `replace=FALSE`). The i th quadrat in the original dataset is then replaced by the i th sampled quadrat, after the latter is shifted so that it occupies the correct spatial position. The quadrats are then reconstituted into a point pattern inside the same window as X .

If `replace=FALSE`, this procedure effectively involves a random permutation of the quadrats. The resulting resampled point pattern has the same number of points as X . If `replace=TRUE`, the number of points in the resampled point pattern is random.

Value

A point pattern (if `nsamples = 1`) or a list of point patterns (if `nsamples > 1`).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[quadrats](#), [quadratcount](#).

See [varblock](#) to estimate the variance of a summary statistic by block resampling.

Examples

```
data(besi)
quadratresample(besi, 6, 3)
```

quadrats

Divide Region into Quadrats

Description

Divides window into rectangular quadrats and returns the quadrats as a tessellation.

Usage

```
quadrats(X, nx = 5, ny = nx, xbreaks = NULL, ybreaks = NULL, keepempty=FALSE)
```

Arguments

X	A window (object of class "owin") or anything that can be coerced to a window using <code>as.owin</code> , such as a point pattern.
nx, ny	Numbers of quadrats in the x and y directions. Incompatible with <code>xbreaks</code> and <code>ybreaks</code> .
<code>xbreaks</code>	Numeric vector giving the x coordinates of the boundaries of the quadrats. Incompatible with <code>nx</code> .
<code>ybreaks</code>	Numeric vector giving the y coordinates of the boundaries of the quadrats. Incompatible with <code>ny</code> .
<code>keepempty</code>	Logical value indicating whether to delete or retain empty quadrats. See Details.

Details

If the window X is a rectangle, it is divided into an $nx * ny$ grid of rectangular tiles or 'quadrats'.

If X is not a rectangle, then the bounding rectangle of X is first divided into an $nx * ny$ grid of rectangular tiles, and these tiles are then intersected with the window X.

The resulting tiles are returned as a tessellation (object of class "tess") which can be plotted and used in other analyses.

If `xbreaks` is given, it should be a numeric vector giving the x coordinates of the quadrat boundaries. If it is not given, it defaults to a sequence of $nx+1$ values equally spaced over the range of x coordinates in the window X>window.

Similarly if `ybreaks` is given, it should be a numeric vector giving the y coordinates of the quadrat boundaries. It defaults to a vector of $ny+1$ values equally spaced over the range of y coordinates in the window. The lengths of `xbreaks` and `ybreaks` may be different.

By default (if `keepempty=FALSE`), any rectangular tile which does not intersect the window X is ignored, and only the non-empty intersections are treated as quadrats, so the tessellation may consist of fewer than $nx * ny$ tiles. If `keepempty=TRUE`, empty intersections are retained, and the tessellation always contains exactly $nx * ny$ tiles, some of which may be empty.

Value

A tessellation (object of class "tess") as described under `tess`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`tess`, `quadratcount`, `quadrat.test`, `quadratresample`

Examples

```

W <- square(10)
Z <- quadrats(W, 4, 5)
plot(Z)

data(letterR)
plot(quadrats(letterR, 5, 7))

```

quadscheme

Generate a Quadrature Scheme from a Point Pattern

Description

Generates a quadrature scheme (an object of class "quad") from point patterns of data and dummy points.

Usage

```
quadscheme(data, dummy, method="grid", ...)
```

Arguments

<code>data</code>	The observed data point pattern. An object of class "ppp" or in a format recognised by as.ppp()
<code>dummy</code>	The pattern of dummy points for the quadrature. An object of class "ppp" or in a format recognised by as.ppp() . Defaults to <code>default.dummy(data, ...)</code>
<code>method</code>	The name of the method for calculating quadrature weights: either "grid" or "dirichlet".
<code>...</code>	Parameters of the weighting method (see below) and parameters for constructing the dummy points if necessary.

Details

This is the primary method for producing a quadrature schemes for use by [ppm](#).

The function [ppm](#) fits a point process model to an observed point pattern using the Berman-Turner quadrature approximation (Berman and Turner, 1992; Baddeley and Turner, 2000) to the pseudo-likelihood of the model. It requires a quadrature scheme consisting of the original data point pattern, an additional pattern of dummy points, and a vector of quadrature weights for all these points. Such quadrature schemes are represented by objects of class "quad". See [quad.object](#) for a description of this class.

Quadrature schemes are created by the function `quadscheme`. The arguments `data` and `dummy` specify the data and dummy points, respectively. There is a sensible default for the dummy points (provided by [default.dummy](#)). Alternatively the dummy points may be specified arbitrarily and given in any format recognised by [as.ppp](#). There are also functions for creating dummy patterns including [corners](#), [gridcentres](#), [stratrand](#) and [spokes](#).

The quadrature region is the region over which we are integrating, and approximating integrals by finite sums. If `dummy` is a point pattern object (class "ppp") then the quadrature region is taken to be `dummy$window`. If `dummy` is just a list of x, y coordinates then the quadrature region defaults to the observation window of the data pattern, `data$window`.

If `dummy` is missing, then the optional arguments (for ...) include an argument `nd`. An `nd[1]` by `nd[2]` grid of dummy points is generated by [default.dummy](#).

If `method = "grid"` then the optional arguments (for ...) are (`nd`, `ntile`). The quadrature region (see below) is divided into an `ntile[1]` by `ntile[2]` grid of rectangular tiles. The weight for each quadrature point is the area of a tile divided by the number of quadrature points in that tile.

If `method="dirichlet"` then the optional arguments are (`exact=TRUE`, `nd`). The quadrature points (both data and dummy) are used to construct the Dirichlet tessellation. The quadrature weight

of each point is the area of its Dirichlet tile inside the quadrature region. If `exact == TRUE` then this area is computed exactly using the package `deldir`; otherwise it is computed approximately by discretisation.

Value

An object of class "quad" describing the quadrature scheme (data points, dummy points, and quadrature weights) suitable as the argument `Q` of the function `ppm()` for fitting a point process model.

The quadrature scheme can be inspected using the `print` and `plot` methods for objects of class "quad".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Baddeley, A. and Turner, R. Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42** (2000) 283–322.
 Berman, M. and Turner, T.R. Approximating point process likelihoods with GLIM. *Applied Statistics* **41** (1992) 31–38.

See Also

`ppm`, `as.ppp`, `quad.object`, `gridweights`, `dirichlet.weights`, `corners`, `gridcentres`, `stratrand`, `spokes`

Examples

```
data(simdat)

# grid weights
Q <- quadscheme(simdat)
Q <- quadscheme(simdat, method="grid")
Q <- quadscheme(simdat, nd=50)           # 1 dummy point per tile
Q <- quadscheme(simdat, ntile=25, nd=50) # 4 dummy points per tile

# Dirichlet weights
Q <- quadscheme(simdat, method="dirichlet", exact=FALSE)

# random dummy pattern
## Not run:
D <- runifpoint(250, simdat$window)
Q <- quadscheme(simdat, D, method="dirichlet", exact=FALSE)

## End(Not run)

# polygonal window
data(demopat)
X <- unmark(demopat)
Q <- quadscheme(X)

# mask window
```

```
X>window <- as.mask(X>window)
Q <- quadscheme(X)
```

quantile.im*Sample Quantiles of Pixel Image***Description**

Compute the sample quantiles of the pixel values of a given pixel image.

Usage

```
## S3 method for class 'im'
quantile(x, ...)
```

Arguments

- x A pixel image. An object of class "im".
- ... Optional arguments passed to [quantile.default](#). They determine the probabilities for which quantiles should be computed. See [quantile.default](#).

Details

This simple function applies the generic [quantile](#) operation to the pixel values of the image x.

This function is a convenient way to inspect an image and to obtain summary statistics. See the examples.

Value

A vector of quantiles.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[quantile](#), [cut.im](#), [im.object](#)

Examples

```
# artificial image data
Z <- setcov(square(1))

# find the quartiles
quantile(Z)

# find the deciles
quantile(Z, probs=(0:10)/10)
```

raster.x*Cartesian Coordinates for a Pixel Raster*

Description

Return the *x* and *y* coordinates of each pixel in a binary pixel image.

Usage

```
raster.x(w)
raster.y(w)
raster.xy(w)
```

Arguments

w A window (an object of class "owin") of type "mask" representing a binary pixel image.

Details

The argument *w* should be a window (an object of class "owin", see [owin.object](#) for details). A window of type "mask" represents a binary pixel image.

The functions `raster.x` and `raster.y` return a matrix of the same dimensions as the binary pixel image itself, with entries giving the *x* coordinate (for `raster.x`) or *y* coordinate (for `raster.y`) of each pixel in the image.

The function `raster.xy` returns a list with components *x* and *y* which are numeric vectors of equal length containing the pixel coordinates.

Value

`raster.x` and `raster.y` return a matrix of the same dimensions as the pixel grid in *w*, and giving the value of the *x* (or *y*) coordinate of each pixel in the raster.

`raster.xy` returns a list with components *x* and *y* which are numeric vectors of equal length containing the pixel coordinates.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[owin](#), [as.mask](#)

Examples

```
u <- owin(c(-1,1),c(-1,1)) # square of side 2
w <- as.mask(u, eps=0.01) # 200 x 200 grid
X <- raster.x(w)
Y <- raster.y(w)
disc <- owin(c(-1,1), c(-1,1), mask=(X^2 + Y^2 <= 1))
## Not run: plot(disc)
# approximation to the unit disc
```

rat	<i>Ratio object</i>
-----	---------------------

Description

Stores the numerator, denominator, and value of a ratio as a single object.

Usage

```
rat(ratio, numerator, denominator, check = TRUE)
```

Arguments

ratio, numerator, denominator

Three objects belonging to the same class.

check

Logical. Whether to check that the objects are [compatible](#).

Details

The class "rat" is a simple mechanism for keeping track of the numerator and denominator when calculating a ratio. Its main purpose is simply to signal that the object is a ratio.

The function `rat` creates an object of class "rat" given the numerator, the denominator and the ratio. No calculation is performed; the three objects are simply stored together.

The arguments `ratio`, `numerator`, `denominator` can be objects of any kind. They should belong to the same class. It is assumed that the relationship

$$\text{ratio} = \frac{\text{numerator}}{\text{denominator}}$$

holds in some version of arithmetic. However, no calculation is performed.

By default the algorithm checks whether the three arguments `ratio`, `numerator`, `denominator` are compatible objects, according to [compatible](#).

The result is equivalent to `ratio` except for the addition of extra information.

Value

An object equivalent to the object `ratio` except that it also belongs to the class "rat" and has additional attributes `numerator` and `denominator`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[compatible](#), [pool](#)

rCauchy*Simulate Neyman-Scott Point Process with Cauchy cluster kernel*

Description

Generate a random point pattern, a simulated realisation of the Neyman-Scott process with Cauchy cluster kernel.

Usage

```
rCauchy(kappa, omega, mu, win = owin(), eps = 0.001)
```

Arguments

kappa	Intensity of the Poisson process of cluster centres. A single positive number, a function, or a pixel image.
omega	Scale parameter for cluster kernel. Determines the size of clusters. A positive number, in the same units as the spatial coordinates.
mu	Mean number of points per cluster (a single positive number) or reference intensity for the cluster points (a function or a pixel image).
win	Window in which to simulate the pattern. An object of class "owin" or something acceptable to as.owin .
eps	Threshold below which the values of the cluster kernel will be treated as zero for simulation purposes.

Details

This algorithm generates a realisation of the Neyman-Scott process with Cauchy cluster kernel, inside the window `win`.

The process is constructed by first generating a Poisson point process of “parent” points with intensity `kappa`. Then each parent point is replaced by a random cluster of points, the number of points in each cluster being random with a Poisson (`mu`) distribution, and the points being placed independently and uniformly according to a Cauchy kernel.

In this implementation, parent points are not restricted to lie in the window; the parent process is effectively the uniform Poisson process on the infinite plane.

This model can be fitted to data by the method of minimum contrast, using [cauchy.estK](#), [cauchy.estpcf](#) or [kppm](#).

The algorithm can also generate spatially inhomogeneous versions of the cluster process:

- The parent points can be spatially inhomogeneous. If the argument `kappa` is a function(`x,y`) or a pixel image (object of class "im"), then it is taken as specifying the intensity function of an inhomogeneous Poisson process that generates the parent points.
- The offspring points can be inhomogeneous. If the argument `mu` is a function(`x,y`) or a pixel image (object of class "im"), then it is interpreted as the reference density for offspring points, in the sense of Waagepetersen (2006).

When the parents are homogeneous (`kappa` is a single number) and the offspring are inhomogeneous (`mu` is a function or pixel image), the model can be fitted to data using [kppm](#), or using [cauchy.estK](#) or [cauchy.estpcf](#) applied to the inhomogeneous K function.

Value

The simulated point pattern (an object of class "ppp").

Additionally, some intermediate results of the simulation are returned as attributes of this point pattern. See [rNeymanScott](#).

Author(s)

Abdollah Jalilian and Rasmus Waagepetersen. Adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@csiro.au>
<http://www.maths.uwa.edu.au/~adrian/>

References

Jalilian, A., Guan, Y. and Waagepetersen, R. (2011) Decomposition of variance for spatial Cox processes. Manuscript submitted for publication.

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

See Also

[rpoispp](#), [rNeymanScott](#), [cauchy.estK](#), [cauchy.estpcf](#), [kppm](#).

Examples

```
# homogeneous
X <- rCauchy(30, 0.01, 5)
# inhomogeneous
Z <- as.im(function(x,y){ exp(2 - 3 * x) }, W= owin())
Y <- rCauchy(50, 0.01, Z)
```

rcell

Simulate Baddeley-Silverman Cell Process

Description

Generates a random point pattern, a simulated realisation of the Baddeley-Silverman cell process model.

Usage

```
rcell(win=square(1), nx, ny=nx, dx=NULL, dy=NULL)
```

Arguments

<code>win</code>	A window. An object of class owin , or data in any format acceptable to as.owin() .
<code>dx</code>	Width of the cells. Incompatible with <code>nx</code> .
<code>dy</code>	Height of the cells. Incompatible with <code>ny</code> .
<code>nx</code>	Number of columns of cells in the window. Incompatible with <code>dx</code> .
<code>ny</code>	Number of rows of cells in the window. Incompatible with <code>dy</code> .

Details

This function generates a simulated realisation of the “cell process” (Baddeley and Silverman, 1984), a random point process with the same second-order properties as the uniform Poisson process. In particular, the K function of this process is identical to the K function of the uniform Poisson process (aka Complete Spatial Randomness). The same holds for the pair correlation function and all other second-order properties. The cell process is a counterexample to the claim that the K function completely characterises a point pattern.

A cell process is generated by dividing space into equal rectangular tiles. In each tile, a random number N of points is placed, where N takes the values 0, 1 and 10 with probabilities 1/10, 8/9 and 1/90 respectively. The points within a tile are independent and uniformly distributed in that tile, and the numbers of points in different tiles are independent random integers.

In the function `rcell` the tile dimensions are determined by the quantities `dx`, `dy` if they are present. If they are absent, then the grid spacing is determined so that there will be `nx` columns and `ny` rows of tiles in the bounding rectangle of `win`. The cell process is then generated in these tiles.

Some of the resulting random points may lie outside the window `win`: if they do, they are deleted. The result is a point pattern inside the window `win`.

Value

A point pattern (object of class “`ppp`”).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

Baddeley, A.J. and Silverman, B.W. (1984) A cautionary example on the use of second-order methods for analyzing point patterns. *Biometrics* **40**, 1089-1094.

See Also

`rstrat`, `rsyst`, `runifpoint`, `Kest`

Examples

```
X <- rcell(nx=15)
plot(X)
plot(Kest(X))
```

Description

Generate a random pattern of points, a simulated realisation of the Diggle-Gates-Stibbard process, using a perfect simulation algorithm.

Usage

```
rDGS(beta, rho, W = owin())
```

Arguments

<code>beta</code>	intensity parameter (a positive number).
<code>rho</code>	interaction range (a non-negative number).
<code>W</code>	window (object of class "owin") in which to generate the random pattern. Currently this must be a rectangular window.

Details

This function generates a realisation of the Diggle-Gates-Stibbard point process in the window `W` using a ‘perfect simulation’ algorithm.

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 simulation algorithm used to generate the point pattern is ‘dominated coupling from the past’ as implemented by Berthelsen and Moller (2002, 2003). This is a ‘perfect simulation’ or ‘exact simulation’ algorithm, so called because the output of the algorithm is guaranteed to have the correct probability distribution exactly (unlike the Metropolis-Hastings algorithm used in `rmh`, whose output is only approximately correct).

There is a tiny chance that the algorithm will run out of space before it has terminated. If this occurs, an error message will be generated.

Value

A point pattern (object of class “`ppp`”).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> based on original code for the Strauss process by Kasper Klitgaard Berthelsen.

References

- Berthelsen, K.K. and Moller, J. (2002) A primer on perfect simulation for spatial point processes. *Bulletin of the Brazilian Mathematical Society* 33, 351–367.
- Berthelsen, K.K. and Moller, J. (2003) Likelihood and non-parametric Bayesian MCMC inference for spatial point processes based on perfect simulation and path sampling. *Scandinavian Journal of Statistics* 30, 549–564.
- 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.
- Moller, J. and Waagepetersen, R. (2003). *Statistical Inference and Simulation for Spatial Point Processes*. Chapman and Hall/CRC.

See Also

[rmh](#), [DiggleGatesStibbard](#), [rStrauss](#), [rHardcore](#), [rDiggleGratton](#).

Examples

```
X <- rDGS(50, 0.05)
```

rDiggleGratton

Perfect Simulation of the Diggle-Gratton Process

Description

Generate a random pattern of points, a simulated realisation of the Diggle-Gratton process, using a perfect simulation algorithm.

Usage

```
rDiggleGratton(beta, delta, rho, kappa=1, W = owin())
```

Arguments

beta	intensity parameter (a positive number).
delta	hard core distance (a non-negative number).
rho	interaction range (a number greater than delta).
kappa	interaction exponent (a non-negative number).
W	window (object of class "owin") in which to generate the random pattern. Currently this must be a rectangular window.

Details

This function generates a realisation of the Diggle-Gratton point process in the window W using a ‘perfect simulation’ algorithm.

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

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

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) use β , since in [spatstat](#) we reserve the symbol β for an intensity parameter.

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

The simulation algorithm used to generate the point pattern is ‘dominated coupling from the past’ as implemented by Berthelsen and Moller (2002, 2003). This is a ‘perfect simulation’ or ‘exact simulation’ algorithm, so called because the output of the algorithm is guaranteed to have the correct probability distribution exactly (unlike the Metropolis-Hastings algorithm used in [rmh](#), whose output is only approximately correct).

There is a tiny chance that the algorithm will run out of space before it has terminated. If this occurs, an error message will be generated.

Value

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

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> based on original code for the Strauss process by Kasper Klitgaard Berthelsen.

References

- Berthelsen, K.K. and Moller, J. (2002) A primer on perfect simulation for spatial point processes. *Bulletin of the Brazilian Mathematical Society* 33, 351-367.
- Berthelsen, K.K. and Moller, J. (2003) Likelihood and non-parametric Bayesian MCMC inference for spatial point processes based on perfect simulation and path sampling. *Scandinavian Journal of Statistics* 30, 549-564.
- 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.
- Moller, J. and Waagepetersen, R. (2003). *Statistical Inference and Simulation for Spatial Point Processes*. Chapman and Hall/CRC.

See Also

`rmh`, `DiggleGratton`, `rStrauss`, `rHardcore`.

Examples

```
X <- rDiggleGratton(50, 0.02, 0.07)
```

reach

Interaction Distance of a Point Process

Description

Computes the interaction distance of a point process.

Usage

```
reach(x, ...)
## S3 method for class 'ppm'
reach(x, ..., epsilon=0)
## S3 method for class 'interact'
reach(x, ...)
## S3 method for class 'rmhmodel'
reach(x, ...)
```

Arguments

- x Either a fitted point process model (object of class "ppm"), an interpoint interaction (object of class "interact") 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 ‘interaction distance’ or ‘interaction range’ of a point process model is the smallest 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](#)) 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(x)` 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 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.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppm](#), [Poisson](#), [Strauss](#), [StraussHard](#), [MultiStrauss](#), [MultiStraussHard](#), [Softcore](#), [DiggleGratton](#), [Pairwise](#), [PairPiece](#), [Geyer](#), [LennardJones](#), [Saturated](#), [OrdThresh](#), [Ord](#), [rmhmodel](#)

Examples

```
reach(Poisson())
# returns 0

reach(Strauss(r=7))
# returns 7
```

```

data(swedishpines)
fit <- ppm(swedishpines, ~1, Strauss(r=7))
reach(fit)
# returns 7

reach(OrdThresh(42))
# returns Inf

reach(MultiStrauss(1:2, matrix(c(1,3,3,1),2,2)))
# returns 3

```

reduced.sample*Reduced Sample Estimator using Histogram Data***Description**

Compute the Reduced Sample estimator of a survival time distribution function, from histogram data

Usage

```
reduced.sample(nco, cen, ncc, show=FALSE, uppercen=0)
```

Arguments

nco	vector of counts giving the histogram of uncensored observations (those survival times that are less than or equal to the censoring time)
cen	vector of counts giving the histogram of censoring times
ncc	vector of counts giving the histogram of censoring times for the uncensored observations only
uppercen	number of censoring times greater than the rightmost histogram breakpoint (if there are any)
show	Logical value controlling the amount of detail returned by the function value (see below)

Details

This function is needed mainly for internal use in **spatstat**, but may be useful in other applications where you want to form the reduced sample estimator from a huge dataset.

Suppose T_i are the survival times of individuals $i = 1, \dots, M$ with unknown distribution function $F(t)$ which we wish to estimate. Suppose these times are right-censored by random censoring times C_i . Thus the observations consist of right-censored survival times $\tilde{T}_i = \min(T_i, C_i)$ and non-censoring indicators $D_i = 1\{T_i \leq C_i\}$ for each i .

If the number of observations M is large, it is efficient to use histograms. Form the histogram `cen` of all censoring times C_i . That is, `obs[k]` counts the number of values C_i in the interval `(breaks[k], breaks[k+1])` for $k > 1$ and `[breaks[1], breaks[2]]` for $k = 1$. Also form the histogram `nco` of all uncensored times, i.e. those \tilde{T}_i such that $D_i = 1$, and the histogram of all censoring times for which the survival time is uncensored, i.e. those C_i such that $D_i = 1$. These three histograms are the arguments passed to `kaplan.meier`.

The return value `rs` is the reduced-sample estimator of the distribution function $F(t)$. Specifically, `rs[k]` is the reduced sample estimate of $F(\text{breaks}[k+1])$. The value is exact, i.e. the use of histograms does not introduce any approximation error.

Note that, for the results to be valid, either the histogram breaks must span the censoring times, or the number of censoring times that do not fall in a histogram cell must have been counted in `uppercen`.

Value

If `show = FALSE`, a numeric vector giving the values of the reduced sample estimator. If `show=TRUE`, a list with three components which are vectors of equal length,

<code>rs</code>	Reduced sample estimate of the survival time c.d.f. $F(t)$
<code>numerator</code>	numerator of the reduced sample estimator
<code>denominator</code>	denominator of the reduced sample estimator

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[kaplan.meier](#), [km.rs](#)

`redwood`

California Redwoods Point Pattern (Ripley's Subset)

Description

Locations of 62 seedlings and saplings of California redwood trees.

The data represent the locations of 62 seedlings and saplings of California redwood trees in a square sampling region. They originate from Strauss (1975); the present data are a subset extracted by Ripley (1977) in a subregion that has been rescaled to a unit square.

There are many further analyses of this dataset. It is often used as a canonical example of a clustered point pattern (see e.g. Diggle, 1983).

The original, full redwood dataset is supplied in the `spatstat` library as `redwoodfull`.

Usage

`data(redwood)`

Format

An object of class "ppp" representing the point pattern of tree locations. The window has been rescaled to the unit square.

See [ppp.object](#) for details of the format of a point pattern object.

Source

Strauss (1975), subset extracted by Ripley (1977)

References

- Diggle, P.J. (1983) *Statistical analysis of spatial point patterns*. Academic Press.
- Ripley, B.D. (1977) Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, Series B* **39**, 172–212.
- Strauss, D.J. (1975) A model for clustering. *Biometrika* **63**, 467–475.

See Also

[redwoodfull](#)

`redwoodfull`

California Redwoods Point Pattern (Entire Dataset)

Description

These data represent the locations of 195 seedlings and saplings of California redwood trees in a square sampling region. They were described and analysed by Strauss (1975). This is the “full” dataset; most writers have analysed a subset extracted by Ripley (1977) which is available as [redwood](#).

Strauss (1975) divided the sampling region into two subregions I and II demarcated by a diagonal line across the region. The spatial pattern appears to be slightly regular in region I and strongly clustered in region II.

The dataset `redwoodfull` contains the full point pattern of 195 trees. The auxiliary information about the subregions is contained in `redwoodfull.extra`, which is a list with entries

<code>diag</code>	The coordinates of the diagonal boundary between regions I and II
<code>regionI</code>	Region I as a window object
<code>regionII</code>	Region II as a window object
<code>regionR</code>	Ripley’s subrectangle (approximate)
<code>plot</code>	Function to plot the full data and auxiliary markings

Ripley (1977) extracted a subset of these data, containing 62 points, lying within a square subregion which overlaps regions I and II. He rescaled the data to the unit square. This has been re-analysed many times, and is the dataset usually known as “the redwood data” in the spatial statistics literature. The exact dataset used by Ripley is supplied in the `spatstat` library as [redwood](#). There are some minor inconsistencies with `redwood` since it originates from a different digitisation.

The approximate position of the square chosen by Ripley within the `redwoodfull` pattern is indicated by the window `redwoodfull.extra$regionR`.

Usage

```
data(redwoodfull)
```

Format

The dataset `redwoodfull` is an object of class “ppp” representing the point pattern of tree locations. The window has been rescaled to the unit square. See [ppp.object](#) for details of the format of a point pattern object.

The dataset `redwoodfull.extra` is a list with entries

<code>diag</code>	coordinates of endpoints of a line, in format <code>list(x=numeric(2),y=numeric(2))</code>
<code>regionI</code>	a window object
<code>regionII</code>	a window object
<code>regionR</code>	a window object
<code>plot</code>	Function with no arguments

Source

Strauss (1975). The plot of the data published by Strauss (1975) was scanned and digitised by Sandra Pereira, University of Western Australia, 2002.

References

- Diggle, P.J. (1983) *Statistical analysis of spatial point patterns*. Academic Press.
 Ripley, B.D. (1977) Modelling spatial patterns (with discussion). *Journal of the Royal Statistical Society, Series B* **39**, 172–212.
 Strauss, D.J. (1975) A model for clustering. *Biometrika* **63**, 467–475.

See Also

[redwood](#)

Examples

```
data(redwoodfull)
plot(redwoodfull)
redwoodfull.extra$plot()
# extract the pattern in region II
redwoodII <- redwoodfull[, redwoodfull.extra$regionII]
```

`reflect`

Reflect In Origin

Description

Reflects a geometrical object through the origin.

Usage

```
reflect(X)

## S3 method for class 'im'
reflect(X)

## Default S3 method:
reflect(X)
```

Arguments

- `X` Any suitable dataset representing a two-dimensional object, such as a point pattern (object of class "ppp"), or a window (object of class "owin").

Details

The object `X` is reflected through the origin. That is, each point in `X` with coordinates (x, y) is mapped to the position $(-x, -y)$.

This is equivalent to applying the affine transformation with matrix `diag(c(-1, -1))`. It is also equivalent to rotation about the origin by 180 degrees.

The command `reflect` is generic, with a method for pixel images and a default method.

Value

Another object of the same type, representing the result of reflection.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[affine](#), [flipxy](#)

Examples

```
plot(reflect(as.im(letterR)))
plot(reflect(letterR), add=TRUE)
```

Description

Given a multitype point pattern, this function estimates the spatially-varying probability of each type of point, using kernel smoothing. The default smoothing bandwidth is selected by cross-validation.

Usage

```
relrisk(X, sigma = NULL, ..., varcov = NULL, at = "pixels", casecontrol=TRUE)
```

Arguments

- | | |
|---------------------|---|
| <code>X</code> | A multitype point pattern (object of class "ppp" which has factor valued marks). |
| <code>sigma</code> | Optional. Standard deviation of isotropic Gaussian smoothing kernel. |
| <code>...</code> | Arguments passed to <code>bw.relrisk</code> to select the bandwidth, or passed to <code>density.ppp</code> to control the pixel resolution. |
| <code>varcov</code> | Optional. Variance-covariance matrix of anisotropic Gaussian smoothing kernel.
Incompatible with <code>sigma</code> . |

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").
casecontrol	Logical. Whether to treat a bivariate point pattern as consisting of cases and controls. See Details.

Details

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 this command computes the spatially-varying risk of an event, i.e. the probability $p(u)$ that a point at spatial location u will be a case.

If X is a multitype point pattern with $m > 2$ types, or if X is a bivariate point pattern and `casecontrol=FALSE`, then this command computes, for each type j , a nonparametric estimate of the spatially-varying risk 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 `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 $p_j(u)$ for $j = 1, \dots, m$.

If `at = "points"` the calculation is performed only at the data points x_i . 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.

Estimation is performed by a simple Nadaraya-Watson type kernel smoother (Diggle, 2003). If `sigma` and `varcov` are both missing or null, then the smoothing bandwidth `sigma` is selected by cross-validation using `bw.relrisk`.

Value

If X consists of only two types of points, the result is a pixel image (if `at="pixels"`) or a vector of probabilities (if `at="points"`).

If X consists of more than two types of points, 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 "listof" 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 .

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

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

See Also

`bw.relrisk`, `density.ppp`, `smooth.ppp`, `eval.im`, `which.max.im`.

Examples

```

data(urkiola)
p <- relrisk(urkiola, 20)
if(interactive()) {
  plot(p, main="proportion of oak")
  plot(eval.im(p > 0.3), main="More than 30 percent oak")
  data(lansing)
  z <- relrisk(lansing)
  plot(z, main="Lansing Woods")
  plot(which.max.im(z), main="Most common species")
}

```

Replace.im

Reset Values in Subset of Image

Description

Reset the values in a subset of a pixel image.

Usage

```
## S3 replacement method for class 'im'
x[i] <- value
```

Arguments

x	A two-dimensional pixel image. An object of class "im".
i	Object defining the subregion or subset to be extracted. Either a spatial window (an object of class "owin") or a point pattern (an object of class "ppp") or something that can be converted into a point pattern by as.ppp , or any type of index that applies to a matrix.
value	Vector, matrix, factor or pixel image containing the replacement values. Short vectors will be recycled.

Details

This function changes some of the pixel values in a pixel image. The image X must be an object of class "im" representing a pixel image defined inside a rectangle in two-dimensional space (see [im.object](#)).

The subset to be changed is determined by the argument i.

If i is a spatial window (an object of class "owin"), the values of the image inside this window are changed.

If i is a point pattern (an object of class "ppp", or something that can be converted into a point pattern by [as.ppp](#)), then the values of the pixel image at the points of this pattern are changed.

If i does not satisfy any of the conditions above, then it is assumed to be a valid index for the matrix [as.matrix\(x\)](#).

Value

The image x with the values replaced.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[im.object](#), [\[.im](#), [ppp.object](#), [as.ppp](#), [owin.object](#)

Examples

```
# make up an image
X <- setcov(unit.square())
plot(X)

# a rectangular subset
W <- owin(c(0,0.5),c(0.2,0.8))
X[W] <- 2
plot(X)

# a polygonal subset
data(letterR)
R <- affine(letterR, diag(c(1,1)/2), c(-2,-0.7))
X[R] <- 3
plot(X)

# a point pattern
P <- rpoispp(20)
X[P] <- 10
plot(X)

# change pixel value at a specific location
X[list(x=0.1,y=0.2)] <- 7
```

rescale

Convert dataset to another unit of length

Description

Converts between different units of length in a spatial dataset, such as a point pattern or a window.

Usage

```
rescale(X, s)
```

Arguments

- | | |
|----------------|--|
| <code>X</code> | Any suitable dataset representing a two-dimensional object, such as a point pattern (object of class "ppp"), or a window (object of class "owin"). |
| <code>s</code> | Conversion factor: the new units are <code>s</code> times the old units. |

Details

This is generic. Methods are provided for point patterns ([rescale.ppp](#)) and windows ([rescale.owin](#)).

The spatial coordinates in the dataset X will be re-expressed in terms of a new unit of length that is s times the current unit of length given in X .

For example if X is a dataset giving coordinates in metres, then `rescale(X, 1000)` will divide the coordinate values by 1000 to obtain coordinates in kilometres, and the unit name will be changed from "metres" to "1000 metres".

Value

Another object of the same type, representing the same data, but expressed in the new units.

Note

The result of this operation is equivalent to the original dataset. If you want to actually change the coordinates by a linear transformation, producing a dataset that is not equivalent to the original one, use [affine](#).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[unitname](#), [rescale.ppp](#), [rescale.owin](#), [affine](#), [rotate](#), [shift](#)

[rescale.im](#)

Convert Pixel Image to Another Unit of Length

Description

Converts a pixel image to another unit of length.

Usage

```
## S3 method for class 'im'
rescale(X, s)
```

Arguments

X	Pixel image (object of class "im").
s	Conversion factor: the new units are s times the old units.

Details

This is a method for the generic function [rescale](#).

The spatial coordinates of the pixels in X will be re-expressed in terms of a new unit of length that is s times the current unit of length given in X . (Thus, the coordinate values are *divided* by s , while the unit value is multiplied by s).

The result is a pixel image representing the *same* data but re-expressed in a different unit.

Pixel values are unchanged.

Value

Another pixel image (of class "im"), containing the same pixel values, but with pixel coordinates expressed in the new units.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[im](#), [rescale](#), [unitname](#), [eval.im](#)

Examples

```
# Bramble Canes data: 1 unit = 9 metres
data(bramblecanes)
# distance transform
Z <- distmap(bramblecanes)
# convert to metres
# first alter the pixel values
Zm <- eval.im(9 * Z)
# now rescale the pixel coordinates
Zm <- rescale(Zm, 1/9)
```

rescale.owin

Convert Window to Another Unit of Length

Description

Converts a window to another unit of length.

Usage

```
## S3 method for class 'owin'
rescale(X, s)
```

Arguments

X Window (object of class "owin").
s Conversion factor: the new units are s times the old units.

Details

This is a method for the generic function [rescale](#).

The spatial coordinates in the window X (and its window) will be re-expressed in terms of a new unit of length that is s times the current unit of length given in X. (Thus, the coordinate values are divided by s, while the unit value is multiplied by s).

The result is a window representing the *same* region of space, but re-expressed in a different unit.

Value

Another window object (of class "owin") representing the same window, but expressed in the new units.

Note

The result of this operation is equivalent to the original window. If you want to actually change the coordinates by a linear transformation, producing a window that is larger or smaller than the original one, use [affine](#).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[unitname](#), [rescale](#), [rescale.owin](#), [affine](#), [rotate](#), [shift](#)

Examples

```
data(swedishpines)
W <- swedishpines>window
W
# coordinates are in decimetres (0.1 metre)
# convert to metres
rescale(W, 10)
```

rescale.ppp

Convert Point Pattern to Another Unit of Length

Description

Converts a point pattern dataset to another unit of length.

Usage

```
## S3 method for class 'ppp'
rescale(X, s)
```

Arguments

X	Point pattern (object of class "ppp").
s	Conversion factor: the new units are s times the old units.

Details

This is a method for the generic function [rescale](#).

The spatial coordinates in the point pattern X (and its window) will be re-expressed in terms of a new unit of length that is s times the current unit of length given in X. (Thus, the coordinate values are *divided* by s, while the unit value is multiplied by s).

The result is a point pattern representing the *same* data but re-expressed in a different unit.

Mark values are unchanged.

Value

Another point pattern (of class "ppp"), representing the same data, but expressed in the new units.

Note

The result of this operation is equivalent to the original point pattern. If you want to actually change the coordinates by a linear transformation, producing a point pattern that is not equivalent to the original one, use [affine](#).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[unitname](#), [rescale](#), [rescale.owin](#), [affine](#), [rotate](#), [shift](#)

Examples

```
# Bramble Canes data: 1 unit = 9 metres
data(bramblecanes)
# convert to metres
bram <- rescale(bramblecanes, 1/9)
```

rescale.psp

*Convert Line Segment Pattern to Another Unit of Length***Description**

Converts a line segment pattern dataset to another unit of length.

Usage

```
## S3 method for class 'psp'
rescale(X, s)
```

Arguments

X	Line segment pattern (object of class "psp").
s	Conversion factor: the new units are s times the old units.

Details

This is a method for the generic function [rescale](#).

The spatial coordinates in the line segment pattern X (and its window) will be re-expressed in terms of a new unit of length that is s times the current unit of length given in X. (Thus, the coordinate values are *divided* by s, while the unit value is multiplied by s).

The result is a line segment pattern representing the *same* data but re-expressed in a different unit.

Mark values are unchanged.

Value

Another line segment pattern (of class "psp"), representing the same data, but expressed in the new units.

Note

The result of this operation is equivalent to the original segment pattern. If you want to actually change the coordinates by a linear transformation, producing a segment pattern that is not equivalent to the original one, use **affine**.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[units](#), [affine](#), [rotate](#), [shift](#)

Examples

```
data(copper)
X <- copper$Lines
X
# data are in km
# convert to metres
rescale(X, 1/1000)
```

rescue.rectangle

Convert Window Back To Rectangle

Description

Determines whether the given window is really a rectangle aligned with the coordinate axes, and if so, converts it to a rectangle object.

Usage

```
rescue.rectangle(W)
```

Arguments

W	A window (object of class "owin").
---	------------------------------------

Details

This function decides whether the window W is actually a rectangle aligned with the coordinate axes. This will be true if W is

- a rectangle (window object of type "rectangle");
- a polygon (window object of type "polygonal" with a single polygonal boundary) that is a rectangle aligned with the coordinate axes;

- a binary mask (window object of type "mask") in which all the pixel entries are TRUE.

If so, the function returns this rectangle, a window object of type "rectangle". If not, the function returns W .

Value

Another object of class "owin" representing the same window.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[as.owin](#), [owin.object](#)

Examples

```
w <- owin(poly=list(x=c(0,1,1,0),y=c(0,0,1,1)))
rw <- rescue.rectangle(w)

w <- as.mask(unit.square())
rw <- rescue.rectangle(w)
```

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=fitted.ppm(object, check=check, drop=drop),
           coefs=NULL, 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.

<code>drop</code>	Logical value determining whether to delete quadrature points that were not used to fit the model. See quad.ppm for explanation.
<code>fittedvalues</code>	Vector of fitted values for the conditional intensity at the quadrature points, from which the residuals will be computed. For expert use only.
<code>coefs</code>	Optional. Numeric vector of coefficients for the model, replacing <code>coef(object)</code> . See the section on Modified Residuals below.
<code>quad</code>	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.

Use [plot.msr](#) to plot the residuals directly, or [diagnose.ppm](#) to produce diagnostic plots based on these residuals.

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 `coefs` and `quad`.

If `coefs` is given, then the residuals will be computed by taking the model parameters to be `coefs`. This should be a numeric vector of the same length as the vector of fitted model parameters `coef(object)`.

If `coefs` is missing and `quad` is given, then the model parameters will be determined by re-fitting 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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

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

```
data(cells)
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)
# true model parameters
truecoef <- c(log(100), log(0.7))
res.true <- residuals(fit, coefs=truecoef)
```

Description

This dataset contains the point patterns used as examples in the paper of Baddeley et al (2005). [Figure 2 is already available in **spatstat** as the **copper** dataset.]

R code is also provided to reproduce all the Figures displayed in Baddeley et al (2005). The component **plotfig** is a function, which can be called with a numeric or character argument specifying the Figure or Figures that should be plotted. See the Examples.

Usage

```
data(residualspaper)
```

Format

`residualspaper` is a list with the following components:

Fig1 The locations of Japanese pine seedlings and saplings from Figure 1 of the paper. A point pattern (object of class "ppp").

Fig3 The Chorley-Ribble data from Figure 3 of the paper. A list with three components, `lung`, `larynx` and `incin`. Each is a matrix with 2 columns giving the coordinates of the lung cancer cases, larynx cancer cases, and the incinerator, respectively. Coordinates are Eastings and Northings in km.

Fig4a The synthetic dataset in Figure 4 (a) of the paper.

Fig4b The synthetic dataset in Figure 4 (b) of the paper.

Fig4c The synthetic dataset in Figure 4 (c) of the paper.

Fig11 The covariate displayed in Figure 11. A pixel image (object of class "im") whose pixel values are distances to the nearest line segment in the `copper` data.

plotfig A function which will compute and plot any of the Figures from the paper. The argument of **plotfig** is either a numeric vector or a character vector, specifying the Figure or Figures to be plotted. See the Examples.

Source

Figure 1: Prof M. Numata. Data kindly supplied by Professor Y. Ogata with kind permission of Prof M. Tanemura.

Figure 3: Professor P.J. Diggle (rescaled by Adrian Baddeley)

Figure 4 (a,b,c): Adrian Baddeley

References

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

Examples

```
## Not run:
data(residualspaper)

X <- residualspaper$Fig4a
summary(X)
plot(X)

# reproduce all Figures
residualspaper$plotfig()

# reproduce Figures 1 to 10
residualspaper$plotfig(1:10)

# reproduce Figure 7 (a)
residualspaper$plotfig("7a")

## End(Not run)
```

rGaussPoisson

Simulate Gauss-Poisson Process

Description

Generate a random point pattern, a simulated realisation of the Gauss-Poisson Process.

Usage

```
rGaussPoisson(kappa, r, p2, win = owin(c(0,1),c(0,1)))
```

Arguments

<code>kappa</code>	Intensity of the Poisson process of cluster centres. A single positive number, a function, or a pixel image.
<code>r</code>	Diameter of each cluster that consists of exactly 2 points.
<code>p2</code>	Probability that a cluster contains exactly 2 points.
<code>win</code>	Window in which to simulate the pattern. An object of class "owin" or something acceptable to as.owin .

Details

This algorithm generates a realisation of the Gauss-Poisson point process inside the window `win`. The process is constructed by first generating a Poisson point process of parent points with intensity `kappa`. Then each parent point is either retained (with probability $1 - p2$) or replaced by a pair of points at a fixed distance `r` apart (with probability `p2`). In the case of clusters of 2 points, the line joining the two points has uniform random orientation.

In this implementation, parent points are not restricted to lie in the window; the parent process is effectively the uniform Poisson process on the infinite plane.

Value

The simulated point pattern (an object of class "ppp").

Additionally, some intermediate results of the simulation are returned as attributes of this point pattern. See [rNeymanScott](#).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[rpoispp](#), [rThomas](#), [rMatClust](#), [rNeymanScott](#)

Examples

```
pp <- rGaussPoisson(30, 0.07, 0.5)
```

rgbim

Create Colour-Valued Pixel Image

Description

Creates an object of class "im" representing a two-dimensional pixel image whose pixel values are colours.

Usage

```
rgbim(R, G, B, maxColorValue=255)
hsvim(H, S, V)
```

Arguments

R, G, B	Pixel images (objects of class "im") or constants giving the red, green, and blue components of a colour, respectively.
maxColorValue	Maximum colour value for R, G, B.
H, S, V	Pixel images (objects of class "im") or constants giving the hue, saturation, and value components of a colour, respectively.

Details

These functions take three pixel images, with real or integer pixel values, and create a single pixel image whose pixel values are colours recognisable to R.

Some of the arguments may be constant numeric values, but at least one of the arguments must be a pixel image. The image arguments should be compatible (in array dimension and in spatial position).

`rgbim` calls `rgb` to compute the colours, while `hsvim` calls `hsv`. See the help for the relevant function for more information about the meaning of the colour channels.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[im.object](#), [rgb](#), [hsv](#).

See [colourtools](#) for additional colour tools.

Examples

```
# create three images with values in [0,1]
X <- setcov(owin())
X <- eval.im(pmin(1,X))
Y <- as.im(function(x,y){(x+1)/2}, W=as.owin(X))
Z <- as.im(function(x,y){(y+1)/2}, W=as.owin(X))
RGB <- rgbeam(X, Y, Z, 1)
HSV <- hsim(X, Y, Z)
plot(RGB, valuesAreColours=TRUE)
plot(HSV, valuesAreColours=TRUE)
```

Description

Generate a random pattern of points, a simulated realisation of the Hardcore process, using a perfect simulation algorithm.

Usage

```
rHardcore(beta, R = 0, W = owin())
```

Arguments

<code>beta</code>	intensity parameter (a positive number).
<code>R</code>	hard core distance (a non-negative number).
<code>W</code>	window (object of class "owin") in which to generate the random pattern. Currently this must be a rectangular window.

Details

This function generates a realisation of the Hardcore point process in the window W using a ‘perfect simulation’ algorithm.

The Hardcore process is a model for strong spatial inhibition. Two points of the process are forbidden to lie closer than R units apart. The Hardcore process is the special case of the Strauss process (see [rStrauss](#)) with interaction parameter γ equal to zero.

The simulation algorithm used to generate the point pattern is ‘dominated coupling from the past’ as implemented by Berthelsen and Moller (2002, 2003). This is a ‘perfect simulation’ or ‘exact

simulation' algorithm, so called because the output of the algorithm is guaranteed to have the correct probability distribution exactly (unlike the Metropolis-Hastings algorithm used in `rmh`, whose output is only approximately correct).

There is a tiny chance that the algorithm will run out of space before it has terminated. If this occurs, an error message will be generated.

Value

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

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> based on original code for the Strauss process by Kasper Klitgaard Berthelsen.

References

- Berthelsen, K.K. and Moller, J. (2002) A primer on perfect simulation for spatial point processes. *Bulletin of the Brazilian Mathematical Society* 33, 351-367.
- Berthelsen, K.K. and Moller, J. (2003) Likelihood and non-parametric Bayesian MCMC inference for spatial point processes based on perfect simulation and path sampling. *Scandinavian Journal of Statistics* 30, 549-564.
- Moller, J. and Waagepetersen, R. (2003). *Statistical Inference and Simulation for Spatial Point Processes*. Chapman and Hall/CRC.

See Also

`rmh`, `Hardcore`, `rStrauss`, `rDiggleGratton`.

Examples

```
X <- rHardcore(0.05, 1.5, square(141.4))
Z <- rHardcore(100, 0.05)
```

rho2hat

Smoothed Relative Density of Pairs of Covariate Values

Description

Given a point pattern and two spatial covariates Z_1 and Z_2 , construct a smooth estimate of the relative risk of the pair (Z_1, Z_2) .

Usage

```
rho2hat(object, cov1, cov2, ..., method=c("ratio", "reweight"))
```

Arguments

object	A point pattern (object of class "ppp"), a quadrature scheme (object of class "quad") or a fitted point process model (object of class "ppm").
cov1, cov2	The two covariates. Each argument is either a function(x,y) or a pixel image (object of class "im") providing the values of the covariate at any location, or one of the strings "x" or "y" signifying the Cartesian coordinates.
...	Additional arguments passed to density.ppp to smooth the scatterplots.
method	Character string determining the smoothing method. See Details.

Details

This is a bivariate version of [rhohat](#).

If object is a point pattern, this command produces a smoothed version of the scatterplot of the values of the covariates cov1 and cov2 observed at the points of the point pattern.

The covariates cov1, cov2 must have continuous values.

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

$$\lambda(u) = \rho(Z_1(u), Z_2(u))\kappa(u)$$

where $\kappa(u)$ is the intensity of the fitted model object, and $\rho(z_1, z_2)$ is a function to be estimated. The algorithm computes a smooth estimate of the function ρ .

The method determines how the density estimates will be combined to obtain an estimate of $\rho(z_1, z_2)$:

- If method="ratio", then $\rho(z_1, z_2)$ is estimated by the ratio of two density estimates. The numerator is a (rescaled) density estimate obtained by smoothing the points $(Z_1(y_i), Z_2(y_i))$ obtained by evaluating the two covariate Z_1, Z_2 at the data points y_i . The denominator is a density estimate of the reference distribution of (Z_1, Z_2) .
- If method="reweight", then $\rho(z_1, z_2)$ is estimated by applying density estimation to the points $(Z_1(y_i), Z_2(y_i))$ obtained by evaluating the two covariate Z_1, Z_2 at the data points y_i , with weights inversely proportional to the reference density of (Z_1, Z_2) .

Value

A pixel image (object of class "im"). Also belongs to the special class "rho2hat" which has a plot method.

Author(s)

Adrian Baddeley

References

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

See Also

[rhohat](#), [methods.rho2hat](#)

Examples

```
data(besi)
attach(besi.extra)
plot(rho2hat(besi, elev, grad))
fit <- ppm(besi, ~polynom(elev, 3), covariates=besi.extra)
plot(rho2hat(fit, elev, grad))
plot(rho2hat(fit, elev, grad, method="reweight"))
```

rhohat

Smoothing Estimate of Covariate Transformation

Description

Computes a smoothing estimate of the intensity of a point process, as a function of a (continuous) spatial covariate.

Usage

```
rhohat(object, covariate, ...,
       method=c("ratio", "reweight", "transform"),
       smoother=c("kernel", "local"),
       dimyx=NULL, eps=NULL,
       n = 512, bw = "nrd0", adjust=1, from = NULL, to = NULL,
       bwref=bw,
       covname, confidence=0.95)
```

Arguments

object	A point pattern (object of class "ppp"), a quadrature scheme (object of class "quad") or a fitted point process model (object of class "ppm").
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.
method	Character string determining the smoothing method. See Details.
smoother	Character string determining the smoothing algorithm. See Details.
dimyx,eps	Arguments passed to <code>as.mask</code> to control the pixel resolution at which the covariate will be evaluated.
bw	Smoothing bandwidth or bandwidth rule (passed to <code>density.default</code>).
adjust	Smoothing bandwidth adjustment factor (passed to <code>density.default</code>).
n, from, to	Arguments passed to <code>density.default</code> 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 <code>density.default</code> or <code>locfit</code> .
covname	Optional. Character string to use as the name of the covariate.
confidence	Confidence level for confidence intervals. A number between 0 and 1.

Details

If `object` is a point pattern, this command assumes that `object` is a realisation of a Poisson 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 a function to be estimated. This command computes estimators of $\rho(z)$ proposed by Baddeley and Turner (2005) and Baddeley et al (2012).

The covariate Z must have continuous values.

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

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

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

The estimation procedure is determined by the character strings `method` and `smoother`. The estimation procedure involves computing several density estimates and combining them. The algorithm used to compute density estimates is determined by `smoother`:

- If `smoother="kernel"`, each 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 `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 .
- 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 .
- 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.

Value

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

Categorical and discrete covariates

This technique assumes that the 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 `quadratcount(X, tess=covariate)`

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

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.

See Also

`rho2hat`, `methods.rho2hat`

Examples

```
X <- rpoispp(function(x,y){exp(3+3*x)})
rho <- rho2hat(X, "x")
rho <- rho2hat(X, function(x,y){x})
plot(rho)
curve(exp(3+3*x), lty=3, col=2, add=TRUE)

rhoB <- rho2hat(X, "x", method="reweight")
rhoC <- rho2hat(X, "x", method="transform")

fit <- ppm(X, ~x)
rr <- rho2hat(fit, "y")
```

ripras

Estimate window from points alone

Description

Given an observed pattern of points, computes the Ripley-Rasson estimate of the spatial domain from which they came.

Usage

```
ripras(x, y=NULL, shape="convex", f)
```

Arguments

- | | |
|--------------------|--|
| <code>x</code> | vector of x coordinates of observed points, or a 2-column matrix giving x,y coordinates, or a list with components <code>x</code> , <code>y</code> giving coordinates (such as a point pattern object of class "ppp".) |
| <code>y</code> | (optional) vector of y coordinates of observed points, if <code>x</code> is a vector. |
| <code>shape</code> | String indicating the type of window to be estimated: either "convex" or "rectangle". |
| <code>f</code> | (optional) scaling factor. See Details. |

Details

Given an observed pattern of points with coordinates given by `x` and `y`, this function computes an estimate due to Ripley and Rasson (1977) of the spatial domain from which the points came.

The points are assumed to have been generated independently and uniformly distributed inside an unknown domain D .

If `shape="convex"` (the default), the domain D is assumed to be a convex set. The maximum likelihood estimate of D is the convex hull of the points (computed by `convexhull.xy`). Analogously to the problems of estimating the endpoint of a uniform distribution, the MLE is not optimal. Ripley and Rasson's estimator is a rescaled copy of the convex hull, centred at the centroid of the convex hull. The scaling factor is $1/\sqrt{1-m/n}$ where n is the number of data points and m the number of vertices of the convex hull. The scaling factor may be overridden using the argument `f`.

If `shape="rectangle"`, the domain D is assumed to be a rectangle with sides parallel to the coordinate axes. The maximum likelihood estimate of D is the bounding box of the points (computed by `bounding.box.xy`). The Ripley-Rasson estimator is a rescaled copy of the bounding box, with scaling factor $1/\sqrt{1-4/n}$ where n is the number of data points, centred at the centroid of the bounding box. The scaling factor may be overridden using the argument `f`.

Value

A window (an object of class "owin").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

References

Ripley, B.D. and Rasson, J.-P. (1977) Finding the edge of a Poisson forest. *Journal of Applied Probability*, **14**, 483 – 491.

See Also

`owin`, `as.owin`, `bounding.box.xy`, `convexhull.xy`

Examples

```

x <- runif(30)
y <- runif(30)
w <- ripras(x,y)
plot(owin(), main="ripas(x,y)")
plot(w, add=TRUE)
points(x,y)

X <- rpoispp(15)
plot(X, main="ripas(X)")
plot(ripas(X), add=TRUE)

# two points insufficient
ripas(c(0,1),c(0,0))
# triangle
ripas(c(0,1,0.5), c(0,0,1))
# three collinear points
ripas(c(0,0,0), c(0,1,2))

```

rjitter*Random Perturbation of a Point Pattern***Description**

Applies independent random displacements to each point in a point pattern.

Usage

```
rjitter(X, radius, retry=TRUE, giveup = 10000)
```

Arguments

X	A point pattern (object of class "ppp").
radius	Scale of perturbations. A positive numerical value. The displacement vectors will be uniformly distributed in a circle of this radius.
retry	What to do when a perturbed point lies outside the window of the original point pattern. If <code>retry=FALSE</code> , the point will be lost; if <code>retry=TRUE</code> , the algorithm will try again.
giveup	Maximum number of unsuccessful attempts.

Details

Each of the points in the point pattern X is subjected to an independent random displacement. The displacement vectors are uniformly distributed in a circle of radius `radius`.

If a displaced point lies outside the window, then if `retry=FALSE` the point will be lost.

However if `retry=TRUE`, the algorithm will try again: each time a perturbed point lies outside the window, the algorithm will reject it and generate another proposed perturbation of the original point, until one lies inside the window, or until `giveup` unsuccessful attempts have been made. In the latter case, any unresolved points will be included without any perturbation. The return value will always be a point pattern with the same number of points as X.

Value

A point pattern (object of class "ppp") in the same window as X.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

Examples

```
X <- rysyst(owin(), 10, 10)
Y <- rjitter(X, 0.02)
plot(Y)
```

rknn*Theoretical Distribution of Nearest Neighbour Distance*

Description

Density, distribution function, quantile function and random generation for the random distance to the k th nearest neighbour in a Poisson point process in d dimensions.

Usage

```
dknn(x, k = 1, d = 2, lambda = 1)
pknn(q, k = 1, d = 2, lambda = 1)
qknn(p, k = 1, d = 2, lambda = 1)
rknn(n, k = 1, d = 2, lambda = 1)
```

Arguments

x, q	vector of quantiles.
p	vector of probabilities.
n	number of observations to be generated.
k	order of neighbour.
d	dimension of space.
lambda	intensity of Poisson point process.

Details

In a Poisson point process in d -dimensional space, let the random variable R be the distance from a fixed point to the k -th nearest random point, or the distance from a random point to the k -th nearest other random point.

Then R^d has a Gamma distribution with shape parameter k and rate $\lambda * \alpha$ where α is a constant (equal to the volume of the unit ball in d -dimensional space). See e.g. Cressie (1991, page 61).

These functions support calculation and simulation for the distribution of R .

Value

A numeric vector: dknn returns the probability density, pknn returns cumulative probabilities (distribution function), qknn returns quantiles, and rknn generates random deviates.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

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

Examples

```
x <- seq(0, 5, length=20)
densities <- dknn(x, k=3, d=2)
cdfvalues <- pknn(x, k=3, d=2)
randomvalues <- rknn(100, k=3, d=2)
deciles <- qknn((1:9)/10, k=3, d=2)
```

rlabel

Random Re-Labeling of Point Pattern

Description

Randomly allocates marks to a point pattern, or permutes the existing marks, or resamples from the existing marks.

Usage

```
rlabel(X, labels=marks(X), permute=TRUE)
```

Arguments

X	Point pattern (object of class "ppp").
labels	Vector of values from which the new marks will be drawn at random. Defaults to the vector of existing marks.
permute	Logical value indicating whether to generate new marks by randomly permuting labels or by drawing a random sample with replacement.

Details

This very simple function allocates random marks to an existing point pattern X. It is useful for hypothesis testing purposes.

In the simplest case, the command `rlabel(X)` yields a point pattern obtained from X by randomly permuting the marks of the points.

If `permute=TRUE`, then `labels` should be a vector of length equal to the number of points in X. The result of `rlabel` will be a point pattern with locations given by X and marks given by a random permutation of `labels` (i.e. a random sample without replacement).

If `permute=FALSE`, then `labels` may be a vector of any length. The result of `rlabel` will be a point pattern with locations given by X and marks given by a random sample from `labels` (with replacement).

Value

A marked point pattern.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[marks<-](#) to assign arbitrary marks.

Examples

```
data(amacrine)

# Randomly permute the marks "on" and "off"
# Result always has 142 "off" and 152 "on"
Y <- rlabel(amacrine)

# randomly allocate marks "on" and "off"
# with probabilities p(off) = 0.48, p(on) = 0.52
Y <- rlabel(amacrine, permute=FALSE)

# randomly allocate marks "A" and "B" with equal probability
data(cells)
Y <- rlabel(cells, labels=factor(c("A", "B")), permute=FALSE)
```

rLGCP

Simulate Log-Gaussian Cox Process

Description

Generate a random point pattern, a realisation of the log-Gaussian Cox process.

Usage

```
rLGCP(model="exponential", mu = 0, param = NULL, ..., win)
```

Arguments

model	character string: the name of a covariance model for the Gaussian random field, as recognised by the function GaussRF in the RandomFields package.
mu	mean function of the Gaussian random field. Either a single number, a <code>function(x, y, ...)</code> or a pixel image (object of class "im").
param	Numeric vector of parameters for the covariance, as understood by the function GaussRF in the RandomFields package.
...	Further arguments passed to the function GaussRF in the RandomFields package.
win	Window in which to simulate the pattern. An object of class "owin".

Details

This function generates a realisation of a log-Gaussian Cox process (LGCP). This is a Cox point process in which the logarithm of the random intensity is a Gaussian random field with mean function μ and covariance function $c(r)$. Conditional on the random intensity, the point process is a Poisson process with this intensity.

The arguments `model` and `param` specify the covariance function of the Gaussian random field, in the format expected by the **RandomFields** package. See [GaussRF](#) or [Covariance](#) for information about this format. A list of all implemented models is available by typing `PrintModelList()`.

This algorithm uses the function `GaussRF` in the **RandomFields** package to generate values of a Gaussian random field, with the specified mean function `mu` and the covariance specified by the arguments `model` and `param`, on the points of a regular grid. The exponential of this random field is taken as the intensity of a Poisson point process, and a realisation of the Poisson process is then generated by the function `rpoispp` in the **spatstat** package.

If the simulation window `win` is missing, then it defaults to `as.owin(mu)` if `mu` is a pixel image, and it defaults to the unit square otherwise.

The LGCP model can be fitted to data using `kppm`.

Value

The simulated point pattern (an object of class "ppp").

Additionally, the simulated intensity function is returned as an attribute "Lambda".

Author(s)

Abdollah Jalilian and Rasmus Waagepetersen. Modified by Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

Moller, J., Syversveen, A. and Waagepetersen, R. (1998) Log Gaussian Cox Processes. *Scandinavian Journal of Statistics* **25**, 451–482.

See Also

`rpoispp`, `rMatClust`, `rGaussPoisson`, `rNeymanScott`, `lgcp.estK`, `kppm`

Examples

```
if(require(RandomFields) && RandomFieldsSafe()) {
  # homogeneous LGCP with exponential covariance function
  X <- rLGCP("exp", 3, c(0, variance=0.2, nugget=0, scale=.1))

  # inhomogeneous LGCP with Gaussian covariance function
  m <- as.im(function(x, y){5 - 1.5 * (x - 0.5)^2 + 2 * (y - 0.5)^2}, W=owin())
  X <- rLGCP("gauss", m, c(0, variance=0.15, nugget = 0, scale =0.5))
  plot(attr(X, "Lambda"))
  points(X)

  # inhomogeneous LGCP with Matern covariance function
  X <- rLGCP("matern", function(x, y){ 1 - 0.4 * x},
             c(0, variance=2, nugget=0, scale=0.7, a = 0.5),
             win = owin(c(0, 10), c(0, 10)))
  plot(X)
} else message("Simulation requires the RandomFields package")
```

rlinegrid*Generate grid of parallel lines with random displacement*

Description

Generates a grid of parallel lines, equally spaced, inside the specified window.

Usage

```
rlinegrid(angle = 45, spacing = 0.1, win = owin())
```

Arguments

angle	Common orientation of the lines, in degrees anticlockwise from the x axis.
spacing	Spacing between successive lines.
win	Window in which to generate the lines. An object of class "owin" or something acceptable to as.owin .

Details

The grid is randomly displaced from the origin.

Value

A line segment pattern (object of class "psp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[psp](#), [rpoisline](#)

Examples

```
plot(rlinegrid(30, 0.05))
```

rMatClust*Simulate Matern Cluster Process***Description**

Generate a random point pattern, a simulated realisation of the Mat'ern Cluster Process.

Usage

```
rMatClust(kappa, r, mu, win = owin(c(0,1),c(0,1)))
```

Arguments

<code>kappa</code>	Intensity of the Poisson process of cluster centres. A single positive number, a function, or a pixel image.
<code>r</code>	Radius parameter of the clusters.
<code>mu</code>	Mean number of points per cluster (a single positive number) or reference intensity for the cluster points (a function or a pixel image).
<code>win</code>	Window in which to simulate the pattern. An object of class "owin" or something acceptable to as.owin .

Details

This algorithm generates a realisation of Mat'ern's cluster process inside the window `win`. The process is constructed by first generating a Poisson point process of "parent" points with intensity `kappa`. Then each parent point is replaced by a random cluster of points, the number of points in each cluster being random with a Poisson (`mu`) distribution, and the points being placed independently and uniformly inside a disc of radius `r` centred on the parent point.

In this implementation, parent points are not restricted to lie in the window; the parent process is effectively the uniform Poisson process on the infinite plane.

This classical model can be fitted to data by the method of minimum contrast, using [matclust.estK](#) or [kppm](#).

The algorithm can also generate spatially inhomogeneous versions of the Mat'ern cluster process:

- The parent points can be spatially inhomogeneous. If the argument `kappa` is a `function(x,y)` or a pixel image (object of class "im"), then it is taken as specifying the intensity function of an inhomogeneous Poisson process that generates the parent points.
- The offspring points can be inhomogeneous. If the argument `mu` is a `function(x,y)` or a pixel image (object of class "im"), then it is interpreted as the reference density for offspring points, in the sense of Waagepetersen (2006). For a given parent point, the offspring constitute a Poisson process with intensity function equal to the *average* value of `mu` inside the disc of radius `r` centred on the parent point, and zero intensity outside this disc.

When the parents are homogeneous (`kappa` is a single number) and the offspring are inhomogeneous (`mu` is a function or pixel image), the model can be fitted to data using [kppm](#), or using [matclust.estK](#) applied to the inhomogeneous *K* function.

Value

The simulated point pattern (an object of class "ppp").

Additionally, some intermediate results of the simulation are returned as attributes of this point pattern. See [rNeymanScott](#).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

Mat'ern, B. (1960) *Spatial Variation*. Meddelanden fraan Statens Skogsforkningsinstitut, volume 59, number 5. Statens Skogsforkningsinstitut, Sweden.

Mat'ern, B. (1986) *Spatial Variation*. Lecture Notes in Statistics 36, Springer-Verlag, New York.

Waagepetersen, R. (2006) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. Submitted for publication.

See Also

[rpoispp](#), [rThomas](#), [rGaussPoisson](#), [rNeymanScott](#), [matclust.estK](#), [kppm](#).

Examples

```
# homogeneous
X <- rMatClust(10, 0.05, 4)
# inhomogeneous
Z <- as.im(function(x,y){ 4 * exp(2 * x - 1) }, owin())
Y <- rMatClust(10, 0.05, Z)
```

Description

Generate a random point pattern, a simulated realisation of the Mat'ern Model I inhibition process model.

Usage

```
rMaternI(kappa, r, win = owin(c(0,1),c(0,1)), stationary=TRUE)
```

Arguments

<code>kappa</code>	Intensity of the Poisson process of proposal points. A single positive number.
<code>r</code>	Inhibition distance.
<code>win</code>	Window in which to simulate the pattern. An object of class "owin" or something acceptable to as.owin .
<code>stationary</code>	Logical. Whether to start with a stationary process of proposal points (<code>stationary=TRUE</code>) or to generate the proposal points only inside the window (<code>stationary=FALSE</code>).

Details

This algorithm generates a realisation of Mat'ern's Model I inhibition process inside the window `win`.

The process is constructed by first generating a uniform Poisson point process of "proposal" points with intensity `kappa`. If `stationary = TRUE` (the default), the proposal points are generated in a window larger than `win` that effectively means the proposals are stationary. If `stationary=FALSE` then the proposal points are only generated inside the window `win`.

A proposal point is then deleted if it lies within `r` units' distance of another proposal point. Otherwise it is retained.

The retained points constitute Mat'ern's Model I.

Value

The simulated point pattern (an object of class "ppp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>, Ute Hahn, and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[rpoispp](#), [rMatClust](#)

Examples

```
X <- rMaternI(20, 0.05)
Y <- rMaternI(20, 0.05, stationary=FALSE)
```

rMaternII

Simulate Matern Model II

Description

Generate a random point pattern, a simulated realisation of the Mat'ern Model II inhibition process.

Usage

```
rMaternII(kappa, r, win = owin(c(0,1),c(0,1)), stationary=TRUE)
```

Arguments

<code>kappa</code>	Intensity of the Poisson process of proposal points. A single positive number.
<code>r</code>	Inhibition distance.
<code>win</code>	Window in which to simulate the pattern. An object of class "owin" or something acceptable to as.owin .
<code>stationary</code>	Logical. Whether to start with a stationary process of proposal points (<code>stationary=TRUE</code>) or to generate the proposal points only inside the window (<code>stationary=FALSE</code>).

Details

This algorithm generates a realisation of Mat'ern's Model II inhibition process inside the window `win`.

The process is constructed by first generating a uniform Poisson point process of "proposal" points with intensity `kappa`. If `stationary = TRUE` (the default), the proposal points are generated in a window larger than `win` that effectively means the proposals are stationary. If `stationary=FALSE` then the proposal points are only generated inside the window `win`.

Then each proposal point is marked by an "arrival time", a number uniformly distributed in $[0, 1]$ independently of other variables.

A proposal point is deleted if it lies within `r` units' distance of another proposal point *that has an earlier arrival time*. Otherwise it is retained. The retained points constitute Mat'ern's Model II.

The difference between Mat'ern's Model I and II is the italicised statement above. Model II has a higher intensity for the same parameter values.

Value

The simulated point pattern (an object of class "ppp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>, Ute Hahn, and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[rpoispp](#), [rMatClust](#), [rMaternI](#)

Examples

```
X <- rMaternII(20, 0.05)
Y <- rMaternII(20, 0.05, stationary=FALSE)
```

rmh

Simulate point patterns using the Metropolis-Hastings algorithm.

Description

Generic function for running the Metropolis-Hastings algorithm to produce simulated realisations of a point process model.

Usage

```
rmh(model, ...)
```

Arguments

- | | |
|-------|---|
| model | The point process model to be simulated. |
| ... | Further arguments controlling the simulation. |

Details

The Metropolis-Hastings algorithm can be used to generate simulated realisations from a wide range of spatial point processes. For caveats, see below.

The function `rmh` is generic; it has methods `rmh.ppm` (for objects of class "ppm") and `rmh.default` (the default). The actual implementation of the Metropolis-Hastings algorithm is contained in `rmh.default`. For details of its use, see `rmh.ppm` or `rmh.default`.

[If the model is a Poisson process, then Metropolis-Hastings is not used; the Poisson model is generated directly using `rpoispp` or `rmpoispp`.]

In brief, the Metropolis-Hastings algorithm is a Markov Chain, whose states are spatial point patterns, and whose limiting distribution is the desired point process. After running the algorithm for a very large number of iterations, we may regard the state of the algorithm as a realisation from the desired point process.

However, there are difficulties in deciding whether the algorithm has run for "long enough". The convergence of the algorithm may indeed be extremely slow. No guarantees of convergence are given!

While it is fashionable to decry the Metropolis-Hastings algorithm for its poor convergence and other properties, it has the advantage of being easy to implement for a wide range of models.

Value

A point pattern, in the form of an object of class "ppp". See `rmh.default` for details.

Warning

As of version 1.22-1 of `spatstat` a subtle change was made to `rmh.default()`. We had noticed that the results produced were sometimes not "scalable" in that two models, differing in effect only by the units in which distances are measured and starting from the same seed, gave different results. This was traced to an idiosyncracy of floating point arithmetic. The code of `rmh.default()` has been changed so that the results produced by `rmh` are now scalable. The downside of this is that code which users previously ran may now give results which are different from what they formerly were.

In order to recover former behaviour (so that previous results can be reproduced) set `spatstat.options(scalable=FALSE)`. See the last example in the help for `rmh.default`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`rmh.default`

Examples

```
# See examples in rmh.default and rmh.ppm
```

rmh.default	<i>Simulate Point Process Models using the Metropolis-Hastings Algorithm.</i>
-------------	---

Description

Generates a random point pattern, simulated from a chosen point process model, using the Metropolis-Hastings algorithm.

Usage

```
## Default S3 method:  
rmh(model, start=NULL,  
    control=default.rmhcontrol(model),  
    verbose=TRUE, track=FALSE, ...)
```

Arguments

model	Data specifying the point process model that is to be simulated.
start	Data determining the initial state of the algorithm.
control	Data controlling the iterative behaviour and termination of the algorithm.
verbose	Logical flag indicating whether to print progress reports.
track	Logical flag indicating whether to save the transition history of the simulations.
...	Further arguments passed to rmhcontrol or to trend functions in model.

Details

This function generates simulated realisations from any of a range of spatial point processes, using the Metropolis-Hastings algorithm. It is the default method for the generic function [rmh](#).

This function executes a Metropolis-Hastings algorithm with birth, death and shift proposals as described in Geyer and Moller (1994).

The argument `model` specifies the point process model to be simulated. It is either a list, or an object of class "rmhmodel", with the following components:

- cif** A character string specifying the choice of interpoint interaction for the point process.
- par** Parameter values for the conditional intensity function.
- w** (Optional) window in which the pattern is to be generated. An object of class "owin", or data acceptable to [as.owin](#).
- trend** Data specifying the spatial trend in the model, if it has a trend. This may be a function, a pixel image (of class "im"), (or a list of functions or images if the model is multitype). If the trend is a function or functions, any auxiliary arguments ... to `rmh.default` will be passed to these functions, which should be of the form `function(x, y, ...)`.
- types** List of possible types, for a multitype point process.

For full details of these parameters, see [rmhmodel](#).

The argument `start` determines the initial state of the Metropolis-Hastings algorithm. It is either `NULL`, or an object of class "rmhstart", or a list with the following components:

n.start Number of points in the initial point pattern. A single integer, or a vector of integers giving the numbers of points of each type in a multitype point pattern. Incompatible with **x.start**.

x.start Initial point pattern configuration. Incompatible with **n.start**.

x.start may be a point pattern (an object of class "ppp"), or data which can be coerced to this class by [as.ppp](#), or an object with components **x** and **y**, or a two-column matrix. In the last two cases, the window for the pattern is determined by **model\$w**. In the first two cases, if **model\$w** is also present, then the final simulated pattern will be clipped to the window **model\$w**.

For full details of these parameters, see [rmhstart](#).

The third argument **control** controls the simulation procedure (including *conditional simulation*), iterative behaviour, and termination of the Metropolis-Hastings algorithm. It is either NULL, or a list, or an object of class "rmhcontrol", with components:

p The probability of proposing a "shift" (as opposed to a birth or death) in the Metropolis-Hastings algorithm.

q The conditional probability of proposing a death (rather than a birth) given that birth/death has been chosen over shift.

nrep The number of repetitions or iterations to be made by the Metropolis-Hastings algorithm. It should be large.

expand Either a numerical expansion factor, or a window (object of class "owin"). Indicates that the process is to be simulated on a larger domain than the original data window **w**, then clipped to **w** when the algorithm has finished.

The default is to expand the simulation window if the model is stationary and non-Poisson (i.e. it has no trend and the interaction is not Poisson) and not to expand in all other cases.

If the model has a trend, then in order for expansion to be feasible, the trend must be given either as a function, or an image whose bounding box is large enough to contain the expanded window.

periodic A logical scalar; if **periodic** is TRUE we simulate a process on the torus formed by identifying opposite edges of a rectangular window.

ptypes A vector of probabilities (summing to 1) to be used in assigning a random type to a new point.

fixall A logical scalar specifying whether to condition on the number of points of each type.

nverb An integer specifying how often "progress reports" (which consist simply of the number of repetitions completed) should be printed out. If **nverb** is left at 0, the default, the simulation proceeds silently.

x.cond If this argument is present, then *conditional simulation* will be performed, and **x.cond** specifies the conditioning points and the type of conditioning.

For full details of these parameters, see [rmhcontrol](#). The control parameters can also be given in the ... arguments.

Value

A point pattern (an object of class "ppp", see [ppp.object](#)).

The returned value has an attribute **info** containing modified versions of the arguments **model**, **start**, and **control** which together specify the exact simulation procedure. The **info** attribute can be printed (and is printed automatically by [summary.ppp](#)).

The value of [.Random.seed](#) at the start of the simulations is also saved and returned as an attribute **seed**.

If `track=TRUE`, the transition history of the algorithm is saved, and returned as an attribute `history`. The transition history is a data frame containing a factor `proposaltype` identifying the proposal type (Birth, Death or Shift) and a logical vector `accepted` indicating whether the proposal was accepted.

Conditional Simulation

There are several kinds of conditional simulation.

- Simulation *conditional upon the number of points*, that is, holding the number of points fixed. To do this, set `control$p` (the probability of a shift) equal to 1. The number of points is then determined by the starting state, which may be specified either by setting `start$n.start` to be a scalar, or by setting the initial pattern `start$x.start`.
- In the case of multitype processes, it is possible to simulate the model *conditionally upon the number of points of each type*, i.e. holding the number of points of each type to be fixed. To do this, set `control$p` equal to 1 and `control$fixall` to be `TRUE`. The number of points is then determined by the starting state, which may be specified either by setting `start$n.start` to be an integer vector, or by setting the initial pattern `start$x.start`.
- Simulation *conditional on the configuration observed in a sub-window*, that is, requiring that, inside a specified sub-window V , the simulated pattern should agree with a specified point pattern y . To do this, set `control$x.cond` to equal the specified point pattern y , making sure that it is an object of class "ppp" and that the window `as.owin(control$x.cond)` is the conditioning window V .
- Simulation *conditional on the presence of specified points*, that is, requiring that the simulated pattern should include a specified set of points. This is simulation from the Palm distribution of the point process given a pattern y . To do this, set `control$x.cond` to be a `data.frame` containing the coordinates (and marks, if appropriate) of the specified points.

For further information, see [rmhcontrol](#).

Note that, when we simulate conditionally on the number of points, or conditionally on the number of points of each type, no expansion of the window is possible.

Warnings

There is never a guarantee that the Metropolis-Hastings algorithm has converged to its limiting distribution.

If `start$x.start` is specified then `expand` is set equal to 1 and simulation takes place in `x.start>window`. Any specified value for `expand` is simply ignored.

The presence of both a component `w` of `model` and a non-null value for `x.start>window` makes sense ONLY if `w` is contained in `x.start>window`.

For multitype processes make sure that, even if there is to be no trend corresponding to a particular type, there is still a component (a `NULL` component) for that type, in the list.

Other models

In theory, any finite point process model can be simulated using the Metropolis-Hastings algorithm, provided the conditional intensity is uniformly bounded.

In practice, the list of point process models that can be simulated using `rmh.default` is limited to those that have been implemented in the package's internal C code. More options will be added in the future.

Note that the lookup conditional intensity function permits the simulation (in theory, to any desired degree of approximation) of any pairwise interaction process for which the interaction depends only on the distance between the pair of points.

Reproducible simulations

If the user wants the simulation to be exactly reproducible (e.g. for a figure in a journal article, where it is useful to have the figure consistent from draft to draft) then the state of the random number generator should be set before calling `rmh.default`. This can be done either by calling `set.seed` or by assigning a value to `.Random.seed`. In the examples below, we use `set.seed`.

If a simulation has been performed and the user now wants to repeat it exactly, the random seed should be extracted from the simulated point pattern `X` by `seed <- attr(x, "seed")`, then assigned to the system random number state by `.Random.seed <- seed` before calling `rmh.default`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283 – 322.
- 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.
- Diggle, P.J., Gates, D.J., and Stibbard, A. (1987) A nonparametric estimator for pairwise-interaction point processes. *Biometrika* **74**, 763 – 770.
- Geyer, C.J. and Moller, J. (1994) Simulation procedures and likelihood inference for spatial point processes. *Scandinavian Journal of Statistics* **21**, 359–373.
- 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

`rmh`, `rmh.ppm`, `rStrauss`, `ppp`, `ppm`, `AreaInter`, `BadGey`, `DiggleGatesStibbard`, `DiggleGratton`, `Fiksel`, `Geyer`, `Hardcore`, `LennardJones`, `MultiHard`, `MultiStrauss`, `MultiStraussHard`, `PairPiece`, `Poisson`, `Softcore`, `Strauss`, `StraussHard`, `Triplets`

Examples

```
## Not run:
nr <- 1e5
nv <- 5000
ns <- 200

## End(Not run)

set.seed(961018)
```

```

# Strauss process.
mod01 <- list(cif="strauss",par=list(beta=2,gamma=0.2,r=0.7),
               w=c(0,10,0,10))
X1.strauss <- rmh(model=mod01,start=list(n.start=ns),
                   control=list(nrep=nr,nverb=nv))

# Strauss process, conditioning on n = 42:
X2.strauss <- rmh(model=mod01,start=list(n.start=42),
                   control=list(p=1,nrep=nr,nverb=nv))

# Hard core process:
mod02 <- list(cif="hardcore",par=list(beta=2,hc=0.7),w=c(0,10,0,10))
X3.hardcore <- rmh(model=mod02,start=list(n.start=ns),
                     control=list(nrep=nr,nverb=nv))

# Strauss process equal to pure hardcore:
mod02s <- list(cif="strauss",par=list(beta=2,gamma=0,r=0.7),w=c(0,10,0,10))
X3.strauss <- rmh(model=mod02s,start=list(n.start=ns),
                     control=list(nrep=nr,nverb=nv))

# Strauss process in a polygonal window.
x      <- c(0.55,0.68,0.75,0.58,0.39,0.37,0.19,0.26,0.42)
y      <- c(0.20,0.27,0.68,0.99,0.80,0.61,0.45,0.28,0.33)
mod03 <- list(cif="strauss",par=list(beta=2000,gamma=0.6,r=0.07),
               w=owin(poly=list(x=x,y=y)))
X4.strauss <- rmh(model=mod03,start=list(n.start=ns),
                     control=list(nrep=nr,nverb=nv))

# Strauss process in a polygonal window, conditioning on n = 80.
X5.strauss <- rmh(model=mod03,start=list(n.start=ns),
                     control=list(p=1,nrep=nr,nverb=nv))

# Strauss process, starting off from X4.strauss, but with the
# polygonal window replace by a rectangular one. At the end,
# the generated pattern is clipped to the original polygonal window.
xxx <- X4.strauss
xxx$window <- as.owin(c(0,1,0,1))
X6.strauss <- rmh(model=mod03,start=list(x.start=xxx),
                     control=list(nrep=nr,nverb=nv))

# Strauss with hardcore:
mod04 <- list(cif="straush",par=list(beta=2,gamma=0.2,r=0.7,hc=0.3),
               w=c(0,10,0,10))
X1.straush <- rmh(model=mod04,start=list(n.start=ns),
                     control=list(nrep=nr,nverb=nv))

# Another Strauss with hardcore (with a perhaps surprising result):
mod05 <- list(cif="straush",par=list(beta=80,gamma=0.36,r=45,hc=2.5),
               w=c(0,250,0,250))
X2.straush <- rmh(model=mod05,start=list(n.start=ns),
                     control=list(nrep=nr,nverb=nv))

# Pure hardcore (identical to X3.strauss).
mod06 <- list(cif="straush",par=list(beta=2,gamma=1,r=1,hc=0.7),
               w=c(0,10,0,10))

```

```

X3.straush <- rmh(model=mod06,start=list(n.start=ns),
                  control=list(nrep=nr,nverb=nv))

# Soft core:
w    <- c(0,10,0,10)
mod07 <- list(cif="sftcr",par=list(beta=0.8,sigma=0.1,kappa=0.5),
              w=c(0,10,0,10))
X.sftcr <- rmh(model=mod07,start=list(n.start=ns),
                 control=list(nrep=nr,nverb=nv))

# Area-interaction process:
mod42 <- rmhmodel(cif="areaint",par=list(beta=2,eta=1.6,r=0.7),
                   w=c(0,10,0,10))
X.area <- rmh(model=mod42,start=list(n.start=ns),
                control=list(nrep=nr,nverb=nv))

# Triplets process
modtrip <- list(cif="triplets",par=list(beta=2,gamma=0.2,r=0.7),
                  w=c(0,10,0,10))
X.triplets <- rmh(model=modtrip,
                     start=list(n.start=ns),
                     control=list(nrep=nr,nverb=nv))

# Multitype Strauss:
beta <- c(0.027,0.008)
gmma <- matrix(c(0.43,0.98,0.98,0.36),2,2)
r    <- matrix(c(45,45,45,45),2,2)
mod08 <- list(cif="straussm",par=list(beta=beta,gamma=gmma,radii=r),
               w=c(0,250,0,250))
X1.straussm <- rmh(model=mod08,start=list(n.start=ns),
                      control=list(ptypes=c(0.75,0.25),nrep=nr,nverb=nv))

# Multitype Strauss conditioning upon the total number
# of points being 80:
X2.straussm <- rmh(model=mod08,start=list(n.start=ns),
                      control=list(p=1,ptypes=c(0.75,0.25),nrep=nr,
                                   nverb=nv))

# Conditioning upon the number of points of type 1 being 60
# and the number of points of type 2 being 20:
X3.straussm <- rmh(model=mod08,start=list(n.start=c(60,20)),
                      control=list(fixall=TRUE,p=1,ptypes=c(0.75,0.25),
                                   nrep=nr,nverb=nv))

# Multitype Strauss hardcore:
rhc  <- matrix(c(9.1,5.0,5.0,2.5),2,2)
mod09 <- list(cif="straushm",par=list(beta=beta,gamma=gmma,
                                         iradii=r,hradii=rhc,w=c(0,250,0,250)))
X.straushm <- rmh(model=mod09,start=list(n.start=ns),
                     control=list(ptypes=c(0.75,0.25),nrep=nr,nverb=nv))

# Multitype Strauss hardcore with trends for each type:
beta  <- c(0.27,0.08)
tr3   <- function(x,y){x <- x/250; y <- y/250;
                        exp((6*x + 5*y - 18*x^2 + 12*x*y - 9*y^2)/6)
                        }
                        # log quadratic trend

```

```

tr4   <- function(x,y){x <- x/250; y <- y/250;
                      exp(-0.6*x+0.5*y)}
      # log linear trend
mod10 <- list(cif="straushm",par=list(beta=beta,gamma=gmma,
                                         iradii=r,hradii=rhc),w=c(0,250,0,250),
                           trend=list(tr3,tr4))
X1.straushm.trend <- rmh(model=mod10,start=list(n.start=ns),
                           control=list(ptypes=c(0.75,0.25),
                                         nrep=nr,nverb=nv))

# Multitype Strauss hardcore with trends for each type, given as images:
bigwin <- square(250)
i1 <- as.im(tr3, bigwin)
i2 <- as.im(tr4, bigwin)
mod11 <- list(cif="straushm",par=list(beta=beta,gamma=gmma,
                                         iradii=r,hradii=rhc),w=bigwin,
                           trend=list(i1,i2))
X2.straushm.trend <- rmh(model=mod11,start=list(n.start=ns),
                           control=list(ptypes=c(0.75,0.25),expand=1,
                                         nrep=nr,nverb=nv))

# Diggle, Gates, and Stibbard:
mod12 <- list(cif="dgs",par=list(beta=3600,rho=0.08),w=c(0,1,0,1))
X.dgs <- rmh(model=mod12,start=list(n.start=ns),
              control=list(nrep=nr,nverb=nv))

# Diggle-Gratton:
mod13 <- list(cif="diggra",
               par=list(beta=1800,kappa=3,delta=0.02,rho=0.04),
               w=square(1))
X.diggra <- rmh(model=mod13,start=list(n.start=ns),
                  control=list(nrep=nr,nverb=nv))

# Fiksel:
modFik <- list(cif="fiksel",
                 par=list(beta=180,r=0.15,hc=0.07,kappa=2,a= -1.0),
                 w=square(1))
X.diggra <- rmh(model=modFik,start=list(n.start=ns),
                  control=list(nrep=nr,nverb=nv))

# Geyer:
mod14 <- list(cif="geyer",par=list(beta=1.25,gamma=1.6,r=0.2,sat=4.5),
               w=c(0,10,0,10))
X1.geyer <- rmh(model=mod14,start=list(n.start=ns),
                  control=list(nrep=nr,nverb=nv))

# Geyer; same as a Strauss process with parameters
# (beta=2.25,gamma=0.16,r=0.7):

mod15 <- list(cif="geyer",par=list(beta=2.25,gamma=0.4,r=0.7,sat=10000),
               w=c(0,10,0,10))
X2.geyer <- rmh(model=mod15,start=list(n.start=ns),
                  control=list(nrep=nr,nverb=nv))

mod16 <- list(cif="geyer",par=list(beta=8.1,gamma=2.2,r=0.08,sat=3))
data(redwood)
X3.geyer <- rmh(model=mod16,start=list(x.start=redwood),

```

```

control=list(periodic=TRUE,nrep=nr,nverb=nv))

# Geyer, starting from the redwood data set, simulating
# on a torus, and conditioning on n:
X4.geyer <- rmh(model=mod16,start=list(x.start=redwood),
                 control=list(p=1,periodic=TRUE,nrep=nr,nverb=nv))

# Lookup (interaction function h_2 from page 76, Diggle (2003)):
r <- seq(from=0,to=0.2,length=101)[-1] # Drop 0.
h <- 20*(r-0.05)
h[r<0.05] <- 0
h[r>0.10] <- 1
mod17 <- list(cif="lookup",par=list(beta=4000,h=h,r=r),w=c(0,1,0,1))
X.lookup <- rmh(model=mod17,start=list(n.start=ns),
                  control=list(nrep=nr,nverb=nv))

# Strauss with trend
tr <- function(x,y){x <- x/250; y <- y/250;
                      exp((6*x + 5*y - 18*x^2 + 12*x*y - 9*y^2)/6)
}
beta <- 0.3
gmma <- 0.5
r     <- 45
mod17 <- list(cif="strauss",par=list(beta=beta,gamma=gmma,r=r),
               w=square(1), trend=tr3)
X1.strauss.trend <- rmh(model=mod17,start=list(n.start=ns),
                           control=list(nrep=nr,nverb=nv))

# Baddeley-Geyer
r <- seq(0,0.2,length=8)[-1]
gmma <- c(0.5,0.6,0.7,0.8,0.7,0.6,0.5)
mod18 <- list(cif="badgey",par=list(beta=4000, gamma=gmma,r=r,sat=5),
               w=square(1))
X1.badgey <- rmh(model=mod18,start=list(n.start=ns),
                   control=list(nrep=nr,nverb=nv))
mod19 <- list(cif="badgey",
               par=list(beta=4000, gamma=gmma,r=r,sat=1e4),
               w=square(1))
set.seed(1329)
X2.badgey <- rmh(model=mod18,start=list(n.start=ns),
                   control=list(nrep=nr,nverb=nv))

# Check:
h <- ((prod(gmma)/cumprod(c(1,gmma)))[-8])^2
hs <- stepfun(r,c(h,1))
mod20 <- list(cif="lookup",par=list(beta=4000,h=hs),w=square(1))
set.seed(1329)
X.check <- rmh(model=mod20,start=list(n.start=ns),
                  control=list(nrep=nr,nverb=nv))
# X2.badgey and X.check will be identical.

mod21 <- list(cif="badgey",par=list(beta=300,gamma=c(1,0.4,1),
                                         r=c(0.035,0.07,0.14),sat=5, w=square(1)))
X3.badgey <- rmh(model=mod21,start=list(n.start=ns),
                   control=list(nrep=nr,nverb=nv))
# Same result as Geyer model with beta=300, gamma=0.4, r=0.07,
# sat = 5 (if seeds and control parameters are the same)

```

```

# Or more simply:
mod22 <- list(cif="badgey",
               par=list(beta=300,gamma=0.4,r=0.07, sat=5),
               w=square(1))
X4.badgey <- rmh(model=mod22,start=list(n.start=ns),
                  control=list(nrep=nr,nverb=nv))
# Same again --- i.e. the BadGey model includes the Geyer model.

# Illustrating scalability.
## Not run:
M1 <- rmhmodel(cif="strauss",par=list(beta=60,gamma=0.5,r=0.04),w=owin())
set.seed(496)
X1 <- rmh(model=M1,start=list(n.start=300))
M2 <- rmhmodel(cif="strauss",par=list(beta=0.6,gamma=0.5,r=0.4),
                w=owin(c(0,10),c(0,10)))
set.seed(496)
X2 <- rmh(model=M2,start=list(n.start=300))
chk <- affine(X1,mat=diag(c(10,10)))
all.equal(chk,X2,check.attribute=FALSE)
# Under the default spatstat options the foregoing all.equal()
# will yield TRUE. Setting spatstat.options(scalable=FALSE) and
# re-running the code will reveal differences between X1 and X2.

## End(Not run)

```

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),
     ...,
     project=TRUE, verbose=TRUE)

```

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 <code>list(x.start=data.ppm(model))</code> |
| control | Data controlling the iterative behaviour of the Metropolis-Hastings algorithm. See rmhcontrol for description of these arguments. |

...	Further arguments passed to <code>rmhcontrol</code> , or to <code>rmh.default</code> , 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 <code>project=TRUE</code> the closest valid model will be simulated; if <code>project=FALSE</code> an error will occur.
verbose	Logical flag indicating whether to print progress reports.

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 `DiggleGratton`, `Geyer`, `MultiStrauss`, `MultiStraussHard`, `PairPiece`, `Poisson`, `Strauss`, `StraussHard` and `Softcore`, including nonstationary models. See 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`).

Warnings

See Warnings in [rmh.default](#).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[simulate.ppm](#), [rmh](#), [rmhmodel](#), [rmhcontrol](#), [default.rmhcontrol](#), [update.rmhcontrol](#), [rmhstart](#),
[rmh.default](#), [ppp.object](#), [ppm](#), [PairPiece](#), [Poisson](#), [Strauss](#), [StraussHard](#), [Softcore](#), [Geyer](#),
[AreaInter](#), [DiggleGratton](#)

Examples

```
live <- interactive()
op <- spatstat.options()
spatstat.options(rmh.nrep=1e5)
Nrep <- 1e5

data(swedishpines)
X <- swedishpines
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")

# Strauss process simulated on a larger window
# then clipped to original window
Xsim <- rmh(fit, control=list(nrep=Nrep, expand=2, periodic=TRUE))
Xsim <- rmh(fit, nrep=Nrep, expand=2, periodic=TRUE)

# Strauss - hard core process
fit <- ppm(X, ~1, StraussHard(r=7, hc=2))
Xsim <- rmh(fit, start=list(n.start=X$n))
if(live) plot(Xsim, main="simulation from fitted Strauss hard core model")

# Geyer saturation process
fit <- ppm(X, ~1, Geyer(r=7,sat=2))
Xsim <- rmh(fit, start=list(n.start=X$n))
if(live) plot(Xsim, main="simulation from fitted Geyer model")

# Area-interaction process
fit <- ppm(X, ~1, AreaInter(r=7))
Xsim <- rmh(fit, start=list(n.start=X$n))
```

```

if(live) plot(Xsim, main="simulation from fitted area-interaction model")

# soft core interaction process
Q <- quadscheme(X, nd=50)
fit <- ppm(Q, ~1, Softcore(kappa=0.1), correction="isotropic")
Xsim <- rmh(fit, start=list(n.start=X$n))
if(live) plot(Xsim, main="simulation from fitted Soft Core model")

data(cells)
if(live) plot(cells)
# Diggle-Gratton pairwise interaction model
fit <- ppm(cells, ~1, DiggleGratton(0.05, 0.1))
Xsim <- rmh(fit, start=list(n.start=cells$n))
if(live) plot(Xsim, main="simulation from fitted Diggle-Gratton model")

X <- rSSI(0.05, 100)
if(live) plot(X, main="new data")

# piecewise-constant pairwise interaction function
fit <- ppm(X, ~1, PairPiece(seq(0.02, 0.1, by=0.01)))
Xsim <- rmh(fit)
if(live) plot(Xsim, main="simulation from fitted pairwise model")

# marked point pattern
data(amacrine)
Y <- amacrine
if(live) plot(Y, main="Amacrine data")

# marked Poisson models
fit <- ppm(Y)
Ysim <- rmh(fit)
if(live) plot(Ysim, main="simulation from ppm(Y)")

fit <- ppm(Y,~marks)
Ysim <- rmh(fit)
if(live) plot(Ysim, main="simulation from ppm(Y, ~marks)")

fit <- ppm(Y,~polynom(x,y,2))
Ysim <- rmh(fit)
if(live) plot(Ysim, main="simulation from ppm(Y, ~polynom(x,y,2))")

fit <- ppm(Y,~marks+polynom(x,y,2))
Ysim <- rmh(fit)
if(live) plot(Ysim, main="simulation from ppm(Y, ~marks+polynom(x,y,2))")

fit <- ppm(Y,~marks*polynom(x,y,2))
Ysim <- rmh(fit)
if(live) plot(Ysim, main="simulation from ppm(Y, ~marks*polynom(x,y,2))")

# multitype Strauss models
MS <- MultiStrauss(types = levels(Y$marks),
                      radii=matrix(0.07, ncol=2, nrow=2))
fit <- ppm(Y, ~marks, MS)
Ysim <- rmh(fit)
if(live) plot(Ysim, main="simulation from fitted Multitype Strauss")

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)

```

rmhcontrol*Set Control Parameters for Metropolis-Hastings Algorithm.***Description**

Sets up a list of parameters controlling the iterative behaviour of the Metropolis-Hastings algorithm.

Usage

```

rmhcontrol(...)

## Default S3 method:
rmhcontrol(..., p=0.9, q=0.5, nrep=5e5,
           expand=NULL, periodic=NULL, ptypes=NULL,
           x.cond=NULL, fixall=FALSE, nverb=0)

```

Arguments

...	Arguments passed to methods.
p	Probability of proposing a shift (as against a birth/death).
q	Conditional probability of proposing a death given that a birth or death will be proposed.
nrep	Total number of steps (proposals) of Metropolis-Hastings algorithm that should be run.
expand	Simulation window or expansion rule. Either a window (object of class "owin") or a numerical expansion factor, specifying that simulations are to be performed in a domain other than the original data window, then clipped to the original data window. This argument is passed to rmhexpand . A numerical expansion factor can be in several formats: see rmhexpand .
periodic	Logical value (or NULL) indicating whether to simulate “periodically”, i.e. identifying opposite edges of the rectangular simulation window. A NULL value means “undecided.”
ptypes	For multitype point processes, the distribution of the mark attached to a new random point (when a birth is proposed)
x.cond	Conditioning points for conditional simulation.
fixall	(Logical) for multitype point processes, whether to fix the number of points of each type.
nverb	Progress reports will be printed every nverb iterations

Details

The Metropolis-Hastings algorithm, implemented as `rmh`, generates simulated realisations of point process models. The function `rmhcontrol` sets up a list of parameters which control the iterative behaviour and termination of the Metropolis-Hastings algorithm, for use in a subsequent call to `rmh`. It also checks that the parameters are valid.

(A separate function `rmhstart` determines the initial state of the algorithm, and `rmhmodel` determines the model to be simulated.)

The parameters are as follows:

- p** The probability of proposing a “shift” (as opposed to a birth or death) in the Metropolis-Hastings algorithm.

If $p = 1$ then the algorithm only alters existing points, so the number of points never changes, i.e. we are simulating conditionally upon the number of points. The number of points is determined by the initial state (specified by `rmhstart`).

If $p = 1$ and `fixall=TRUE` and the model is a multitype point process model, then the algorithm only shifts the locations of existing points and does not alter their marks (types). This is equivalent to simulating conditionally upon the number of points of each type. These numbers are again specified by the initial state.

If $p = 1$ then no expansion of the simulation window is allowed (see `expand` below).

The default value of `p` can be changed by setting the parameter `rmh.p` in `spatstat.options`.

- q** The conditional probability of proposing a death (rather than a birth) given that a shift is not proposed. This is of course ignored if `p` is equal to 1.

The default value of `q` can be changed by setting the parameter `rmh.q` in `spatstat.options`.

- nrep** The number of repetitions or iterations to be made by the Metropolis-Hastings algorithm. It should be large.

The default value of `nrep` can be changed by setting the parameter `rmh.nrep` in `spatstat.options`.

- expand** Either a number or a window (object of class "owin"). Indicates that the process is to be simulated on a domain other than the original data window `w`, then clipped to `w` when the algorithm has finished. This would often be done in order to approximate the simulation of a stationary process (Geyer, 1999) or more generally a process existing in the whole plane, rather than just in the window `w`.

If `expand` is a window object, it is taken as the larger domain in which simulation is performed.

If `expand` is numeric, it is interpreted as an expansion factor or expansion distance for determining the simulation domain from the data window. It should be a *named* scalar, such as `expand=c(area=2)`, `expand=c(distance=0.1)`, `expand=c(length=1.2)`. See `rmhexpand()` for more details. If the name is omitted, it defaults to `area`.

Expansion is not permitted if the number of points has been fixed by setting `p = 1` or if the starting configuration has been specified via the argument `x.start` in `rmhstart`.

If `expand` is `NULL`, this is interpreted to mean “not yet decided”. An expansion rule will be determined at a later stage, using appropriate defaults. See `rmhexpand`.

- periodic** A logical value (or `NULL`) determining whether to simulate “periodically”. If `periodic` is `TRUE`, and if the simulation window is a rectangle, then the simulation algorithm effectively identifies opposite edges of the rectangle. Points near the right-hand edge of the rectangle are deemed to be close to points near the left-hand edge. Periodic simulation usually gives a better approximation to a stationary point process. For periodic simulation, the simulation window must be a rectangle. (The simulation window is determined by `expand` as described above.) The value `NULL` means ‘undecided’. The decision is postponed until `rmh` is called. Depending on the point process model to be simulated, `rmh` will then set `periodic=TRUE` if the simulation window is expanded *and* the expanded simulation window is rectangular; otherwise `periodic=FALSE`.

Note that `periodic=TRUE` is only permitted when the simulation window (i.e. the expanded window) is rectangular.

ptypes A vector of probabilities (summing to 1) to be used in assigning a random type to a new point. Defaults to a vector each of whose entries is $1/nt$ where nt is the number of types for the process. Convergence of the simulation algorithm should be improved if `ptypes` is close to the relative frequencies of the types which will result from the simulation.

x.cond If this argument is given, then *conditional simulation* will be performed, and `x.cond` specifies the location of the fixed points as well as the type of conditioning. It should be either a point pattern (object of class "ppp") or a list(`x,y`) or a `data.frame`. See the section on Conditional Simulation.

fixall A logical scalar specifying whether to condition on the number of points of each type. Meaningful only if a marked process is being simulated, and if $p = 1$. A warning message is given if `fixall` is set equal to `TRUE` when it is not meaningful.

nverb An integer specifying how often “progress reports” (which consist simply of the number of repetitions completed) should be printed out. If `nverb` is left at 0, the default, the simulation proceeds silently.

Value

An object of class "rmhcontrol", which is essentially a list of parameter values for the algorithm. There is a `print` method for this class, which prints a sensible description of the parameters chosen.

Conditional Simulation

For a Gibbs point process X , the Metropolis-Hastings algorithm easily accommodates several kinds of conditional simulation:

conditioning on the total number of points: We fix the total number of points $N(X)$ to be equal to n . We simulate from the conditional distribution of X given $N(X) = n$.

conditioning on the number of points of each type: In a multitype point process, where Y_j denotes the process of points of type j , we fix the number $N(Y_j)$ of points of type j to be equal to n_j , for $j = 1, 2, \dots, m$. We simulate from the conditional distribution of X given $N(Y_j) = n_j$ for $j = 1, 2, \dots, m$.

conditioning on the realisation in a subwindow: We require that the point process X should, within a specified sub-window V , coincide with a specified point pattern y . We simulate from the conditional distribution of X given $X \cap V = y$.

Palm conditioning: We require that the point process X include a specified list of points y . We simulate from the point process with probability density $g(x) = cf(x \cup y)$ where f is the probability density of the original process X , and c is a normalising constant.

To achieve each of these types of conditioning we do as follows:

conditioning on the total number of points: Set `p=1`. The number of points is determined by the initial state of the simulation: see [rmhstart](#).

conditioning on the number of points of each type: Set `p=1` and `fixall=TRUE`. The number of points of each type is determined by the initial state of the simulation: see [rmhstart](#).

conditioning on the realisation in a subwindow: Set `x.cond` to be a point pattern (object of class "ppp"). Its window `V=x.cond$window` becomes the conditioning subwindow V .

Palm conditioning: Set `x.cond` to be a list(`x,y`) or `data.frame` with two columns containing the coordinates of the points, or a list(`x,y,marks`) or `data.frame` with three columns containing the coordinates and marks of the points.

The arguments `x.cond`, `p` and `fixall` can be combined.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

References

Geyer, C.J. (1999) Likelihood Inference for Spatial Point Processes. Chapter 3 in O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. Van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*, Chapman and Hall / CRC, Monographs on Statistics and Applied Probability, number 80. Pages 79–140.

See Also

[rmh](#), [rmhmodel](#), [rmhstart](#), [rmhexpand](#), [spatstat.options](#)

Examples

```
# parameters given as named arguments
c1 <- rmhcontrol(p=0.3, periodic=TRUE, nrep=1e6, nverb=1e5)

# parameters given as a list
liz <- list(p=0.9, nrep=1e4)
c2 <- rmhcontrol(liz)

# parameters given in rmhcontrol object
c3 <- rmhcontrol(c1)
```

rmhexpand

Specify Simulation Window or Expansion Rule

Description

Specify a spatial domain in which point process simulations will be performed. Alternatively, specify a rule which will be used to determine the simulation window.

Usage

```
rmhexpand(x = NULL, ..., area = NULL, length = NULL, distance = NULL)
```

Arguments

- | | |
|-----------------------|---|
| <code>x</code> | Any kind of data determining the simulation window or the expansion rule. A window (object of class "owin") specifying the simulation window, a numerical value specifying an expansion factor or expansion distance, a list containing one numerical value, an object of class "rmhexpand", or NULL. |
| <code>...</code> | Ignored. |
| <code>area</code> | Area expansion factor. Incompatible with other arguments. |
| <code>length</code> | Length expansion factor. Incompatible with other arguments. |
| <code>distance</code> | Expansion distance (buffer width). Incompatible with other arguments. |

Details

In the Metropolis-Hastings algorithm [rmh](#) for simulating spatial point processes, simulations are usually carried out on a spatial domain that is larger than the original window of the point process model, then subsequently clipped to the original window.

The command `rmhexpand` can be used to specify the simulation window, or to specify a rule which will later be used to determine the simulation window from data.

The arguments are all incompatible: at most one of them should be given.

If the first argument `x` is given, it may be any of the following:

- a window (object of class "owin") specifying the simulation window.
- an object of class "rmhexpand" specifying the expansion rule.
- a single numerical value, without attributes. This will be interpreted as the value of the argument `area`.
- either `c(area=v)` or `list(area=v)`, where `v` is a single numeric value. This will be interpreted as the value of the argument `area`.
- either `c(length=v)` or `list(length=v)`, where `v` is a single numeric value. This will be interpreted as the value of the argument `length`.
- either `c(distance=v)` or `list(distance=v)`, where `v` is a single numeric value. This will be interpreted as the value of the argument `distance`.
- `NULL`, meaning that the expansion rule is not yet determined.

If one of the arguments `area`, `length` or `distance` is given, then the simulation window is determined from the original data window as follows.

area The bounding box of the original data window will be extracted, and the simulation window will be a scalar dilation of this rectangle. The argument `area` should be a numerical value, greater than or equal to 1. It specifies the area expansion factor, i.e. the ratio of the area of the simulation window to the area of the original point process window's bounding box.

length The bounding box of the original data window will be extracted, and the simulation window will be a scalar dilation of this rectangle. The argument `length` should be a numerical value, greater than or equal to 1. It specifies the length expansion factor, i.e. the ratio of the width (height) of the simulation window to the width (height) of the original point process window's bounding box.

distance The argument `distance` should be a numerical value, greater than or equal to 0. It specifies the width of a buffer region around the original data window. If the original data window is a rectangle, then this window is extended by a margin of width equal to `distance` around all sides of the original rectangle. The result is a rectangle. If the original data window is not a rectangle, then morphological dilation is applied using [dilation.owin](#) so that a margin or buffer of width equal to `distance` is created around all sides of the original window. The result is a non-rectangular window, typically of a different shape.

Value

An object of class "rmhexpand" specifying the expansion rule. There is a `print` method for this class.

Undetermined expansion

If `expand=NULL`, this is interpreted to mean that the expansion rule is “not yet decided”. Expansion will be decided later, by the simulation algorithm `rmh`. If the model cannot be expanded (for example if the covariate data in the model are not available on a larger domain) then expansion will not occur. If the model can be expanded, then if the point process model has a finite interaction range `r`, the default is `rmhexpand(distance=2*r)`, and otherwise `rmhexpand(area=2)`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`expand.owin` to apply the rule to a window.
`will.expand` to test whether expansion will occur.
`rmh`, `rmhcontrol` for background details.

Examples

```
rmhexpand()
rmhexpand(2)
rmhexpand(1)
rmhexpand(length=1.5)
rmhexpand(distance=0.1)
rmhexpand(letterR)
```

`rmhmodel`

Define Point Process Model for Metropolis-Hastings Simulation.

Description

Builds a description of a point process model for use in simulating the model by the Metropolis-Hastings algorithm.

Usage

```
rmhmodel(...)
```

Arguments

...	Arguments specifying the point process model in some format.
-----	--

Details

Simulated realisations of many point process models can be generated using the Metropolis-Hastings algorithm `rmh`. The algorithm requires the model to be specified in a particular format: an object of class “`rmhmodel`”.

The function `rmhmodel` takes a description of a point process model in some other format, and converts it into an object of class “`rmhmodel`”. It also checks that the parameters of the model are valid.

The function `rmhmodel` is generic, with methods for

fitted point process models: an object of class "ppm", obtained by a call to the model-fitting function [rmhmodel.ppm](#). See [rmhmodel.ppm](#).

lists: a list of parameter values in a certain format. See [rmhmodel.list](#).

default: parameter values specified as separate arguments to See [rmhmodel.default](#).

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Diggle, P. J. (2003) *Statistical Analysis of Spatial Point Patterns* (2nd ed.) Arnold, London.
- Diggle, P.J. and Gratton, R.J. (1984) Monte Carlo methods of inference for implicit statistical models. *Journal of the Royal Statistical Society, series B* **46**, 193 – 212.
- 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.
- 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

[rmhmodel.ppm](#), [rmhmodel.default](#), [rmhmodel.list](#), [rmh](#), [rmhcontrol](#), [rmhstart](#), [ppm](#), [Strauss](#), [Softcore](#), [StraussHard](#), [Triplets](#), [MultiStrauss](#), [MultiStraussHard](#), [DiggleGratton](#), [PairPiece](#)

[rmhmodel.default](#)

Build Point Process Model for Metropolis-Hastings Simulation.

Description

Builds a description of a point process model for use in simulating the model by the Metropolis-Hastings algorithm.

Usage

```
## Default S3 method:
rmhmodel(...,
         cif=NULL, par=NULL, w=NULL, trend=NULL, types=NULL)
```

Arguments

...	Ignored.
cif	Character string specifying the choice of model
par	Parameters of the model
w	Spatial window in which to simulate
trend	Specification of the trend in the model
types	A vector of factor levels defining the possible marks, for a multitype process.

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.default` is the default method. It builds a description of the point process model from the simple arguments listed.

The argument `cif` is a character string specifying the choice of interpoint interaction for the point process. The current options are

- 'areaint' Area-interaction process.
- 'badgely' Baddeley-Geyer (hybrid Geyer) process.
- 'dgs' Diggle, Gates and Stibbard (1987) process
- 'diggra' Diggle and Gratton (1984) process
- 'fiksel' Fiksel double exponential process (Fiksel, 1984).
- 'geyer' Saturation process (Geyer, 1999).
- 'hardcore' Hard core process
- 'lennard' Lennard-Jones process
- 'lookup' General isotropic pairwise interaction process, with the interaction function specified via a "lookup table".
- 'multihard' Multitype hardcore process
- 'strauss' The Strauss process
- 'straush' The Strauss process with hard core
- 'sftcr' The Softcore process
- 'straussm' The multitype Strauss process
- 'straushm' Multitype Strauss process with hard core
- 'triplets' Triplets process (Geyer, 1999).

The argument `par` supplies parameter values appropriate to the conditional intensity function being invoked. These are:

areaint: (Area-interaction process.) A **named** list with components `beta`, `eta`, `r` which are respectively the "base" intensity, the scaled interaction parameter and the interaction radius.

badgely: (Baddeley-Geyer process.) A **named** list with components `beta` (the "base" intensity), `gamma` (a vector of non-negative interaction parameters), `r` (a vector of interaction radii, of the same length as `gamma`, in *increasing* order), and `sat` (the saturation parameter(s); this may be a scalar, or a vector of the same length as `gamma` and `r`; all values should be at least 1). Note that because of the presence of "saturation" the `gamma` values are permitted to be larger than 1.

dgs: (Diggle, Gates, and Stibbard process. See Diggle, Gates, and Stibbard (1987)) A **named** list with components beta and rho. This process has pairwise interaction function equal to

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

for $t < \rho$, and equal to 1 for $t \geq \rho$.

diggra: (Diggle-Gratton process. See Diggle and Gratton (1984) and Diggle, Gates and Stibbard (1987).) A **named** list with components beta, kappa, delta and rho. This process has pairwise interaction function $e(t)$ equal to 0 for $t < \delta$, equal to

$$\left(\frac{t - \delta}{\rho - \delta}\right)^{\kappa}$$

for $\delta \leq t < \rho$, and equal to 1 for $t \geq \rho$. Note that here we use the symbol κ where Diggle, Gates, and Stibbard use β since we reserve the symbol β for an intensity parameter.

fiksel: (Fiksel double exponential process, see Fiksel (1984)) A **named** list with components beta, r, hc, kappa and a. This process has pairwise interaction function $e(t)$ equal to 0 for $t < hc$, equal to

$$\exp(a \exp(-\kappa t))$$

for $hc \leq t < r$, and equal to 1 for $t \geq r$.

geyer: (Geyer's saturation process. See Geyer (1999).) A **named** list with components beta, gamma, r, and sat. The components beta, gamma, r are as for the Strauss model, and sat is the "saturation" parameter. The model is Geyer's "saturation" point process model, a modification of the Strauss process in which we effectively impose an upper limit (sat) on the number of neighbours which will be counted as close to a given point.

Explicitly, a 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 " r -close neighbours" of x_i in the pattern X .

If the saturation threshold s is infinite, the Geyer process reduces to a Strauss process with interaction parameter γ^2 rather than γ .

hardcore: (Hard core process.) A **named** list with components beta and hc where beta is the base intensity and hc is the hard core distance. This process has pairwise interaction function $e(t)$ equal to 1 if $t > hc$ and 0 if $t \leq hc$.

lennard: (Lennard-Jones process.) A **named** list with components sigma and epsilon, where sigma is the characteristic diameter and epsilon is the well depth. See [LennardJones](#) for explanation.

multihard: (Multitype hard core process.) A **named** list with components beta and hradii, where beta is a vector of base intensities for each type of point, and hradii is a matrix of hard core radii between each pair of types.

strauss: (Strauss process.) A **named** list with components beta, gamma, r which are respectively the "base" intensity, the pairwise interaction parameter and the interaction radius. Note that gamma must be less than or equal to 1. (Note that there is also an algorithm for perfect simulation of the Strauss process, [rStrauss](#))

straush: (Strauss process with hardcore.) A **named** list with entries beta, gamma, r, hc where beta, gamma, and r are as for the Strauss process, and hc is the hardcore radius. Of course hc must be less than r.

sfter: (Softcore process.) A **named** list with components `beta`, `sigma`, `kappa`. Again `beta` is a “base” intensity. The pairwise interaction between two points $u \neq v$ is

$$\exp \left\{ - \left(\frac{\sigma}{\|u - v\|} \right)^{2/\kappa} \right\}$$

Note that it is necessary that $0 < \kappa < 1$.

straussm: (Multitype Strauss process.) A **named** list with components

- `beta`: A vector of “base” intensities, one for each possible type.
- `gamma`: A **symmetric** matrix of interaction parameters, with γ_{ij} pertaining to the interaction between type i and type j .
- `radii`: A **symmetric** matrix of interaction radii, with entries r_{ij} pertaining to the interaction between type i and type j .

straushm: (Multitype Strauss process with hardcore.) A **named** list with components `beta` and `gamma` as for `straussm` and **two** “radii” components:

- `iradii`: the interaction radii
- `hradii`: the hardcore radii

which are both symmetric matrices of nonnegative numbers. The entries of `hradii` must be less than the corresponding entries of `iradii`.

triplets: (Triplets process.) A **named** list with components `beta`, `gamma`, `r` which are respectively the “base” intensity, the triplet interaction parameter and the interaction radius. Note that `gamma` must be less than or equal to 1.

lookup: (Arbitrary pairwise interaction process with isotropic interaction.) A **named** list with components `beta`, `r`, and `h`, or just with components `beta` and `h`.

This model is the pairwise interaction process with an isotropic interaction given by any chosen function H . Each pair of points x_i, x_j in the point pattern contributes a factor $H(d(x_i, x_j))$ to the probability density, where d denotes distance and H is the pair interaction function.

The component `beta` is a (positive) scalar which determines the “base” intensity of the process.

In this implementation, H must be a step function. It is specified by the user in one of two ways.

- **as a vector of values:** If `r` is present, then `r` is assumed to give the locations of jumps in the function H , while the vector `h` gives the corresponding values of the function.

Specifically, the interaction function $H(t)$ takes the value $h[1]$ for distances t in the interval $[0, r[1]]$; takes the value $h[i]$ for distances t in the interval $[r[i-1], r[i]]$ where $i = 2, \dots, n$; and takes the value 1 for $t \geq r[n]$. Here n denotes the length of `r`.

The components `r` and `h` must be numeric vectors of equal length. The `r` values must be strictly positive, and sorted in increasing order.

The entries of `h` must be non-negative. If any entry of `h` is greater than 1, then the entry `h[1]` must be 0 (otherwise the specified process is non-existent).

Greatest efficiency is achieved if the values of `r` are equally spaced.

[Note: The usage of `r` and `h` has *changed* from the previous usage in **spatstat** versions 1.4-7 to 1.5-1, in which ascending order was not required, and in which the first entry of `r` had to be 0.]

- **as a stepfun object:** If `r` is absent, then `h` must be an object of class “`stepfun`” specifying a step function. Such objects are created by `stepfun`.

The `stepfun` object `h` must be right-continuous (which is the default using `stepfun`.)

The values of the step function must all be nonnegative. The values must all be less than 1 unless the function is identically zero on some initial interval $[0, r)$. The rightmost value (the value of `h(t)` for large `t`) must be equal to 1.

Greatest efficiency is achieved if the jumps (the “knots” of the step function) are equally spaced.

The optional argument `trend` determines the spatial trend in the model, if it has one. It should be a function or image (or a list of such, if the model is multitype) to provide the value of the trend at an arbitrary point.

trend given as a function: A trend function may be a function of any number of arguments, but the first two must be the x, y coordinates of a point. Auxiliary arguments may be passed to the trend function at the time of simulation, via the `...` argument to `rmh`.

The function **must be vectorized**. That is, it must be capable of accepting vector valued x and y arguments. Put another way, it must be capable of calculating the trend value at a number of points, simultaneously, and should return the **vector** of corresponding trend values.

trend given as an image: An image (see `im.object`) provides the trend values at a grid of points in the observation window and determines the trend value at other points as the value at the nearest grid point.

Note that the trend or trends must be **non-negative**; no checking is done for this.

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`.

The optional argument `types` specifies the possible types in a multitype point process. If the model being simulated is multitype, and `types` is not specified, then this vector defaults to `1:ntypes` where `ntypes` is the number of types.

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.

Warnings in Respect of “lookup”

For the lookup cif, the entries of the `r` component of `par` must be *strictly positive* and sorted into ascending order.

Note that if you specify the lookup pairwise interaction function via `stepfun()` the arguments `x` and `y` which are passed to `stepfun()` are slightly different from `r` and `h`: `length(y)` is equal to `1+length(x)`; the final entry of `y` must be equal to 1 — i.e. this value is explicitly supplied by the user rather than getting tacked on internally.

The step function returned by `stepfun()` must be right continuous (this is the default behaviour of `stepfun()`) otherwise an error is given.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Diggle, P. J. (2003) *Statistical Analysis of Spatial Point Patterns* (2nd ed.) Arnold, London.
- Diggle, P.J. and Gratton, R.J. (1984) Monte Carlo methods of inference for implicit statistical models. *Journal of the Royal Statistical Society, series B* **46**, 193 – 212.
- 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.

- Fiksel, T. (1984) Estimation of parameterized pair potentials of marked and non-marked Gibbsian point processes. *Electronische Informationsverarbeitung und Kybernetika* **20**, 270–278.
- 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

[rmh](#), [rmhcontrol](#), [rmhstart](#), [ppm](#), [AreaInter](#), [BadGey](#), [DiggleGatesStibbard](#), [DiggleGratton](#), [Fiksel](#), [Geyer](#), [Hardcore](#), [LennardJones](#), [MultiHard](#), [MultiStrauss](#), [MultiStraussHard](#), [PairPiece](#), [Poisson](#), [Softcore](#), [Strauss](#), [StraussHard](#), [Triplets](#)

Examples

```
# Strauss process:
mod01 <- rmhmodel(cif="strauss",par=list(beta=2,gamma=0.2,r=0.7),
                    w=c(0,10,0,10))
# The above could also be simulated using 'rStrauss'

# Strauss with hardcore:
mod04 <- rmhmodel(cif="straush",par=list(beta=2,gamma=0.2,r=0.7,hc=0.3),
                    w=owin(c(0,10),c(0,5)))

# Hard core:
mod05 <- rmhmodel(cif="hardcore",par=list(beta=2,hc=0.3),
                    w=square(5))

# Soft core:
w     <- square(10)
mod07 <- rmhmodel(cif="sftcr",
                   par=list(beta=0.8,sigma=0.1,kappa=0.5),
                   w=w)

# Area-interaction process:
mod42 <- rmhmodel(cif="areaint",par=list(beta=2,eta=1.6,r=0.7),
                    w=c(0,10,0,10))

# Baddeley-Geyer process:
mod99 <- rmhmodel(cif="badgey",par=list(beta=0.3,
                                             gamma=c(0.2,1.8,2.4),r=c(0.035,0.07,0.14),sat=5),
                                             w=unit.square())

# Multitype Strauss:
beta <- c(0.027,0.008)
gmma <- matrix(c(0.43,0.98,0.98,0.36),2,2)
r    <- matrix(c(45,45,45,45),2,2)
mod08 <- rmhmodel(cif="straussm",
                   par=list(beta=beta,gamma=gmma,radii=r),
                   w=square(250))

# specify types
mod09 <- rmhmodel(cif="straussm",
                   par=list(beta=beta,gamma=gmma,radii=r),
                   w=square(250),
                   types=c("A", "B"))
```

```

# Multitype Hardcore:
rhc <- matrix(c(9.1,5.0,5.0,2.5),2,2)
mod08hard <- rmhmodel(cif="multihard",
                       par=list(beta=rhc),
                       w=square(250),
                       types=c("A", "B"))

# Multitype Strauss hardcore with trends for each type:
beta <- c(0.27,0.08)
ri <- matrix(c(45,45,45,45),2,2)
rhc <- matrix(c(9.1,5.0,5.0,2.5),2,2)
tr3 <- function(x,y){x <- x/250; y <- y/250;
                      exp((6*x + 5*y - 18*x^2 + 12*x*y - 9*y^2)/6)
}
tr4 <- function(x,y){x <- x/250; y <- y/250;
                      exp(-0.6*x+0.5*y)}
mod10 <- rmhmodel(cif="straushm",par=list(beta=beta,gamma=gamma,
                                             iradii=ri,hradii=rhc),w=c(0,250,0,250),
                        trend=list(tr3,tr4))

# Triplets process:
mod11 <- rmhmodel(cif="triplets",par=list(beta=2,gamma=0.2,r=0.7),
                   w=c(0,10,0,10))

# Lookup (interaction function h_2 from page 76, Diggle (2003)):
r <- seq(from=0,to=0.2,length=101)[-1] # Drop 0.
h <- 20*(r-0.05)
h[r<0.05] <- 0
h[r>0.10] <- 1
mod17 <- rmhmodel(cif="lookup",par=list(beta=4000,h=h,r=r),w=c(0,1,0,1))

```

rmhmodel.list*Define Point Process Model for Metropolis-Hastings Simulation.***Description**

Given a list of parameters, builds a description of a point process model for use in simulating the model by the Metropolis-Hastings algorithm.

Usage

```
## S3 method for class 'list'
rmhmodel(model, ...)
```

Arguments

- | | |
|-------|---|
| model | A list of parameters. See Details. |
| ... | Optional list of additional named parameters. |

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.list` is the method for lists. The argument `model` should be a named list of parameters of the form

```
list(cif, par, w, trend, types)
```

where `cif` and `par` are required and the others are optional. For details about these components, see `rmhmodel.default`.

The subsequent arguments ... (if any) may also have these names, and they will take precedence over elements of the list `model`.

Value

An object of class "rmhmodel", which is essentially a validated 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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Diggle, P. J. (2003) *Statistical Analysis of Spatial Point Patterns* (2nd ed.) Arnold, London.
- Diggle, P.J. and Gratton, R.J. (1984) Monte Carlo methods of inference for implicit statistical models. *Journal of the Royal Statistical Society, series B* **46**, 193 – 212.
- 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.
- 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

`rmhmodel`, `rmhmodel.default`, `rmhmodel.ppm`, `rmh`, `rmhcontrol`, `rmhstart`, `ppm`, `Strauss`, `Softcore`, `StraussHard`, `MultiStrauss`, `MultiStraussHard`, `DiggleGratton`, `PairPiece`

Examples

```
# Strauss process:
mod01 <- list(cif="strauss",par=list(beta=2,gamma=0.2,r=0.7),
               w=c(0,10,0,10))
mod01 <- rmhmodel(mod01)

# Strauss with hardcore:
mod04 <- list(cif="straush",par=list(beta=2,gamma=0.2,r=0.7,hc=0.3),
               w=owin(c(0,10),c(0,5)))
mod04 <- rmhmodel(mod04)
```

```

# Soft core:
w    <- square(10)
mod07 <- list(cif="sftcr",
              par=list(beta=0.8,sigma=0.1,kappa=0.5),
              w=w)
mod07 <- rmhmodel(mod07)

# Multitype Strauss:
beta <- c(0.027,0.008)
gmma <- matrix(c(0.43,0.98,0.98,0.36),2,2)
r    <- matrix(c(45,45,45,45),2,2)
mod08 <- list(cif="straussm",
              par=list(beta=beta,gamma=gmma,radii=r),
              w=square(250))
mod08 <- rmhmodel(mod08)

# specify types
mod09 <- rmhmodel(list(cif="straussm",
                        par=list(beta=beta,gamma=gmma,radii=r),
                        w=square(250),
                        types=c("A", "B")))

# Multitype Strauss hardcore with trends for each type:
beta  <- c(0.27,0.08)
ri    <- matrix(c(45,45,45,45),2,2)
rhc   <- matrix(c(9.1,5.0,5.0,2.5),2,2)
tr3   <- function(x,y){x <- x/250; y <- y/250;
                      exp((6*x + 5*y - 18*x^2 + 12*x*y - 9*y^2)/6)
                      }
                      # log quadratic trend
tr4   <- function(x,y){x <- x/250; y <- y/250;
                      exp(-0.6*x+0.5*y)}
                      # log linear trend
mod10 <- list(cif="straushm",par=list(beta=beta,gamma=gmma,
                                         iradii=ri,hradii=rhc),w=c(0,250,0,250),
                           trend=list(tr3,tr4))
mod10 <- rmhmodel(mod10)

# Lookup (interaction function h_2 from page 76, Diggle (2003)):
r <- seq(from=0,to=0.2,length=101)[-1] # Drop 0.
h <- 20*(r-0.05)
h[r<0.05] <- 0
h[r>0.10] <- 1
mod17 <- list(cif="lookup",par=list(beta=4000,h=h,r=r),w=c(0,1,0,1))
mod17 <- rmhmodel(mod17)

```

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, win, ..., verbose=TRUE, project=TRUE,
          control=rmhcontrol())
```

Arguments

model	Fitted point process model (object of class "ppm").
win	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 .

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 `win` 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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Diggle, P. J. (2003) *Statistical Analysis of Spatial Point Patterns* (2nd ed.) Arnold, London.
- Diggle, P.J. and Gratton, R.J. (1984) Monte Carlo methods of inference for implicit statistical models. *Journal of the Royal Statistical Society, series B* **46**, 193 – 212.
- Geyer, C.J. (1999) Likelihood Inference for Spatial Point Processes. Chapter 3 in O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. Van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*, Chapman and Hall / CRC, Monographs on Statistics and Applied Probability, number 80. Pages 79–140.
- Kelly, F.P. and Ripley, B.D. (1976) On Strauss's model for clustering. *Biometrika* **63**, 357–360.

See Also

`rmhmodel`, `rmhmodel.list`, `rmhmodel.default`, `rmh`, `rmhcontrol`, `rmhstart`, `ppm`, `AreaInter`, `BadGey`, `DiggleGatesStibbard`, `DiggleGratton`, `Fiksel`, `Geyer`, `Hardcore`, `LennardJones`, `MultiStrauss`, `MultiStraussHard`, `PairPiece`, `Poisson`, `Softcore`, `Strauss`, `StraussHard`, `Triplets`

Examples

```
data(cells)
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
```

`rmhstart`

Determine Initial State for Metropolis-Hastings Simulation.

Description

Builds a description of the initial state for the Metropolis-Hastings algorithm.

Usage

```
rmhstart(start, ...)
## Default S3 method:
rmhstart(start=NULL, ..., n.start=NULL, x.start=NULL)
```

Arguments

- | | |
|----------------------|--|
| <code>start</code> | An existing description of the initial state in some format. Incompatible with the arguments listed below. |
| <code>...</code> | There should be no other arguments. |
| <code>n.start</code> | Number of initial points (to be randomly generated). Incompatible with <code>x.start</code> . |
| <code>x.start</code> | Initial point pattern configuration. Incompatible with <code>n.start</code> . |

Details

Simulated realisations of many point process models can be generated using the Metropolis-Hastings algorithm implemented in [rmh](#).

This function [rmhstart](#) creates a full description of the initial state of the Metropolis-Hastings algorithm, *including possibly the initial state of the random number generator*, for use in a subsequent call to [rmh](#). It also checks that the initial state is valid.

The initial state should be specified **either** by the first argument `start` **or** by the other arguments `n.start`, `x.start` etc.

If `start` is a list, then it should have components named `n.start` or `x.start`, with the same interpretation as described below.

The arguments are:

n.start The number of “initial” points to be randomly (uniformly) generated in the simulation window `w`. Incompatible with `x.start`.

For a multitype point process, `n.start` may be a vector (of length equal to the number of types) giving the number of points of each type to be generated.

If expansion of the simulation window is selected (see the argument `expand` to [rmhcontrol](#)), then the actual number of starting points in the simulation will be `n.start` multiplied by the expansion factor (ratio of the areas of the expanded window and original window).

For faster convergence of the Metropolis-Hastings algorithm, the value of `n.start` should be roughly equal to (an educated guess at) the expected number of points for the point process inside the window.

x.start Initial point pattern configuration. Incompatible with `n.start`.

`x.start` may be a point pattern (an object of class `ppp`), or an object which can be coerced to this class by [as.ppp](#), or a dataset containing vectors `x` and `y`.

If `x.start` is specified, then expansion of the simulation window (the argument `expand` of [rmhcontrol](#)) is not permitted.

The parameters `n.start` and `x.start` are *incompatible*.

Value

An object of class “`rmhstart`”, which is essentially a list of parameters describing the initial point pattern and (optionally) the initial state of the random number generator.

There is a `print` method for this class, which prints a sensible description of the initial state.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[rmh](#), [rmhcontrol](#), [rmhmodel](#)

Examples

```
# 30 random points
a <- rmhstart(n.start=30)

# a particular point pattern
```

```
data(cells)
b <- rmhstart(x.start=cells)
```

rMosaicField*Mosaic Random Field***Description**

Generate a realisation of a random field which is piecewise constant on the tiles of a given tessellation.

Usage

```
rMosaicField(X,
  rgen = function(n) { sample(0:1, n, replace = TRUE)},
  ...,
  rgenargs=NULL)
```

Arguments

X	A tessellation (object of class "tess").
...	Arguments passed to <code>as.mask</code> determining the pixel resolution.
rgen	Function that generates random values for the tiles of the tessellation.
rgenargs	List containing extra arguments that should be passed to rgen (typically specifying parameters of the distribution of the values).

Details

This function generates a realisation of a random field which is piecewise constant on the tiles of the given tessellation X. The values in each tile are independent and identically distributed.

Value

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

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`rpoislinetess`, `rMosaicSet`

Examples

```
X <- rpoislinetess(3)
plot(rMosaicField(X, runif))
plot(rMosaicField(X, runif, dimyx=256))
plot(rMosaicField(X, rnorm, rgenargs=list(mean=10, sd=2)))

plot(rMosaicField(dirichlet(runifpoint(30)), rnorm))
```

rMosaicSet*Mosaic Random Set***Description**

Generate a random set by taking a random selection of tiles of a given tessellation.

Usage

```
rMosaicSet(X, p=0.5)
```

Arguments

- | | |
|---|---|
| X | A tessellation (object of class "tess"). |
| p | Probability of including a given tile. A number strictly between 0 and 1. |

Details

Given a tessellation X, this function randomly selects some of the tiles of X, including each tile with probability p independently of the other tiles. The selected tiles are then combined to form a set in the plane.

One application of this is Switzer's (1965) example of a random set which has a Markov property. It is constructed by generating X according to a Poisson line tessellation (see [rpoislinetess](#)).

Value

A window (object of class "owin").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

Switzer, P. A random set process in the plane with a Markovian property. *Annals of Mathematical Statistics* **36** (1965) 1859–1863.

See Also

[rpoislinetess](#), [rMosaicField](#)

Examples

```
# Switzer's random set
X <- rpoislinetess(3)
plot(rMosaicSet(X, 0.5), col="green", border=NA)

# another example
plot(rMosaicSet(dirichlet(runifpoint(30)), 0.4))
```

rmpointGenerate N Random Multitype Points

Description

Generate a random multitype point pattern with a fixed number of points, or a fixed number of points of each type.

Usage

```
rmpoint(n, f=1, fmax=NULL, win=unit.square(),
       types, ptypes,
       ..., giveup=1000, verbose=FALSE)
```

Arguments

n	Number of marked points to generate. Either a single number specifying the total number of points, or a vector specifying the number of points of each type.
f	The probability density of the multitype points, usually un-normalised. Either a constant, a vector, a function $f(x, y, m, \dots)$, a pixel image, a list of functions $f(x, y, \dots)$ or a list of pixel images.
fmax	An upper bound on the values of f. If missing, this number will be estimated.
win	Window in which to simulate the pattern. Ignored if f is a pixel image or list of pixel images.
types	All the possible types for the multitype pattern.
ptypes	Optional vector of probabilities for each type.
...	Arguments passed to f if it is a function.
giveup	Number of attempts in the rejection method after which the algorithm should stop trying to generate new points.
verbose	Flag indicating whether to report details of performance of the simulation algorithm.

Details

This function generates random multitype point patterns consisting of a fixed number of points.

Three different models are available:

- I. Random location and type:** If n is a single number and the argument ptypes is missing, then n independent, identically distributed random multitype points are generated. Their locations $(x[i], y[i])$ and types $m[i]$ have joint probability density proportional to $f(x, y, m)$.
- II. Random type, and random location given type:** If n is a single number and ptypes is given, then n independent, identically distributed random multitype points are generated. Their types $m[i]$ have probability distribution ptypes. Given the types, the locations $(x[i], y[i])$ have conditional probability density proportional to $f(x, y, m)$.
- III. Fixed types, and random location given type:** If n is a vector, then we generate n[i] independent, identically distributed random points of type types[i]. For points of type m the conditional probability density of location (x, y) is proportional to $f(x, y, m)$.

Note that the density f is normalised in different ways in Model I and Models II and III. In Model I the normalised joint density is $g(x, y, m) = f(x, y, m)/Z$ where

$$Z = \sum_m \int \int \lambda(x, y, m) dx dy$$

while in Models II and III the normalised conditional density is $g(x, y \mid m) = f(x, y, m)/Z_m$ where

$$Z_m = \int \int \lambda(x, y, m) dx dy.$$

In Model I, the marginal distribution of types is $p_m = Z_m/Z$.

The unnormalised density f may be specified in any of the following ways.

single number: If f is a single number, the conditional density of location given type is uniform.

That is, the points of each type are uniformly distributed. In Model I, the marginal distribution of types is also uniform (all possible types have equal probability).

vector: If f is a numeric vector, the conditional density of location given type is uniform. That is, the points of each type are uniformly distributed. In Model I, the marginal distribution of types is proportional to the vector f . In Model II, the marginal distribution of types is `ptypes`, that is, the values in f are ignored.

function: If f is a function, it will be called in the form $f(x, y, m, \dots)$ at spatial location (x, y) for points of type m . In Model I, the joint probability density of location and type is proportional to $f(x, y, m, \dots)$. In Models II and III, the conditional probability density of location (x, y) given type m is proportional to $f(x, y, m, \dots)$. The function f must work correctly with vectors x , y and m , returning a vector of function values. (Note that m will be a factor with levels `types`.) The value `fmax` must be given and must be an upper bound on the values of $f(x, y, m, \dots)$ for all locations (x, y) inside the window `win` and all types m . The argument `types` must be given.

list of functions: If f is a list of functions, then the functions will be called in the form $f[[i]](x, y, \dots)$ at spatial location (x, y) for points of type `types[i]`. In Model I, the joint probability density of location and type is proportional to $f[[m]](x, y, \dots)$. In Models II and III, the conditional probability density of location (x, y) given type m is proportional to $f[[m]](x, y, \dots)$. The function $f[[i]]$ must work correctly with vectors x and y , returning a vector of function values. The value `fmax` must be given and must be an upper bound on the values of $f[[i]](x, y, \dots)$ for all locations (x, y) inside the window `win`. The argument `types` defaults to `seq(f)`.

pixel image: If f is a pixel image object of class "im" (see [im.object](#)), the unnormalised density at a location (x, y) for points of any type is equal to the pixel value of f for the pixel nearest to (x, y) . In Model I, the marginal distribution of types is uniform. The argument `win` is ignored; the window of the pixel image is used instead. The argument `types` must be given.

list of pixel images: If f is a list of pixel images, then the image $f[[i]]$ determines the density values of points of type `types[i]`. The argument `win` is ignored; the window of the pixel image is used instead. The argument `types` defaults to `factor(seq(f))`.

The implementation uses the rejection method. For Model I, `rmpoispp` is called repeatedly until n points have been generated. It gives up after `giveup` calls if there are still fewer than n points. For Model II, the types are first generated according to `ptypes`, then the locations of the points of each type are generated using `rpoint`. For Model III, the locations of the points of each type are generated using `rpoint`.

Value

The simulated point pattern (an object of class "ppp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppp.object](#), [owin.object](#)

Examples

```
abc <- c("a", "b", "c")

##### Model I

rmpoint(25, types=abc)
rmpoint(25, 1, types=abc)
# 25 points, equal probability for each type, uniformly distributed locations

rmpoint(25, function(x,y,m) {rep(1, length(x))}, types=abc)
# same as above
rmpoint(25, list(function(x,y){rep(1, length(x))},
                 function(x,y){rep(1, length(x))},
                 function(x,y){rep(1, length(x))}),
         types=abc)
# same as above

rmpoint(25, function(x,y,m) { x }, types=abc)
# 25 points, equal probability for each type,
# locations nonuniform with density proportional to x

rmpoint(25, function(x,y,m) { ifelse(m == "a", 1, x) }, types=abc)
rmpoint(25, list(function(x,y) { rep(1, length(x)) },
                 function(x,y) { x },
                 function(x,y) { x }),
         types=abc)
# 25 points, UNEQUAL probabilities for each type,
# type "a" points uniformly distributed,
# type "b" and "c" points nonuniformly distributed.

##### Model II

rmpoint(25, 1, types=abc, ptypes=rep(1,3)/3)
rmpoint(25, 1, types=abc, ptypes=rep(1,3))
# 25 points, equal probability for each type,
# uniformly distributed locations

rmpoint(25, function(x,y,m) {rep(1, length(x))}, types=abc, ptypes=rep(1,3))
# same as above
rmpoint(25, list(function(x,y){rep(1, length(x))},
                 function(x,y){rep(1, length(x))},
                 function(x,y){rep(1, length(x))}),
         types=abc, ptypes=rep(1,3))
# same as above

rmpoint(25, function(x,y,m) { x }, types=abc, ptypes=rep(1,3))
```

```

# 25 points, equal probability for each type,
# locations nonuniform with density proportional to x

rmpoint(25, function(x,y,m) { ifelse(m == "a", 1, x) }, types=abc, ptypes=rep(1,3))
# 25 points, EQUAL probabilities for each type,
# type "a" points uniformly distributed,
# type "b" and "c" points nonuniformly distributed.

##### Model III

rmpoint(c(12, 8, 4), 1, types=abc)
# 12 points of type "a",
# 8 points of type "b",
# 4 points of type "c",
# each uniformly distributed

rmpoint(c(12, 8, 4), function(x,y,m) { ifelse(m=="a", 1, x)}, types=abc)
rmpoint(c(12, 8, 4), list(function(x,y) { rep(1, length(x)) },
                           function(x,y) { x },
                           function(x,y) { x })),
                           types=abc)

# 12 points of type "a", uniformly distributed
# 8 points of type "b", nonuniform
# 4 points of type "c", nonuniform

#####

## Randomising an existing point pattern:
data(demopat)
X <- demopat

# same numbers of points of each type, uniform random locations (Model III)
rmpoint(table(X$marks), 1, types=levels(X$marks), win=X>window)

# same total number of points, distribution of types estimated from X,
# uniform random locations (Model II)
rmpoint(X$n, 1, types=levels(X$marks), win=X>window,
        ptypes=table(X$marks))

```

Description

Generate a random point pattern, a realisation of the (homogeneous or inhomogeneous) multitype Poisson process.

Usage

```
rmpoispp(lambda, lmax=NULL, win, types, ...)
```

Arguments

lambda	Intensity of the multitype Poisson process. Either a single positive number, a vector, a function(x, y, m, \dots), a pixel image, a list of functions function(x, y, \dots), or a list of pixel images.
lmax	An upper bound for the value of lambda. May be omitted
win	Window in which to simulate the pattern. An object of class "owin" or something acceptable to as.owin . Ignored if lambda is a pixel image or list of images.
types	All the possible types for the multitype pattern.
...	Arguments passed to lambda if it is a function.

Details

This function generates a realisation of the marked Poisson point process with intensity lambda.

Note that the intensity function $\lambda(x, y, m)$ is the average number of points of type **m** per unit area near the location (x, y) . Thus a marked point process with a constant intensity of 10 and three possible types will have an average of 30 points per unit area, with 10 points of each type on average.

The intensity function may be specified in any of the following ways.

single number: If lambda is a single number, then this algorithm generates a realisation of the uniform marked Poisson process inside the window win with intensity lambda for each type. The total intensity of points of all types is lambda * length(types). The argument types must be given and determines the possible types in the multitype pattern.

vector: If lambda is a numeric vector, then this algorithm generates a realisation of the stationary marked Poisson process inside the window win with intensity lambda[i] for points of type types[i]. The total intensity of points of all types is sum(lambda). The argument types defaults to seq(lambda).

function: If lambda is a function, the process has intensity lambda(x, y, m, \dots) at spatial location (x, y) for points of type m . The function lambda must work correctly with vectors x , y and m , returning a vector of function values. (Note that m will be a factor with levels equal to types.) The value lmax, if present, must be an upper bound on the values of lambda(x, y, m, \dots) for all locations (x, y) inside the window win and all types m . The argument types must be given.

list of functions: If lambda is a list of functions, the process has intensity lambda[[i]](x, y, \dots) at spatial location (x, y) for points of type types[i]. The function lambda[[i]] must work correctly with vectors x and y , returning a vector of function values. The value lmax, if given, must be an upper bound on the values of lambda(x, y, \dots) for all locations (x, y) inside the window win. The argument types defaults to seq(lambda).

pixel image: If lambda is a pixel image object of class "im" (see [im.object](#)), the intensity at a location (x, y) for points of any type is equal to the pixel value of lambda for the pixel nearest to (x, y) . The argument win is ignored; the window of the pixel image is used instead. The argument types must be given.

list of pixel images: If lambda is a list of pixel images, then the image lambda[[i]] determines the intensity of points of type types[i]. The argument win is ignored; the window of the pixel image is used instead. The argument types defaults to seq(lambda).

If lmax is missing, an approximate upper bound will be calculated.

To generate an inhomogeneous Poisson process the algorithm uses “thinning”: it first generates a uniform Poisson process of intensity lmax for points of each type m , then randomly deletes or retains each point independently, with retention probability $p(x, y, m) = \lambda(x, y, m)/lmax$.

Value

The simulated multitype point pattern (an object of class "ppp" with a component `marks` which is a factor).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`rpoispp` for unmarked Poisson point process; `rmpoint` for a fixed number of random marked points; `ppp.object`, `owin.object`.

Examples

```
# uniform bivariate Poisson process with total intensity 100 in unit square
pp <- rmpoispp(50, types=c("a","b"))

# stationary bivariate Poisson process with intensity A = 30, B = 70
pp <- rmpoispp(c(30,70), types=c("A","B"))
pp <- rmpoispp(c(30,70))

# works in any window
data(letterR)
pp <- rmpoispp(c(30,70), win=letterR, types=c("A","B"))

# inhomogeneous lambda(x,y,m)
# note argument 'm' is a factor
lam <- function(x,y,m) { 50 * (x^2 + y^3) * ifelse(m=="A", 2, 1)}
pp <- rmpoispp(lam, win=letterR, types=c("A","B"))
# extra arguments
lam <- function(x,y,m,scal) { scal * (x^2 + y^3) * ifelse(m=="A", 2, 1)}
pp <- rmpoispp(lam, win=letterR, types=c("A","B"), scal=50)

# list of functions lambda[[i]](x,y)
lams <- list(function(x,y){50 * x^2}, function(x,y){20 * abs(y)})
pp <- rmpoispp(lams, win=letterR, types=c("A","B"))
pp <- rmpoispp(lams, win=letterR)
# functions with extra arguments
lams <- list(function(x,y,scal){5 * scal * x^2},
             function(x,y, scal){2 * scal * abs(y)})
pp <- rmpoispp(lams, win=letterR, types=c("A","B"), scal=10)
pp <- rmpoispp(lams, win=letterR, scal=10)

# florid example
lams <- list(function(x,y){
    100*exp((6*x + 5*y - 18*x^2 + 12*x*y - 9*y^2)/6)
    }
    # log quadratic trend
    ,
    function(x,y){
        100*exp(-0.6*x+0.5*y)
        }
    # log linear trend
    )
}
```

```
X <- rmpoispp(lams, win=unit.square(), types=c("on", "off"))

# pixel image
Z <- as.im(function(x,y){30 * (x^2 + y^3)}, letterR)
pp <- rmpoispp(Z, types=c("A","B"))

# list of pixel images
ZZ <- list(
  as.im(function(x,y){20 * (x^2 + y^3)}, letterR),
  as.im(function(x,y){40 * (x^3 + y^2)}, letterR))
pp <- rmpoispp(ZZ, types=c("A","B"))
pp <- rmpoispp(ZZ)
```

rNeymanScott*Simulate Neyman-Scott Process***Description**

Generate a random point pattern, a realisation of the Neyman-Scott cluster process.

Usage

```
rNeymanScott(kappa, rmax, rcluster, win = owin(c(0,1),c(0,1)), ..., lmax=NULL)
```

Arguments

<code>kappa</code>	Intensity of the Poisson process of cluster centres. A single positive number, a function, or a pixel image.
<code>rmax</code>	Maximum radius of a random cluster.
<code>rcluster</code>	A function which generates random clusters, or other data specifying the random cluster mechanism. See Details.
<code>win</code>	Window in which to simulate the pattern. An object of class "owin" or something acceptable to as.owin .
<code>...</code>	Arguments passed to <code>rcluster</code>
<code>lmax</code>	Optional. Upper bound on the values of <code>kappa</code> when <code>kappa</code> is a function or pixel image.

Details

This algorithm generates a realisation of the general Neyman-Scott process, with the cluster mechanism given by the function `rcluster`. The clusters must have a finite maximum possible radius `rmax`.

First, the algorithm generates a Poisson point process of “parent” points with intensity `kappa`. Here `kappa` may be a single positive number, a function `kappa(x, y)`, or a pixel image object of class “im” (see [im.object](#)). See [rpoispp](#) for details.

Second, each parent point is replaced by a random cluster of points. These clusters are combined together to yield a single point pattern which is then returned as the result of `rNeymanScott`.

The argument `rcluster` specifies the cluster mechanism. It may be either:

- A function which will be called to generate each random cluster (the offspring points of each parent point). The function should expect to be called in the form `rcluster(x0,y0,...)` for a parent point at a location (x_0, y_0) . The return value of `rcluster` should specify the coordinates of the points in the cluster; it may be a list containing elements `x`, `y`, or a point pattern (object of class "ppp"). If it is a marked point pattern then the result of `rNeymanScott` will be a marked point pattern.
- A list(`mu`, `f`) where `mu` specifies the mean number of offspring points in each cluster, and `f` generates the random displacements (vectors pointing from the parent to the offspring). In this case, the number of offspring in a cluster is assumed to have a Poisson distribution, implying that the Neyman-Scott process is also a Cox process. The first element `mu` should be either a single nonnegative number (interpreted as the mean of the Poisson distribution of cluster size) or a pixel image or a function(`x, y`) giving a spatially varying mean cluster size (interpreted in the sense of Waagepetersen, 2007). The second element `f` should be a function that will be called once in the form `f(n)` to generate `n` independent and identically distributed displacement vectors (i.e. as if there were a cluster of size `n` with a parent at the origin $(0, 0)$). The function should return a point pattern (object of class "ppp") or something acceptable to `xy.coords` that specifies the coordinates of `n` points.

If required, the intermediate stages of the simulation (the parents and the individual clusters) can also be extracted from the return value of `rNeymanScott` through the attributes "parents" and "parentid". The attribute "parents" is the point pattern of parent points. The attribute "parentid" is an integer vector specifying the parent for each of the points in the simulated pattern.

Value

The simulated point pattern (an object of class "ppp").

Additionally, some intermediate results of the simulation are returned as attributes of this point pattern: see Details.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Neyman, J. and Scott, E.L. (1958) A statistical approach to problems of cosmology. *Journal of the Royal Statistical Society, Series B* **20**, 1–43.
- Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman-Scott processes. *Biometrics* **63**, 252–258.

See Also

[rpoispp](#), [rThomas](#), [rGaussPoisson](#), [rMatClust](#), [rCauchy](#), [rVarGamma](#)

Examples

```
# each cluster consist of 10 points in a disc of radius 0.2
nclust <- function(x0, y0, radius, n) {
  return(runifdisc(n, radius, centre=c(x0, y0)))
}
plot(rNeymanScott(10, 0.2, nclust, radius=0.2, n=5))

# multitype Neyman-Scott process (each cluster is a multitype process)
```

```
nclust2 <- function(x0, y0, radius, n, types=c("a", "b")) {  
  X <- runifdisc(n, radius, centre=c(x0, y0))  
  M <- sample(types, n, replace=TRUE)  
  marks(X) <- M  
  return(X)  
}  
plot(rNeymanScott(15, 0.1, nclust2, radius=0.1, n=5))
```

rotate**Rotate**

Description

Applies a rotation to any two-dimensional object, such as a point pattern or a window.

Usage

```
rotate(X, ...)
```

Arguments

- X Any suitable dataset representing a two-dimensional object, such as a point pattern (object of class "ppp"), or a window (object of class "owin").
- ... Data specifying the rotation.

Details

This is generic. Methods are provided for point patterns ([rotate.ppp](#)) and windows ([rotate.owin](#)).

Value

Another object of the same type, representing the result of rotating X through the specified angle.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[rotate.ppp](#), [rotate.owin](#)

rotate.owin*Rotate a Window*

Description

Rotates a window

Usage

```
## S3 method for class 'owin'
rotate(X, angle=pi/2, ..., rescue=TRUE)
```

Arguments

- | | |
|--------|--|
| X | A window (object of class "owin"). |
| angle | Angle of rotation. |
| rescue | Logical. If TRUE, the rotated window will be processed by rescue.rectangle . |
| ... | Optional arguments passed to as.mask controlling the resolution of the rotated window, if X is a binary pixel mask. Ignored if X is not a binary mask. |

Details

Rotates the window by the specified angle. Angles are measured in radians, anticlockwise. The default is to rotate the window 90 degrees anticlockwise. The centre of rotation is the origin.

Value

Another object of class "owin" representing the rotated window.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[owin.object](#)

Examples

```
w <- owin(c(0,1),c(0,1))
v <- rotate(w, pi/3)
## Not run:
plot(v)

## End(Not run)
data(letterR)
w <- as.mask(letterR)
v <- rotate(w, pi/5)
```

rotate.hpp *Rotate a Point Pattern*

Description

Rotates a point pattern

Usage

```
## S3 method for class 'ppp'  
rotate(X, angle=pi/2, ...)
```

Arguments

X	A point pattern (object of class "ppp").
angle	Angle of rotation.
...	Arguments passed to <code>rotate.owin</code> affecting the handling of the observation window, if it is a binary pixel mask.

Details

The points of the pattern, and the window of observation, are rotated about the origin by the angle specified. Angles are measured in radians, anticlockwise. The default is to rotate the pattern 90 degrees anticlockwise. If the points carry marks, these are preserved.

Value

Another object of class "ppp" representing the rotated point pattern.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppp.object](#), [rotate.owin](#)

Examples

```
data(cells)  
X <- rotate(cells, pi/3)  
## Not run:  
plot(X)  
## End(Not run)
```

rotate.psp*Rotate a Line Segment Pattern***Description**

Rotates a line segment pattern

Usage

```
## S3 method for class 'psp'
rotate(X, angle=pi/2, ...)
```

Arguments

- | | |
|-------|--|
| X | A line segment pattern (object of class "psp"). |
| angle | Angle of rotation. |
| ... | Arguments passed to <code>rotate.owin</code> affecting the handling of the observation window, if it is a binary pixel mask. |

Details

The line segments of the pattern, and the window of observation, are rotated about the origin by the angle specified. Angles are measured in radians, anticlockwise. The default is to rotate the pattern 90 degrees anticlockwise. If the line segments carry marks, these are preserved.

Value

Another object of class "psp" representing the rotated line segment pattern.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`psp.object`, `rotate.owin`, `rotate.ppp`

Examples

```
oldpar <- par(mfrow=c(2,1))
X <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
plot(X, main="original")
Y <- rotate(X, pi/4)
plot(Y, main="rotated")
par(oldpar)
```

rpoint	<i>Generate N Random Points</i>
--------	---------------------------------

Description

Generate a random point pattern containing n independent, identically distributed random points with any specified distribution.

Usage

```
rpoint(n, f, fmax=NULL, win=unit.square(), ..., giveup=1000, verbose=FALSE)
```

Arguments

n	Number of points to generate.
f	The probability density of the points, possibly un-normalised. Either a constant, a function $f(x, y, \dots)$, or a pixel image object.
fmax	An upper bound on the values of f. If missing, this number will be estimated.
win	Window in which to simulate the pattern. Ignored if f is a pixel image.
...	Arguments passed to the function f.
giveup	Number of attempts in the rejection method after which the algorithm should stop trying to generate new points.
verbose	Flag indicating whether to report details of performance of the simulation algorithm.

Details

This function generates n independent, identically distributed random points with common probability density proportional to f.

The argument f may be

a numerical constant: uniformly distributed random points will be generated.

a function: random points will be generated in the window win with probability density proportional to $f(x, y, \dots)$ where x and y are the cartesian coordinates. The function f must accept two *vectors* of coordinates x, y and return the corresponding vector of function values. Additional arguments ... of any kind may be passed to the function.

a pixel image: if f is a pixel image object of class "im" (see [im.object](#)) then random points will be generated in the window of this pixel image, with probability density proportional to the pixel values of f.

The algorithm is as follows:

- If f is a constant, we invoke [runifpoint](#).
- If f is a function, then we use the rejection method. Proposal points are generated from the uniform distribution. A proposal point (x, y) is accepted with probability $f(x, y, \dots)/f\text{max}$ and otherwise rejected. The algorithm continues until n points have been accepted. It gives up after giveup * n proposals if there are still fewer than n points.
- If f is a pixel image, then a random sequence of pixels is selected (using [sample](#)) with probabilities proportional to the pixel values of f. Then for each pixel in the sequence we generate a uniformly distributed random point in that pixel.

The algorithm for pixel images is more efficient than that for functions.

Value

The simulated point pattern (an object of class "ppp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppp.object](#), [owin.object](#), [runifpoint](#)

Examples

```
# 100 uniform random points in the unit square
X <- rpoint(100)

# 100 random points with probability density proportional to x^2 + y^2
X <- rpoint(100, function(x,y) { x^2 + y^2}, 1)

# 'fmax' may be omitted
X <- rpoint(100, function(x,y) { x^2 + y^2})

# irregular window
data(letterR)
X <- rpoint(100, function(x,y) { x^2 + y^2}, win=letterR)

# make a pixel image
Z <- setcov(letterR)
# 100 points with density proportional to pixel values
X <- rpoint(100, Z)
```

rpoisline

Generate Poisson Random Line Process

Description

Generate a random pattern of line segments obtained from the Poisson line process.

Usage

```
rpoisline(lambda, win=owin())
```

Arguments

- | | |
|--------|--|
| lambda | Intensity of the Poisson line process. A positive number. |
| win | Window in which to simulate the pattern. An object of class "owin" or something acceptable to as.owin . Currently, the window must be a rectangle. |

Details

This algorithm generates a realisation of the uniform Poisson line process, and clips it to the window `win`.

The argument `lambda` must be a positive number. It controls the intensity of the process. The expected number of lines intersecting a convex region of the plane is equal to `lambda` times the perimeter length of the region. The expected total length of the lines crossing a region of the plane is equal to `lambda * pi` times the area of the region.

Value

A line segment pattern (an object of class "psp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[psp](#)

Examples

```
# uniform Poisson line process with intensity 10,
# clipped to the unit square
rpoisline(10)
```

rpoislinetess

Poisson Line Tessellation

Description

Generate a tessellation delineated by the lines of the Poisson line process

Usage

```
rpoislinetess(lambda, win = owin())
```

Arguments

<code>lambda</code>	Intensity of the Poisson line process. A positive number.
<code>win</code>	Window in which to simulate the pattern. An object of class "owin" or something acceptable to as.owin . Currently, the window must be a rectangle.

Details

This algorithm generates a realisation of the uniform Poisson line process, and divides the window `win` into tiles separated by these lines.

The argument `lambda` must be a positive number. It controls the intensity of the process. The expected number of lines intersecting a convex region of the plane is equal to `lambda` times the perimeter length of the region. The expected total length of the lines crossing a region of the plane is equal to `lambda * pi` times the area of the region.

Value

A tessellation (object of class "tess").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[rpoisline](#) to generate the lines only.

Examples

```
X <- rpoislinetess(3)
plot(as.im(X), main="rpoislinetess(3)")
plot(X, add=TRUE)
```

rpoislpp

Poisson Point Process on a Linear Network

Description

Generates a realisation of the Poisson point process with specified intensity on the given linear network.

Usage

```
rpoislpp(lambda, L, ...)
```

Arguments

lambda	Intensity of the Poisson process. A single number, a function(x,y), or a pixel image (object of class "im").
L	A linear network (object of class "linnet", see linnet).
...	Arguments passed to rpoisppOnLines .

Details

This function uses [rpoisppOnLines](#) to generate the random points.

Value

A point pattern on the linear network, i.e.\ an object of class "lpp".

Author(s)

Ang Qi Wei <aqw07398@hotmail.com> and Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

See Also

[runiflpp](#), [lpp](#), [linnet](#)

Examples

```
data(simplenet)
X <- rpoislpp(5, simplenet)
plot(X)
```

rpoispp

Generate Poisson Point Pattern

Description

Generate a random point pattern using the (homogeneous or inhomogeneous) Poisson process. Includes CSR (complete spatial randomness).

Usage

```
rpoispp(lambda, lmax, win, ...)
```

Arguments

<code>lambda</code>	Intensity of the Poisson process. Either a single positive number, a <code>function(x, y, ...)</code> , or a pixel image.
<code>lmax</code>	An upper bound for the value of <code>lambda(x, y)</code> , if <code>lambda</code> is a function.
<code>win</code>	Window in which to simulate the pattern. An object of class "owin" or something acceptable to as.owin . Ignored if <code>lambda</code> is a pixel image.
<code>...</code>	Arguments passed to <code>lambda</code> if it is a function.

Details

If `lambda` is a single number, then this algorithm generates a realisation of the uniform Poisson process (also known as Complete Spatial Randomness, CSR) inside the window `win` with intensity `lambda` (points per unit area).

If `lambda` is a function, then this algorithm generates a realisation of the inhomogeneous Poisson process with intensity function `lambda(x, y, ...)` at spatial location (x, y) inside the window `win`. The function `lambda` must work correctly with vectors `x` and `y`. The value `lmax` must be given and must be an upper bound on the values of `lambda(x, y, ...)` for all locations (x, y) inside the window `win`.

If `lambda` is a pixel image object of class "im" (see [im.object](#)), this algorithm generates a realisation of the inhomogeneous Poisson process with intensity equal to the pixel values of the image. (The value of the intensity function at an arbitrary location is the pixel value of the nearest pixel.) The argument `win` is ignored; the window of the pixel image is used instead.

To generate an inhomogeneous Poisson process the algorithm uses “thinning”: it first generates a uniform Poisson process of intensity `lmax`, then randomly deletes or retains each point, independently of other points, with retention probability $p(x, y) = \lambda(x, y)/lmax$.

For *marked* point patterns, use [rmpoispp](#).

Value

The simulated point pattern (an object of class "ppp").

Warning

Note that `lambda` is the **intensity**, that is, the expected number of points **per unit area**. The total number of points in the simulated pattern will be random with expected value `mu = lambda * a` where `a` is the area of the window `win`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`rmpoispp` for Poisson *marked* point patterns, `runifpoint` for a fixed number of independent uniform random points; `rpoint`, `rmpoint` for a fixed number of independent random points with any distribution; `rMaternI`, `rMaternII`, `rSSI`, `rStrauss`, `rstrat` for random point processes with spatial inhibition or regularity; `rThomas`, `rGaussPoisson`, `rMatClust`, `rcell` for random point processes exhibiting clustering; `rmh.default` for Gibbs processes. See also `ppp.object`, `owin.object`.

Examples

```
# uniform Poisson process with intensity 100 in the unit square
pp <- rpoispp(100)

# uniform Poisson process with intensity 1 in a 10 x 10 square
pp <- rpoispp(1, win=owin(c(0,10),c(0,10)))
# plots should look similar !

# inhomogeneous Poisson process in unit square
# with intensity lambda(x,y) = 100 * exp(-3*x)
# Intensity is bounded by 100
pp <- rpoispp(function(x,y) {100 * exp(-3*x)}, 100)

# How to tune the coefficient of x
lamb <- function(x,y,a) { 100 * exp( - a * x)}
pp <- rpoispp(lamb, 100, a=3)

# pixel image
Z <- as.im(function(x,y){100 * sqrt(x+y)}, unit.square())
pp <- rpoispp(Z)
```

Description

Generate a random three-dimensional point pattern using the homogeneous Poisson process.

Usage

```
rpoispp3(lambda, domain = box3())
```

Arguments

lambda	Intensity of the Poisson process. A single positive number.
domain	Three-dimensional box in which the process should be generated. An object of class "box3".

Details

This function generates a realisation of the homogeneous Poisson process in three dimensions, with intensity `lambda` (points per unit volume).

The realisation is generated inside the three-dimensional region `domain` which currently must be a rectangular box (object of class "box3").

Value

The simulated three-dimensional point pattern (an object of class "pp3").

Note

The intensity `lambda` is the expected number of points *per unit volume*.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[runifpoint3](#), [pp3](#), [box3](#)

Examples

```
X <- rpoispp3(50)
```

rpoisppOnLines

Generate Poisson Point Pattern on Line Segments

Description

Given a line segment pattern, generate a Poisson random point pattern on the line segments.

Usage

```
rpoisppOnLines(lambda, L, lmax = NULL, ...)
```

Arguments

<code>lambda</code>	Intensity of the Poisson process. A single number, a <code>function(x,y)</code> , or a pixel image (object of class "im").
<code>L</code>	Line segment pattern (object of class "psp") on which the points should be generated.
<code>lmax</code>	Maximum possible value of <code>lambda</code> if it is a function or a pixel image.
<code>...</code>	Additional arguments passed to <code>lambda</code> if it is a function.

Details

This command generates a Poisson point process on the one-dimensional system of line segments in `L`. The result is a point pattern consisting of points lying on the line segments in `L`. The number of random points falling on any given line segment follows a Poisson distribution. The patterns of points on different segments are independent.

The intensity `lambda` is the expected number of points per unit **length** of line segment. It may be constant, or it may depend on spatial location.

The argument `lambda` may be a single number, or a `function(x,y)`, or a pixel image (object of class "im"). In the two latter cases, the rejection method is used.

The rejection method requires knowledge of `lmax`, the maximum possible value of `lambda`. If `lmax` is not given, it will be computed approximately, by sampling many values of `lambda`.

Value

A point pattern (object of class "ppp") in the same window as `L`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`psp`, `ppp`, `runifpointOnLines`, `rpoispp`

Examples

```

L <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
plot(L, main="")

# uniform intensity
Y <- rpoisppOnLines(4, L)
plot(Y, add=TRUE, pch="+")

# intensity is a function
Y <- rpoisppOnLines(function(x,y){ 10 * x^2}, L, 10)
plot(L, main="")
plot(Y, add=TRUE, pch="+")

# intensity is an image
Z <- as.im(function(x,y){10 * sqrt(x+y)}, unit.square())
Y <- rpoisppOnLines(Z, L, 10)
plot(L, main="")
plot(Y, add=TRUE, pch="+")

```

rpoisppx*Generate Poisson Point Pattern in Any Dimensions*

Description

Generate a random multi-dimensional point pattern using the homogeneous Poisson process.

Usage

```
rpoisppx(lambda, domain)
```

Arguments

lambda	Intensity of the Poisson process. A single positive number.
domain	Multi-dimensional box in which the process should be generated. An object of class "boxx".

Details

This function generates a realisation of the homogeneous Poisson process in multi dimensions, with intensity `lambda` (points per unit volume).

The realisation is generated inside the multi-dimensional region `domain` which currently must be a rectangular box (object of class "boxx").

Value

The simulated multi-dimensional point pattern (an object of class "ppx").

Note

The intensity `lambda` is the expected number of points *per unit volume*.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[runifpointx](#), [ppx](#), [boxx](#)

Examples

```
w <- boxx(x=c(0,1), y=c(0,1), z=c(0,1), t=c(0,3))
X <- rpoisppx(10, w)
```

rPoissonCluster*Simulate Poisson Cluster Process***Description**

Generate a random point pattern, a realisation of the general Poisson cluster process.

Usage

```
rPoissonCluster(kappa, rmax, rcluster, win = owin(c(0,1),c(0,1)), ..., lmax=NULL)
```

Arguments

<code>kappa</code>	Intensity of the Poisson process of cluster centres. A single positive number, a function, or a pixel image.
<code>rmax</code>	Maximum radius of a random cluster.
<code>rcluster</code>	A function which generates random clusters.
<code>win</code>	Window in which to simulate the pattern. An object of class "owin" or something acceptable to as.owin .
<code>...</code>	Arguments passed to <code>rcluster</code>
<code>lmax</code>	Optional. Upper bound on the values of <code>kappa</code> when <code>kappa</code> is a function or pixel image.

Details

This algorithm generates a realisation of the general Poisson cluster process, with the cluster mechanism given by the function `rcluster`. The clusters must have a finite maximum possible radius `rmax`.

First, the algorithm generates a Poisson point process of “parent” points with intensity `kappa`. Here `kappa` may be a single positive number, a function `kappa(x, y)`, or a pixel image object of class “im” (see [im.object](#)). See [rpoispp](#) for details.

Second, each parent point is replaced by a random cluster of points, created by calling the function `rcluster`. These clusters are combined together to yield a single point pattern which is then returned as the result of `rPoissonCluster`.

The function `rcluster` should expect to be called as `rcluster(xp[i], yp[i], ...)` for each parent point at a location `(xp[i], yp[i])`. The return value of `rcluster` should be a list with elements `x, y` which are vectors of equal length giving the absolute `x` and `y` coordinates of the points in the cluster.

If the return value of `rcluster` is a point pattern (object of class “ppp”) then it may have marks. The result of `rPoissonCluster` will then be a marked point pattern.

If required, the intermediate stages of the simulation (the parents and the individual clusters) can also be extracted from the return value of `rPoissonCluster` through the attributes “parents” and “parentid”. The attribute “parents” is the point pattern of parent points. The attribute “parentid” is an integer vector specifying the parent for each of the points in the simulated pattern.

Value

The simulated point pattern (an object of class "ppp").

Additionally, some intermediate results of the simulation are returned as attributes of this point pattern: see Details.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[rpoispp](#), [rThomas](#), [rGaussPoisson](#), [rMatClust](#)

Examples

```
# each cluster consist of 10 points in a disc of radius 0.2
nclust <- function(x0, y0, radius, n) {
    return(runifdisc(n, radius, centre=c(x0, y0)))
}
plot(rPoissonCluster(10, 0.2, nclust, radius=0.2, n=5))

# multitype Neyman-Scott process (each cluster is a multitype process)
nclust2 <- function(x0, y0, radius, n, types=c("a", "b")) {
    X <- runifdisc(n, radius, centre=c(x0, y0))
    M <- sample(types, n, replace=TRUE)
    marks(X) <- M
    return(X)
}
plot(rPoissonCluster(15, 0.1, nclust2, radius=0.1, n=5))
```

Description

Randomly shifts the points of a point pattern or line segment pattern. Generic.

Usage

`rshift(X, ...)`

Arguments

- | | |
|------------------|--|
| <code>X</code> | Pattern to be subjected to a random shift. A point pattern (class "ppp"), a line segment pattern (class "psp") or an object of class "splitppp". |
| <code>...</code> | Arguments controlling the generation of the random shift vector, or specifying which parts of the pattern will be shifted. |

Details

This operation applies a random shift (vector displacement) to the points in a point pattern, or to the segments in a line segment pattern.

The argument *X* may be

- a point pattern (an object of class "ppp")
- a line segment pattern (an object of class "psp")
- an object of class "splitppp" (basically a list of point patterns, obtained from [split.ppp](#)).

The function *rshift* is generic, with methods for the three classes "ppp", "psp" and "splitppp".

See the help pages for these methods, [rshift.ppp](#), [rshift.psp](#) and [rshift.splitppp](#), for further information.

Value

An object of the same type as *X*.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[rshift.ppp](#), [rshift.psp](#), [rshift.splitppp](#)

rshift.ppp

Randomly Shift a Point Pattern

Description

Randomly shifts the points of a point pattern.

Usage

```
## S3 method for class 'ppp'
rshift(X, ..., which=NULL, group)
```

Arguments

- | | |
|--------------|--|
| <i>X</i> | Point pattern to be subjected to a random shift. An object of class "ppp" |
| ... | Arguments that determine the random shift. See Details. |
| <i>group</i> | Optional. Factor specifying a grouping of the points of <i>X</i> , or NULL indicating that all points belong to the same group. Each group will be shifted together, and separately from other groups. By default, points in a marked point pattern are grouped according to their mark values, while points in an unmarked point pattern are treated as a single group. |
| <i>which</i> | Optional. Identifies which groups of the pattern will be shifted, while other groups are not shifted. A vector of levels of <i>group</i> . |

Details

This operation randomly shifts the locations of the points in a point pattern.

The function `rshift` is generic. This function `rshift.hpp` is the method for point patterns.

The most common use of this function is to shift the points in a multitype point pattern. By default, points of the same type are shifted in parallel (i.e. points of a common type are shifted by a common displacement vector), and independently of other types. This is useful for testing the hypothesis of independence of types (the null hypothesis that the sub-patterns of points of each type are independent point processes).

In general the points of `X` are divided into groups, then the points within a group are shifted by a common random displacement vector. Different groups of points are shifted independently. The grouping is determined as follows:

- If the argument `group` is present, then this determines the grouping.
- Otherwise, if `X` is a multitype point pattern, the marks determine the grouping.
- Otherwise, all points belong to a single group.

The argument `group` should be a factor, of length equal to the number of points in `X`. Alternatively `group` may be `NULL`, which specifies that all points of `X` belong to a single group.

By default, every group of points will be shifted. The argument `which` indicates that only some of the groups should be shifted, while other groups should be left unchanged. `which` must be a vector of levels of `group` (for example, a vector of types in a multitype pattern) indicating which groups are to be shifted.

The displacement vector, i.e. the vector by which the data points are shifted, is generated at random. Parameters that control the randomisation and the handling of edge effects are passed through the `...` argument. They are

radius, width, height Parameters of the random shift vector.

edge String indicating how to deal with edges of the pattern. Options are "torus", "erode" and "none".

clip Optional. Window to which the final point pattern should be clipped.

If the window is a rectangle, the *default* behaviour is to generate a displacement vector at random with equal probability for all possible displacements. This means that the *x* and *y* coordinates of the displacement vector are independent random variables, uniformly distributed over the range of possible coordinates.

Alternatively, the displacement vector can be generated by another random mechanism, controlled by the arguments `radius`, `width` and `height`.

rectangular: if `width` and `height` are given, then the displacement vector is uniformly distributed in a rectangle of these dimensions, centred at the origin. The maximum possible displacement in the *x* direction is `width/2`. The maximum possible displacement in the *y* direction is `height/2`. The *x* and *y* displacements are independent. (If `width` and `height` are actually equal to the dimensions of the observation window, then this is equivalent to the default.)

radial: if `radius` is given, then the displacement vector is generated by choosing a random point inside a disc of the given radius, centred at the origin, with uniform probability density over the disc. Thus the argument `radius` determines the maximum possible displacement distance. The argument `radius` is incompatible with the arguments `width` and `height`.

The argument `edge` controls what happens when a shifted point lies outside the window of `X`. Options are:

"none": Points shifted outside the window of X simply disappear.

"torus": Toroidal or periodic boundary. Treat opposite edges of the window as identical, so that a point which disappears off the right-hand edge will re-appear at the left-hand edge. This is called a "toroidal shift" because it makes the rectangle topologically equivalent to the surface of a torus (doughnut).

The window must be a rectangle. Toroidal shifts are undefined if the window is non-rectangular.

"erode": Clip the point pattern to a smaller window.

If the random displacements are generated by a radial mechanism (see above), then the window of X is eroded by a distance equal to the value of the argument `radius`, using `erosion`.

If the random displacements are generated by a rectangular mechanism, then the window of X is (if it is not rectangular) eroded by a distance `max(height, width)` using `erosion`; or (if it is rectangular) trimmed by a margin of width `width` at the left and right sides and trimmed by a margin of height `height` at the top and bottom.

The rationale for this is that the clipping window is the largest window for which edge effects can be ignored.

The optional argument `clip` specifies a smaller window to which the pattern should be restricted.

Value

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

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[rshift](#), [rshift.psp](#)

Examples

```
data(amacrine)

# random toroidal shift
# shift "on" and "off" points separately
X <- rshift(amacrine)

# shift "on" points and leave "off" points fixed
X <- rshift(amacrine, which="on")

# shift all points simultaneously
X <- rshift(amacrine, group=NULL)

# maximum displacement distance 0.1 units
X <- rshift(amacrine, radius=0.1)

# shift with erosion
X <- rshift(amacrine, radius=0.1, edge="erode")
```

rshift.pspRandomly Shift a Line Segment Pattern

Description

Randomly shifts the segments in a line segment pattern.

Usage

```
## S3 method for class 'psp'
rshift(X, ..., group=NULL, which=NULL)
```

Arguments

X	Line segment pattern to be subjected to a random shift. An object of class "psp".
...	Arguments controlling the randomisation and the handling of edge effects. See rshift.ppp .
group	Optional. Factor specifying a grouping of the line segments of X, or NULL indicating that all line segments belong to the same group. Each group will be shifted together, and separately from other groups.
which	Optional. Identifies which groups of the pattern will be shifted, while other groups are not shifted. A vector of levels of group.

Details

This operation randomly shifts the locations of the line segments in a line segment pattern.

The function `rshift` is generic. This function `rshift.psp` is the method for line segment patterns.

The line segments of X are first divided into groups, then the line segments within a group are shifted by a common random displacement vector. Different groups of line segments are shifted independently. If the argument `group` is present, then this determines the grouping. Otherwise, all line segments belong to a single group.

The argument `group` should be a factor, of length equal to the number of line segments in X. Alternatively `group` may be NULL, which specifies that all line segments of X belong to a single group.

By default, every group of line segments will be shifted. The argument `which` indicates that only some of the groups should be shifted, while other groups should be left unchanged. `which` must be a vector of levels of `group` indicating which groups are to be shifted.

The displacement vector, i.e. the vector by which the data line segments are shifted, is generated at random. The *default* behaviour is to generate a displacement vector at random with equal probability for all possible displacements. This means that the *x* and *y* coordinates of the displacement vector are independent random variables, uniformly distributed over the range of possible coordinates.

Alternatively, the displacement vector can be generated by another random mechanism, controlled by the arguments `radius`, `width` and `height`.

rectangular: if `width` and `height` are given, then the displacement vector is uniformly distributed in a rectangle of these dimensions, centred at the origin. The maximum possible displacement in the *x* direction is `width/2`. The maximum possible displacement in the *y* direction is `height/2`. The *x* and *y* displacements are independent. (If `width` and `height` are actually equal to the dimensions of the observation window, then this is equivalent to the default.)

radial: if radius is given, then the displacement vector is generated by choosing a random line segment inside a disc of the given radius, centred at the origin, with uniform probability density over the disc. Thus the argument radius determines the maximum possible displacement distance. The argument radius is incompatible with the arguments width and height.

The argument edge controls what happens when a shifted line segment lies partially or completely outside the window of X. Currently the only option is "erode" which specifies that the segments will be clipped to a smaller window.

The optional argument clip specifies a smaller window to which the pattern should be restricted.

Value

A line segment pattern (object of class "psp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[rshift](#), [rshift.ppp](#)

Examples

```
X <- psp(runif(20), runif(20), runif(20), runif(20), window=owin())
Y <- rshift(X, radius=0.1)
```

[rshift.splitppp](#)

Randomly Shift a List of Point Patterns

Description

Randomly shifts each point pattern in a list of point patterns.

Usage

```
## S3 method for class 'splitppp'
rshift(X, ..., which=seq_along(X))
```

Arguments

- X An object of class "splitppp". Basically a list of point patterns.
- ... Parameters controlling the generation of the random shift vector and the handling of edge effects. See [rshift.ppp](#).
- which Optional. Identifies which patterns will be shifted, while other patterns are not shifted. Any valid subset index for X.

Details

This operation applies a random shift to each of the point patterns in the list X.

The function `rshift` is generic. This function `rshift.splitppp` is the method for objects of class "splitppp", which are essentially lists of point patterns, created by the function `split.ppp`.

By default, every pattern in the list X will be shifted. The argument `which` indicates that only some of the patterns should be shifted, while other groups should be left unchanged. `which` can be any valid subset index for X.

Each point pattern in the list X (or each pattern in X[`which`]) is shifted by a random displacement vector. The shifting is performed by `rshift.ppp`.

See the help page for `rshift.ppp` for details of the other arguments.

Value

Another object of class "splitppp".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`rshift, rshift.ppp`

Examples

```
data(amacrine)
Y <- split(amacrine)

# random toroidal shift
# shift "on" and "off" points separately
X <- rshift(Y)

# shift "on" points and leave "off" points fixed
X <- rshift(Y, which="on")

# maximum displacement distance 0.1 units
X <- rshift(Y, radius=0.1)

# shift with erosion
X <- rshift(Y, radius=0.1, edge="erode")
```

Description

Generate a random point pattern, a realisation of the Simple Sequential Inhibition (SSI) process.

Usage

```
rSSI(r, n, win = owin(c(0,1),c(0,1)), giveup = 1000, x.init=NULL)
```

Arguments

r	Inhibition distance.
n	Number of points to generate.
win	Window in which to simulate the pattern. An object of class "owin" or something acceptable to as.owin .
giveup	Number of rejected proposals after which the algorithm should terminate.
x.init	Optional. Initial configuration of points. A point pattern (object of class "ppp").

Details

This algorithm generates a realisation of the Simple Sequential Inhibition point process inside the window `win`.

Starting with an empty window (or with the point pattern `x.init` if specified), the algorithm adds points one-by-one. Each new point is generated uniformly in the window and independently of preceding points. If the new point lies closer than `r` units from an existing point, then it is rejected and another random point is generated.

The algorithm terminates either when the desired number `n` of points is reached, or when the current point configuration has not changed for `giveup` iterations.

Value

The simulated point pattern (an object of class "ppp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[rpoispp](#), [rMaternI](#), [rMaternII](#).

Examples

```
pp <- rSSI(0.05, 200)
## Not run: plot(pp)
```

rstrat

Simulate Stratified Random Point Pattern

Description

Generates a “stratified random” pattern of points in a window, by dividing the window into rectangular tiles and placing `k` random points independently in each tile.

Usage

```
rstrat(win=square(1), nx, ny=nx, k = 1)
```

Arguments

<code>win</code>	A window. An object of class <code>owin</code> , or data in any format acceptable to <code>as.owin()</code> .
<code>nx</code>	Number of tiles in each column.
<code>ny</code>	Number of tiles in each row.
<code>k</code>	Number of random points to generate in each tile.

Details

This function generates a random pattern of points in a “stratified random” sampling design. It can be useful for generating random spatial sampling points.

The bounding rectangle of `win` is divided into a regular $nx \times ny$ grid of rectangular tiles. In each tile, `k` random points are generated independently with a uniform distribution in that tile.

Some of these grid points may lie outside the window `win`: if they do, they are deleted.

The result is a point pattern inside the window `win`.

This function is useful in creating dummy points for quadrature schemes (see `quadscheme`) as well as in simulating random point patterns.

Value

A point pattern (object of class “`ppp`”).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`rsyst`, `runifpoint`, `quadscheme`

Examples

```
X <- rstrat(nx=10)
plot(X)

# polygonal boundary
data(letterR)
X <- rstrat(letterR, 5, 10, k=3)
plot(X)
```

Description

Generate a random pattern of points, a simulated realisation of the Strauss process, using a perfect simulation algorithm.

Usage

```
rStrauss(beta, gamma = 1, R = 0, W = owin())
```

Arguments

beta	intensity parameter (a positive number).
gamma	interaction parameter (a number between 0 and 1, inclusive).
R	interaction radius (a non-negative number).
W	window (object of class "owin") in which to generate the random pattern. Currently this must be a rectangular window.

Details

This function generates a realisation of the Strauss point process in the window W using a ‘perfect simulation’ algorithm.

The Strauss process (Strauss, 1975; Kelly and Ripley, 1976) is a model for spatial inhibition, ranging from a strong ‘hard core’ inhibition to a completely random pattern according to the value of γ .

The Strauss process with interaction radius R and parameters β and γ is the pairwise interaction point process with probability density

$$f(x_1, \dots, x_n) = \alpha \beta^{n(x)} \gamma^{s(x)}$$

where x_1, \dots, 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. Intuitively, each point of the pattern contributes a factor β to the probability density, and each pair of points closer than r units apart contributes a factor γ to the density.

The interaction parameter γ must be less than or equal to 1 in order that the process be well-defined (Kelly and Ripley, 1976). This model describes an “ordered” or “inhibitive” pattern. If $\gamma = 1$ it reduces to a Poisson process (complete spatial randomness) with intensity β . If $\gamma = 0$ it is called a “hard core process” with hard core radius $R/2$, since no pair of points is permitted to lie closer than R units apart.

The simulation algorithm used to generate the point pattern is ‘dominated coupling from the past’ as implemented by Berthelsen and Moller (2002, 2003). This is a ‘perfect simulation’ or ‘exact simulation’ algorithm, so called because the output of the algorithm is guaranteed to have the correct probability distribution exactly (unlike the Metropolis-Hastings algorithm used in [rmh](#), whose output is only approximately correct).

There is a tiny chance that the algorithm will run out of space before it has terminated. If this occurs, an error message will be generated.

Value

A point pattern (object of class “ppp”).

Author(s)

Kasper Klitgaard Berthelsen, adapted for **spatstat** by Adrian Baddeley <Adrian.Baddeley@csiro.au>
<http://www.maths.uwa.edu.au/~adrian/>

References

- Berthelsen, K.K. and Moller, J. (2002) A primer on perfect simulation for spatial point processes. *Bulletin of the Brazilian Mathematical Society* 33, 351-367.
- Berthelsen, K.K. and Moller, J. (2003) Likelihood and non-parametric Bayesian MCMC inference for spatial point processes based on perfect simulation and path sampling. *Scandinavian Journal of Statistics* 30, 549-564.
- Kelly, F.P. and Ripley, B.D. (1976) On Strauss's model for clustering. *Biometrika* 63, 357-360.
- Moller, J. and Waagepetersen, R. (2003). *Statistical Inference and Simulation for Spatial Point Processes*. Chapman and Hall/CRC.
- Strauss, D.J. (1975) A model for clustering. *Biometrika* 63, 467-475.

See Also

[rmh](#), [Strauss](#), [rHardcore](#), [rStraussHard](#), [rDiggleGratton](#), [rDGS](#).

Examples

```
X <- rStrauss(0.05, 0.2, 1.5, square(141.4))
Z <- rStrauss(100, 0.7, 0.05)
```

rStraussHard

Perfect Simulation of the Strauss-Hardcore Process

Description

Generate a random pattern of points, a simulated realisation of the Strauss-Hardcore process, using a perfect simulation algorithm.

Usage

```
rStraussHard(beta, gamma = 1, R = 0, H = 0, W = owin())
```

Arguments

- | | |
|-------|---|
| beta | intensity parameter (a positive number). |
| gamma | interaction parameter (a number between 0 and 1, inclusive). |
| R | interaction radius (a non-negative number). |
| H | hard core distance (a non-negative number smaller than R). |
| W | window (object of class "owin") in which to generate the random pattern. Currently this must be a rectangular window. |

Details

This function generates a realisation of the Strauss-Hardcore point process in the window \mathbb{W} using a ‘perfect simulation’ algorithm.

The Strauss-Hardcore process is described in [StraussHard](#).

The simulation algorithm used to generate the point pattern is ‘dominated coupling from the past’ as implemented by Berthelsen and Moller (2002, 2003). This is a ‘perfect simulation’ or ‘exact simulation’ algorithm, so called because the output of the algorithm is guaranteed to have the correct probability distribution exactly (unlike the Metropolis-Hastings algorithm used in [rmh](#), whose output is only approximately correct).

A limitation of the perfect simulation algorithm is that the interaction parameter γ must be less than or equal to 1. To simulate a Strauss-hardcore process with $\gamma > 1$, use [rmh](#).

There is a tiny chance that the algorithm will run out of space before it has terminated. If this occurs, an error message will be generated.

Value

A point pattern (object of class “`ppp`”).

Author(s)

Kasper Klitgaard Berthelsen and Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

References

- Berthelsen, K.K. and Moller, J. (2002) A primer on perfect simulation for spatial point processes. *Bulletin of the Brazilian Mathematical Society* 33, 351-367.
- Berthelsen, K.K. and Moller, J. (2003) Likelihood and non-parametric Bayesian MCMC inference for spatial point processes based on perfect simulation and path sampling. *Scandinavian Journal of Statistics* 30, 549-564.
- Moller, J. and Waagepetersen, R. (2003). *Statistical Inference and Simulation for Spatial Point Processes*. Chapman and Hall/CRC.

See Also

[rmh](#), [rStrauss](#), [StraussHard](#).

Examples

```
Z <- rStraussHard(100, 0.7, 0.05, 0.02)
```

rsyst*Simulate systematic random point pattern*

Description

Generates a “systematic random” pattern of points in a window, consisting of a grid of equally-spaced points with a random common displacement.

Usage

```
rsyst(win=square(1), nx, ny=nx, dx=NULL, dy=NULL)
```

Arguments

win	A window. An object of class owin , or data in any format acceptable to as.owin() .
dx	Spacing of grid points in x direction. Incompatible with nx.
dy	Spacing of grid points in y direction. Incompatible with ny.
nx	Number of columns of grid points in the window. Incompatible with dx.
ny	Number of rows of grid points in the window. Incompatible with dy.

Details

This function generates a “systematic random” pattern of points in the window `win`. The pattern consists of a rectangular grid of points with a random common displacement.

The grid spacing is determined by the distances `dx`, `dy` if they are present. If they are absent, then the grid spacing is determined so that there will be `nx` columns and `ny` rows of grid points in the bounding rectangle of `win`.

The grid is then given a random displacement (the common displacement of the grid points is a uniformly distributed random vector in the tile of dimensions `dx`, `dy`).

Some of the resulting grid points may lie outside the window `win`: if they do, they are deleted. The result is a point pattern inside the window `win`.

This function is useful in creating dummy points for quadrature schemes (see [quadscheme](#)) as well as in simulating random point patterns.

Value

A point pattern (object of class “`ppp`”).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[rstrat](#), [runifpoint](#), [quadscheme](#)

Examples

```
X <- rsys(t(nx=10)
plot(X)

# polygonal boundary
data(letterR)
X <- rsys(t(letterR, 5, 10)
plot(X)
```

rthin

Random Thinning

Description

Applies independent random thinning to a point pattern.

Usage

```
rthin(X, P, ...)
```

Arguments

- | | |
|------------------|--|
| <code>X</code> | A point pattern (object of class "ppp") that will be thinned. |
| <code>P</code> | Data giving the retention probabilities, i.e. the probability that each point in <code>X</code> will be retained. Either a single number, or a vector of numbers, or a function(<code>x, y</code>), or a pixel image (object of class "im"). |
| <code>...</code> | Additional arguments passed to <code>P</code> , if it is a function. |

Details

In a random thinning operation, each point of the pattern `X` is randomly either deleted or retained (i.e. not deleted). The result is a point pattern, consisting of those points of `X` that were retained.

Independent random thinning means that the retention/deletion of each point is independent of other points.

The argument `P` determines the probability of **retaining** each point. It may be

- a **single number**, so that each point will be retained with the same probability `P`;
- a **vector of numbers**, so that the *i*th point of `X` will be retained with probability `P[i]`;
- a **function** `P(x, y)`, so that a point at a location (x, y) will be retained with probability `P(x, y)`;
- a **pixel image**, containing values of the retention probability for all locations in a region encompassing the point pattern.

If `P` is a function, it should be ‘vectorised’, that is, it should accept vector arguments `x, y` and should yield a numeric vector of the same length. The function may have extra arguments which are passed through the `...` argument.

Value

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

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

Examples

```
data(redwood)
plot(redwood, main="thinning")

# delete 20% of points
Y <- rthin(redwood, 0.8)
points(Y, col="green", cex=1.4)

# function
f <- function(x,y) { ifelse(x < 0.4, 1, 0.5) }
Y <- rthin(redwood, f)

# pixel image
Z <- as.im(f, redwood$window)
Y <- rthin(redwood, Z)
```

rThomas

*Simulate Thomas Process***Description**

Generate a random point pattern, a realisation of the Thomas cluster process.

Usage

```
rThomas(kappa, sigma, mu, win = owin(c(0,1),c(0,1)))
```

Arguments

<code>kappa</code>	Intensity of the Poisson process of cluster centres. A single positive number.
<code>sigma</code>	Standard deviation of displacement of a point from its cluster centre.
<code>mu</code>	Expected number of points per cluster.
<code>win</code>	Window in which to simulate the pattern. An object of class "owin" or something acceptable to as.owin .

Details

This algorithm generates a realisation of the Thomas process, a special case of the Neyman-Scott process.

The algorithm generates a uniform Poisson point process of “parent” points with intensity `kappa`. Then each parent point is replaced by a random cluster of points, the number of points per cluster being Poisson (`mu`) distributed, and their positions being isotropic Gaussian displacements from the cluster parent location.

This classical model can be fitted to data by the method of minimum contrast, using [thomas.estK](#) or [kppm](#).

The algorithm can also generate spatially inhomogeneous versions of the Thomas process:

- The parent points can be spatially inhomogeneous. If the argument `kappa` is a function(x, y) or a pixel image (object of class "im"), then it is taken as specifying the intensity function of an inhomogeneous Poisson process that generates the parent points.
- The offspring points can be inhomogeneous. If the argument `mu` is a function(x, y) or a pixel image (object of class "im"), then it is interpreted as the reference density for offspring points, in the sense of Waagepetersen (2006). For a given parent point, the offspring constitute a Poisson process with intensity function equal to $\mu(x, y) * f(x, y)$ where f is the Gaussian density centred at the parent point.

When the parents are homogeneous (`kappa` is a single number) and the offspring are inhomogeneous (`mu` is a function or pixel image), the model can be fitted to data using `kppm`, or using `thomas.estK` applied to the inhomogeneous K function.

Value

The simulated point pattern (an object of class "ppp").

Additionally, some intermediate results of the simulation are returned as attributes of this point pattern. See `rNeymanScott`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

Waagepetersen, R. (2006) An estimating function approach to inference for inhomogeneous Neyman–Scott processes. Submitted for publication.

See Also

`rpoispp`, `rMatClust`, `rGaussPoisson`, `rNeymanScott`, `thomas.estK`, `kppm`

Examples

```
#homogeneous
X <- rThomas(10, 0.2, 5)
#inhomogeneous
Z <- as.im(function(x,y){ 5 * exp(2 * x - 1) }, owin())
Y <- rThomas(10, 0.2, Z)
```

runifdisc

Generate N Uniform Random Points in a Disc

Description

Generate a random point pattern containing n independent uniform random points in a circular disc.

Usage

```
runifdisc(n, radius=1, centre=c(0,0), ...)
```

Arguments

n	Number of points.
radius	Radius of the circle.
centre	Coordinates of the centre of the circle.
...	Arguments passed to disc controlling the accuracy of approximation to the circle.

Details

This function generates n independent random points, uniformly distributed in a circular disc.

It is faster (for a circular window) than the general code used in [runifpoint](#).

To generate random points in an ellipse, first generate points in a circle using [runifdisc](#), then transform to an ellipse using [affine](#), as shown in the examples.

To generate random points in other windows, use [runifpoint](#). To generate non-uniform random points, use [rpoint](#).

Value

The simulated point pattern (an object of class "ppp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[disc](#), [runifpoint](#), [rpoint](#)

Examples

```
# 100 random points in the unit disc
plot(runifdisc(100))
# 42 random points in the ellipse with major axis 3 and minor axis 1
X <- runifdisc(42)
Y <- affine(X, mat=diag(c(3,1)))
plot(Y)
```

Description

Generates n random points, independently and uniformly distributed, on a linear network.

Usage

`runiflpp(n, L)`

Arguments

- n** Number of random points to generate. A nonnegative integer.
L A linear network (object of class "linnet", see [linnet](#)).

Details

This function uses [runifpointOnLines](#) to generate the random points.

Value

A point pattern on the linear network, i.e.\ an object of class "lpp".

Author(s)

Ang Qi Wei <aqw07398@hotmail.com> and Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

See Also

[rpoislpp](#), [lpp](#), [linnet](#)

Examples

```
data(simplenet)
X <- runiflpp(10, simplenet)
plot(X)
```

runifpoint

Generate N Uniform Random Points

Description

Generate a random point pattern containing n independent uniform random points.

Usage

```
runifpoint(n, win=owin(c(0,1),c(0,1)), giveup=1000)
```

Arguments

- n** Number of points.
win Window in which to simulate the pattern. An object of class "owin" or something acceptable to [as.owin](#).
giveup Number of attempts in the rejection method after which the algorithm should stop trying to generate new points.

Details

This function generates n independent random points, uniformly distributed in the window win . (For nonuniform distributions, see [rpoint](#).)

The algorithm depends on the type of window, as follows:

- If win is a rectangle then n independent random points, uniformly distributed in the rectangle, are generated by assigning uniform random values to their cartesian coordinates.
- If win is a binary image mask, then a random sequence of pixels is selected (using [sample](#)) with equal probabilities. Then for each pixel in the sequence we generate a uniformly distributed random point in that pixel.
- If win is a polygonal window, the algorithm uses the rejection method. It finds a rectangle enclosing the window, generates points in this rectangle, and tests whether they fall in the desired window. It gives up when $\text{giveup} * n$ tests have been performed without yielding n successes.

The algorithm for binary image masks is faster than the rejection method but involves discretisation.

Value

The simulated point pattern (an object of class "ppp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppp.object](#), [owin.object](#), [rpoispp](#), [rpoint](#)

Examples

```
# 100 random points in the unit square
pp <- runifpoint(100)
# irregular window
data(letterR)
# polygonal
pp <- runifpoint(100, letterR)
# binary image mask
pp <- runifpoint(100, as.mask(letterR))
```

runifpoint3

Generate N Uniform Random Points in Three Dimensions

Description

Generate a random point pattern containing n independent, uniform random points in three dimensions.

Usage

`runifpoint3(n, domain = box3())`

Arguments

- n** Number of points to be generated.
domain Three-dimensional box in which the process should be generated. An object of class "box3".

Details

This function generates n independent random points, uniformly distributed in the three-dimensional box domain.

Value

The simulated point pattern (an object of class "pp3").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[rpoispp3](#), [pp3](#), [box3](#)

Examples

```
X <- runifpoint3(50)
```

<code>runifpointOnLines</code>	<i>Generate N Uniform Random Points On Line Segments</i>
--------------------------------	--

Description

Given a line segment pattern, generate a random point pattern consisting of n points uniformly distributed on the line segments.

Usage

```
runifpointOnLines(n, L)
```

Arguments

- n** Number of points to generate.
L Line segment pattern (object of class "psp") on which the points should lie.

Details

This command generates a point pattern consisting of n independent random points, each point uniformly distributed on the line segment pattern. This means that, for each random point,

- the probability of falling on a particular segment is proportional to the length of the segment; and
- given that the point falls on a particular segment, it has uniform probability density along that segment.

Value

A point pattern (object of class "ppp") with the same window as L.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[psp](#), [ppp](#), [pointsOnLines](#), [runifpoint](#)

Examples

```
X <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
Y <- runifpointOnLines(20, X)
plot(X, main="")
plot(Y, add=TRUE)
```

runifpointx

Generate N Uniform Random Points in Any Dimensions

Description

Generate a random point pattern containing n independent, uniform random points in any number of spatial dimensions.

Usage

```
runifpointx(n, domain)
```

Arguments

- | | |
|--------|--|
| n | Number of points to be generated. |
| domain | Multi-dimensional box in which the process should be generated. An object of class "boxx". |

Details

This function generates n independent random points, uniformly distributed in the multi-dimensional box domain.

Value

The simulated point pattern (an object of class "ppx").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[rpoisppx](#), [ppx](#), [boxx](#)

Examples

```
w <- boxx(x=c(0,1), y=c(0,1), z=c(0,1), t=c(0,3))
X <- runifpointx(50, w)
```

rVarGamma

Simulate Neyman-Scott Point Process with Variance Gamma cluster kernel

Description

Generate a random point pattern, a simulated realisation of the Neyman-Scott process with Variance Gamma (Bessel) cluster kernel.

Usage

```
rVarGamma(kappa, nu.ker, omega, mu, win = owin(), eps = 0.001)
```

Arguments

kappa	Intensity of the Poisson process of cluster centres. A single positive number, a function, or a pixel image.
nu.ker	Shape parameter for the cluster kernel. A number greater than -1.
omega	Scale parameter for cluster kernel. Determines the size of clusters. A positive number in the same units as the spatial coordinates.
mu	Mean number of points per cluster (a single positive number) or reference intensity for the cluster points (a function or a pixel image).
win	Window in which to simulate the pattern. An object of class "owin" or something acceptable to as.owin .
eps	Threshold below which the values of the cluster kernel will be treated as zero for simulation purposes.

Details

This algorithm generates a realisation of the Neyman-Scott process with Variance Gamma (Bessel) cluster kernel, inside the window `win`.

The process is constructed by first generating a Poisson point process of “parent” points with intensity `kappa`. Then each parent point is replaced by a random cluster of points, the number of points in each cluster being random with a Poisson (`mu`) distribution, and the points being placed independently and uniformly according to a Variance Gamma kernel.

In this implementation, parent points are not restricted to lie in the window; the parent process is effectively the uniform Poisson process on the infinite plane.

This model can be fitted to data by the method of minimum contrast, using [cauchy.estK](#), [cauchy.estpcf](#) or [kppm](#).

The algorithm can also generate spatially inhomogeneous versions of the cluster process:

- The parent points can be spatially inhomogeneous. If the argument `kappa` is a function(`x,y`) or a pixel image (object of class "`im`"), then it is taken as specifying the intensity function of an inhomogeneous Poisson process that generates the parent points.
- The offspring points can be inhomogeneous. If the argument `mu` is a function(`x,y`) or a pixel image (object of class "`im`"), then it is interpreted as the reference density for offspring points, in the sense of Waagepetersen (2006).

When the parents are homogeneous (`kappa` is a single number) and the offspring are inhomogeneous (`mu` is a function or pixel image), the model can be fitted to data using `kppm`, or using `cauchy.estK` or `cauchy.estpcf` applied to the inhomogeneous K function.

Value

The simulated point pattern (an object of class "`ppp`").

Additionally, some intermediate results of the simulation are returned as attributes of this point pattern. See `rNeymanScott`.

Author(s)

Abdollah Jalilian and Rasmus Waagepetersen. Adapted for `spatstat` by Adrian Baddeley <Adrian.Baddeley@csiro.au>
<http://www.maths.uwa.edu.au/~adrian/>

References

- Jalilian, A., Guan, Y. and Waagepetersen, R. (2011) Decomposition of variance for spatial Cox processes. Manuscript submitted for publication.
- Waagepetersen, R. (2007) An estimating function approach to inference for inhomogeneous Neyman–Scott processes. *Biometrics* **63**, 252–258.

See Also

`rpoispp`, `rNeymanScott`, `cauchy.estK`, `cauchy.estpcf`, `kppm`.

Examples

```
# homogeneous
X <- rVarGamma(30, 2, 0.02, 5)
# inhomogeneous
Z <- as.im(function(x,y){ exp(2 - 3 * x) }, W= owin())
Y <- rVarGamma(30, 2, 0.02, Z)
```

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

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

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz> in collaboration with Hao Wang and Jeff Picka

See Also

[ppm](#), [pairsat.family](#), [Geyer](#), [PairPiece](#), [BadGey](#).

Examples

```
SatPiece(c(0.1,0.2), c(1,1))
# prints a sensible description of itself
SatPiece(c(0.1,0.2), 1)
data(cells)
ppm(cells, ~1, SatPiece(c(0.07, 0.1, 0.13), 2))
# fit a stationary piecewise constant Saturated pairwise interaction process

## Not run:
ppm(cells, ~polynom(x,y,3), SatPiece(c(0.07, 0.1, 0.13), 2))
# nonstationary process with log-cubic polynomial trend

## End(Not run)
```

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppm](#), [pairsat.family](#), [Geyer](#), [SatPiece](#), [ppm.object](#)

scalardilate *Apply Scalar Dilation*

Description

Applie scalar dilation to a plane geometrical object, such as a point pattern or a window, relative to a specified origin.

Usage

```
scalardilate(X, f, ...)

## S3 method for class 'owin'
scalardilate(X, f, ..., origin=NULL)

## S3 method for class 'ppp'
scalardilate(X, f, ..., origin=NULL)

## Default S3 method:
scalardilate(X, f, ...)
```

Arguments

X	Any suitable dataset representing a two-dimensional object, such as a point pattern (object of class "ppp"), or a window (object of class "owin").
f	Scalar dilation factor. A finite number greater than zero.
...	Ignored by the methods.
origin	Origin for the scalar dilation. Either a vector of 2 numbers, or one of the character strings "centroid", "midpoint" or "bottomleft" (partially matched).

Details

This command performs scalar dilation of the object X by the factor f relative to the origin specified by `origin`.

The function `scalardilate` is generic, with methods for windows (class "owin") and point patterns (class "ppp") and a default method.

If the argument `origin` is not given, then every spatial coordinate is multiplied by the factor f.

If `origin` is given, then scalar dilation is performed relative to the specified origin. Effectively, X is shifted so that `origin` is moved to `c(0,0)`, then scalar dilation is performed, then the result is shifted so that `c(0,0)` is moved to `origin`.

This command is a special case of an affine transformation: see [affine](#).

Value

Another object of the same type, representing the result of applying the scalar dilation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also[affine](#), [shift](#)**Examples**

```
plot(letterR)
plot(scalardilate(letterR, 0.7, origin="bot"), col="red", add=TRUE)
```

scaletointerval*Rescale Data to Lie Between Specified Limits*

Description

Rescales a dataset so that the values range exactly between the specified limits.

Usage

```
scaletointerval(x, from=0, to=1)
## Default S3 method:
scaletointerval(x, from=0, to=1)
## S3 method for class 'im'
scaletointerval(x, from=0, to=1)
```

Arguments

- | | |
|-----------------------|---|
| <code>x</code> | Data to be rescaled. |
| <code>from, to</code> | Lower and upper endpoints of the interval to which the values of <code>x</code> should be rescaled. |

Details

These functions rescale a dataset `x` so that its values range exactly between the limits `from` and `to`. The method for pixel images (objects of class "im") applies this scaling to the pixel values of `x`. Rescaling cannot be performed if the values in `x` are not interpretable as numeric, or if the values in `x` are all equal.

Value

An object of the same type as `x`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also[scale](#)

Examples

```
X <- as.im(function(x,y) {x+y+3}, unit.square())
summary(X)
Y <- scaletointerval(X)
summary(Y)
```

`scan.test`

Spatial Scan Test

Description

Performs the Spatial Scan Test for clustering in a spatial point pattern, or for clustering of one type of point in a bivariate spatial point pattern.

Usage

```
scan.test(X, r, ...,
          method = c("poisson", "binomial"),
          nsim = 19,
          baseline = NULL,
          case = 2,
          alternative = c("greater", "less", "two.sided"),
          verbose = TRUE)
```

Arguments

<code>X</code>	A point pattern (object of class "ppp").
<code>r</code>	Radius of circle to use. A single number.
<code>...</code>	Optional. Arguments passed to <code>as.mask</code> to determine the spatial resolution of the computations.
<code>method</code>	Either "poisson" or "binomial" specifying the type of likelihood.
<code>nsim</code>	Number of simulations for computing Monte Carlo p-value.
<code>baseline</code>	Baseline for the Poisson intensity, if <code>method="poisson"</code> . A pixel image or a function.
<code>case</code>	Which type of point should be interpreted as a case, if <code>method="binomial"</code> . Integer or character string.
<code>alternative</code>	Alternative hypothesis: "greater" if the alternative postulates that the mean number of points inside the circle will be greater than expected under the null.
<code>verbose</code>	Logical. Whether to print progress reports.

Details

The spatial scan test (Kulldorf, 1997) is applied to the point pattern `X`.

In a nutshell,

- If `method="poisson"` then a significant result would mean that there is a circle of radius `r`, located somewhere in the spatial domain of the data, which contains a significantly higher than expected number of points of `X`. That is, the pattern `X` exhibits spatial clustering.

- If `method="binomial"` then X must be a bivariate (two-type) point pattern. By default, the first type of point is interpreted as a control (non-event) and the second type of point as a case (event). A significant result would mean that there is a circle of radius r which contains a significantly higher than expected number of cases. That is, the cases are clustered together, conditional on the locations of all points.

Following is a more detailed explanation.

- If `method="poisson"` then the scan test based on Poisson likelihood is performed (Kulldorf, 1997). The dataset X is treated as an unmarked point pattern. By default (if `baseline` is not specified) the null hypothesis is complete spatial randomness CSR (i.e. a uniform Poisson process). The alternative hypothesis is a Poisson process with one intensity β_1 inside some circle of radius r and another intensity β_0 outside the circle. If `baseline` is given, then it should be a pixel image or a function(x, y). The null hypothesis is an inhomogeneous Poisson process with intensity proportional to `baseline`. The alternative hypothesis is an inhomogeneous Poisson process with intensity `beta1 * baseline` inside some circle of radius r , and `beta0 * baseline` outside the circle.
- If `method="binomial"` then the scan test based on binomial likelihood is performed (Kulldorf, 1997). The dataset X must be a bivariate point pattern, i.e. a multitype point pattern with two types. The null hypothesis is that all permutations of the type labels are equally likely. The alternative hypothesis is that some circle of radius r has a higher proportion of points of the second type, than expected under the null hypothesis.

The result of `scan.test` is a hypothesis test (object of class "htest") which can be plotted to report the results. The component `p.value` contains the p -value.

The result of `scan.test` can also be plotted (using the `plot` method for the class "scan.test"). The plot is a pixel image of the Likelihood Ratio Test Statistic (2 times the log likelihood ratio) as a function of the location of the centre of the circle. This pixel image can be extracted from the object using `as.im`.

Value

An object of class "htest" (hypothesis test) which also belongs to the class "scan.test". Printing this object gives the result of the test. Plotting this object displays the Likelihood Ratio Test Statistic as a function of the location of the centre of the circle.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

Kulldorff, M. (1997) A spatial scan statistic. *Communications in Statistics — Theory and Methods* **26**, 1481–1496.

See Also

[relrisk](#)

Examples

```
nsim <- if(interactive()) 19 else 2
data(redwood)
scan.test(redwood, 0.1, method="poisson", nsim=nsim)
data(chorley)
scan.test(chorley, 1, method="binomial", case="larynx", nsim=nsim)
```

scanpp

Read Point Pattern From Data File

Description

Reads a point pattern dataset from a text file.

Usage

```
scanpp(filename, window, header=TRUE, dir="", multitype=FALSE)
```

Arguments

filename	String name of the file containing the coordinates of the points in the point pattern, and their marks if any.
window	Window for the point pattern. An object of class "owin".
header	Logical flag indicating whether the first line of the file contains headings for the columns. Passed to read.table .
dir	String containing the path name of the directory in which filename is to be found. Default is the current directory.
multitype	Logical flag indicating whether marks are to be interpreted as a factor (multitype = TRUE) or as numerical values (multitype = FALSE).

Details

This simple function reads a point pattern dataset from a file containing the cartesian coordinates of its points, and optionally the mark values for these points.

The file identified by filename in directory dir should be a text file that can be read using [read.table](#). Thus, each line of the file (except possibly the first line) contains data for one point in the point pattern. Data are arranged in columns. There should be either two columns (for an unmarked point pattern) or three columns (for a marked point pattern).

If header=FALSE then the first two columns of data will be interpreted as the *x* and *y* coordinates of points. A third column, if present, will be interpreted as containing the marks for these points. The marks will be converted to a factor if multitype = TRUE.

If header=TRUE then the first line of the file should contain string names for each of the columns of data. If there are columns named x and y then these will be taken as the cartesian coordinates, and any remaining column will be taken as the marks. If there are no columns named x and y then the first and second columns will be taken as the cartesian coordinates.

Note that there is intentionally no default for window. The window of observation should be specified. If you really need to estimate the window, use the Ripley-Rasson estimator [ripras](#).

Value

A point pattern (an object of class "ppp", see [ppp.object](#)).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppp.object](#), [ppp](#), [as.ppp](#), [ripras](#)

selfcrossing.psp

Crossing Points in a Line Segment Pattern

Description

Finds any crossing points between the line segments in a line segment pattern.

Usage

`selfcrossing.psp(A)`

Arguments

`A` Line segment pattern (object of class "psp").

Details

This function finds any crossing points between different line segments in the line segment pattern `A`.

A crossing point occurs whenever one of the line segments in `A` intersects another line segment in `A`, at a nonzero angle of intersection.

Value

Point pattern (object of class "ppp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[crossing.psp](#), [psp.object](#), [ppp.object](#).

Examples

```
a <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
plot(a, col="green", main="selfcrossing.psp")
P <- selfcrossing.psp(a)
plot(P, add=TRUE, col="red")
```

setcov*Set Covariance of a Window***Description**

Computes the set covariance function of a window.

Usage

```
setcov(W, V=W, ...)
```

Arguments

<code>W</code>	A window (object of class "owin").
<code>V</code>	Optional. Another window.
<code>...</code>	Optional arguments passed to <code>as.mask</code> to control the pixel resolution.

Details

The set covariance function of a region W in the plane is the function $C(v)$ defined for each vector v as the area of the intersection between W and $W + v$, where $W + v$ is the set obtained by shifting (translating) W by v .

We may interpret $C(v)$ as the area of the set of all points x in W such that $x + v$ also lies in W .

This command computes a discretised approximation to the set covariance function of any plane region W represented as a window object (of class "owin", see `owin.object`). The return value is a pixel image (object of class "im") whose greyscale values are values of the set covariance function.

The set covariance is computed using the Fast Fourier Transform, unless W is a rectangle, when an exact formula is used.

If the argument V is present, then `setcov(W, V)` computes the set *cross-covariance* function $C(x)$ defined for each vector x as the area of the intersection between W and $V + x$.

Value

A pixel image (an object of class "im") representing the set covariance function of W , or the cross-covariance of W and V .

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`imcov, owin, as.owin, erosion`

Examples

```
w <- owin(c(0,1),c(0,1))
v <- setcov(w)
plot(v)
```

`shapley`*Galaxies in the Shapley Supercluster*

Description

A point pattern recording the sky positions of 4215 galaxies in the Shapley Supercluster.

Usage

```
data(shapley)
```

Format

`shapley` is an object of class "ppp" representing the point pattern of galaxy locations (see [ppp.object](#)).
`shapley.extra` is a list containing additional data described under Notes.

Notes

This dataset comes from a survey by Drinkwater et al (2004) of the Shapley Supercluster, one of the most massive concentrations of galaxies in the local universe. The data give the sky positions of 4215 galaxies observed using the FLAIR-II spectrograph on the UK Schmidt Telescope (UKST). They were kindly provided by Dr Michael Drinkwater through the Centre for Astrostatistics at Penn State University.

Sky positions are given using the coordinates Right Ascension (degrees from 0 to 360) and Declination (degrees from -90 to 90).

The point pattern has three mark variables:

Mag Galaxy magnitude (a negative logarithmic measure of visible brightness).

V Recession velocity (km/sec) inferred from redshift, with corrections applied.

SigV Estimated standard error for V.

The region covered by the survey was approximately the UKST's standard quadrilateral survey fields 382 to 384 and 443 to 446. However, a few of the galaxy positions lie outside these fields.

The point pattern dataset `shapley` consists of all 4215 galaxy locations. The observation window for this pattern is a dilated copy of the convex hull of the galaxy positions, constructed so that all galaxies lie within the window.

The auxiliary dataset `shapley.extra` contains the following components:

UKSTfields a list of seven windows (objects of class "owin") giving the UKST standard survey fields.

UKSTdomain the union of these seven fields, an object of class "owin".

plotit a function (called without arguments) that will plot the data and the survey fields in the conventional astronomical presentation, in which Right Ascension is converted to hours and minutes (1 hour equals 15 degrees) and Right Ascension decreases as we move to the right of the plot.

Source

M.J. Drinkwater, Department of Physics, University of Queensland

References

Drinkwater, M.J., Parker, Q.A., Proust, D., Slezak, E. and Quintana, H. (2004) The large scale distribution of galaxies in the Shapley Supercluster. *Publications of the Astronomical Society of Australia* **21**, 89-96. DOI 10.1071/AS03057

Examples

```
data(shapley)
shapley.extra$plotit(main="Shapley Supercluster")
```

sharpen

Data Sharpening of Point Pattern

Description

Performs Choi-Hall data sharpening of a spatial point pattern.

Usage

```
sharpen(X, ...)
## S3 method for class 'ppp'
sharpen(X, sigma=NULL, ..., varcov=NULL,
        edgecorrect=FALSE)
```

Arguments

X	A marked point pattern (object of class "ppp").
sigma	Standard deviation of isotropic Gaussian smoothing kernel.
varcov	Variance-covariance matrix of anisotropic Gaussian kernel. Incompatible with sigma.
edgecorrect	Logical value indicating whether to apply edge effect bias correction.
...	Arguments passed to density.ppp to control the pixel resolution of the result.

Details

Choi and Hall (2001) proposed a procedure for *data sharpening* of spatial point patterns. This procedure is appropriate for earthquake epicentres and other point patterns which are believed to exhibit strong concentrations of points along a curve. Data sharpening causes such points to concentrate more tightly along the curve.

If the original data points are X_1, \dots, X_n then the sharpened points are

$$\hat{X}_i = \frac{\sum_j X_j k(X_j - X_i)}{\sum_j k(X_j - X_i)}$$

where k is a smoothing kernel in two dimensions. Thus, the new point \hat{X}_i is a vector average of the nearby points $X[j]$.

The function `sharpen` is generic. It currently has only one method, for two-dimensional point patterns (objects of class "ppp").

If `sigma` is given, the smoothing kernel is the isotropic two-dimensional Gaussian density with standard deviation `sigma` in each axis. If `varcov` is given, the smoothing kernel is the Gaussian density with variance-covariance matrix `varcov`.

The data sharpening procedure tends to cause the point pattern to contract away from the boundary of the window. That is, points $X_{-i}X[i]$ that lie ‘quite close to the edge of the window of the point pattern tend to be displaced inward. If `edgecorrect=TRUE` then the algorithm is modified to correct this vector bias.

Value

A point pattern (object of class “`ppp`”) in the same window as the original pattern `X`, and with the same marks as `X`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

Choi, E. and Hall, P. (2001) Nonparametric analysis of earthquake point-process data. In M. de Gunst, C. Klaassen and A. van der Vaart (eds.) *State of the art in probability and statistics: Festschrift for Willem R. van Zwet*, Institute of Mathematical Statistics, Beachwood, Ohio. Pages 324–344.

See Also

[density.ppp](#), [smooth.ppp](#).

Examples

```
data(shapley)
X <- unmark(shapley)

Y <- sharpen(X, sigma=0.5)
```

`shift`

Apply Vector Translation

Description

Applies a vector shift of the plane to a geometrical object, such as a point pattern or a window.

Usage

```
shift(X, ...)
```

Arguments

- | | |
|----------------|--|
| <code>X</code> | Any suitable dataset representing a two-dimensional object, such as a point pattern (object of class “ <code>ppp</code> ”), or a window (object of class “ <code>owin</code> ”). |
| ... | Arguments determining the shift vector. |

Details

This is generic. Methods are provided for point patterns ([shift.ppp](#)) and windows ([shift.owin](#)).
The object is translated by the vector `vec`.

Value

Another object of the same type, representing the result of applying the shift.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[shift.ppp](#), [shift.owin](#), [rotate](#), [affine](#), [periodify](#)

[shift.im](#)

Apply Vector Translation To Pixel Image

Description

Applies a vector shift to a pixel image

Usage

```
## S3 method for class 'im'
shift(X, vec=c(0,0), ..., origin=NULL)
```

Arguments

<code>X</code>	Pixel image (object of class "im").
<code>vec</code>	Vector of length 2 representing a translation.
<code>...</code>	Ignored
<code>origin</code>	Character string determining a location that will be shifted to the origin. Options are "centroid", "midpoint" and "bottomleft". Partially matched.

Details

The spatial location of each pixel in the image is translated by the vector `vec`. This is a method for the generic function [shift](#).

If `origin` is given, then it should be one of the character strings "centroid", "midpoint" or "bottomleft". The argument `vec` will be ignored; instead the shift will be performed so that the specified geometric location is shifted to the origin. If `origin="centroid"` then the centroid of the image window will be shifted to the origin. If `origin="midpoint"` then the centre of the bounding rectangle of the image will be shifted to the origin. If `origin="bottomleft"` then the bottom left corner of the bounding rectangle of the image will be shifted to the origin.

Value

Another pixel image (of class "im") representing the result of applying the vector shift.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[shift](#)

Examples

```
# make up an image
X <- setcov(unit.square())
plot(X)

Y <- shift(X, c(10,10))
plot(Y)
# no discernible difference except coordinates are different

shift(X, origin="mid")
```

shift.owin

Apply Vector Translation To Window

Description

Applies a vector shift to a window

Usage

```
## S3 method for class 'owin'
shift(X, vec=c(0,0), ..., origin=NULL)
```

Arguments

X	Window (object of class "owin").
vec	Vector of length 2 representing a translation.
...	Ignored
origin	Character string determining a location that will be shifted to the origin. Options are "centroid", "midpoint" and "bottomleft". Partially matched.

Details

The window is translated by the vector `vec`. This is a method for the generic function [shift](#).

If `origin` is given, then it should be one of the character strings "centroid", "midpoint" or "bottomleft". The argument `vec` will be ignored; instead the shift will be performed so that the specified geometric location is shifted to the origin. If `origin="centroid"` then the centroid of the window will be shifted to the origin. If `origin="midpoint"` then the centre of the bounding rectangle of the window will be shifted to the origin. If `origin="bottomleft"` then the bottom left corner of the bounding rectangle of the window will be shifted to the origin.

Value

Another window (of class "owin") representing the result of applying the vector shift.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[shift](#), [shift.hpp](#), [periodify](#), [rotate](#), [affine](#), [centroid.owin](#)

Examples

```
W <- owin(c(0,1),c(0,1))
X <- shift(W, c(2,3))
## Not run:
plot(W)
# no discernible difference except coordinates are different

## End(Not run)
shift(W, origin="mid")
```

[shift.hpp](#)

Apply Vector Translation To Point Pattern

Description

Applies a vector shift to a point pattern.

Usage

```
## S3 method for class 'ppp'
shift(X, vec=c(0,0), ..., origin=NULL)
```

Arguments

X	Point pattern (object of class "ppp").
vec	Vector of length 2 representing a translation.
...	Ignored
origin	Character string determining a location that will be shifted to the origin. Options are "centroid", "midpoint" and "bottomleft". Partially matched.

Details

The point pattern, and its window, are translated by the vector `vec`.

This is a method for the generic function [shift](#).

If `origin` is given, then it should be one of the character strings "centroid", "midpoint" or "bottomleft". The argument `vec` will be ignored; instead the shift will be performed so that the specified geometric location is shifted to the origin. If `origin="centroid"` then the centroid of the window will be shifted to the origin. If `origin="midpoint"` then the centre of the bounding rectangle of the window will be shifted to the origin. If `origin="bottomleft"` then the bottom left corner of the bounding rectangle of the window will be shifted to the origin.

Value

Another point pattern (of class "ppp") representing the result of applying the vector shift.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[shift](#), [shift.owin](#), [periodify](#), [rotate](#), [affine](#)

Examples

```
data(cells)
X <- shift(cells, c(2,3))
## Not run:
plot(X)
# no discernible difference except coordinates are different

## End(Not run)
plot(cells, pch=16)
plot(shift(cells, c(0.03,0.03)), add=TRUE)

shift(cells, origin="mid")
```

[shift.psp](#)

Apply Vector Translation To Line Segment Pattern

Description

Applies a vector shift to a line segment pattern.

Usage

```
## S3 method for class 'psp'
shift(X, vec=c(0,0), ..., origin=NULL)
```

Arguments

X	Line Segment pattern (object of class "psp").
vec	Vector of length 2 representing a translation.
...	Ignored
origin	Character string determining a location that will be shifted to the origin. Options are "centroid", "midpoint" and "bottomleft". Partially matched.

Details

The line segment pattern, and its window, are translated by the vector vec.

This is a method for the generic function [shift](#).

If origin is given, then it should be one of the character strings "centroid", "midpoint" or "bottomleft". The argument vec will be ignored; instead the shift will be performed so that the specified geometric location is shifted to the origin. If origin="centroid" then the centroid of the window will be shifted to the origin. If origin="midpoint" then the centre of the bounding rectangle of the window will be shifted to the origin. If origin="bottomleft" then the bottom left corner of the bounding rectangle of the window will be shifted to the origin.

Value

Another line segment pattern (of class "psp") representing the result of applying the vector shift.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[shift](#), [shift.owin](#), [shift.ppp](#), [periodify](#), [rotate](#), [affine](#)

Examples

```
X <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
plot(X, col="red")
Y <- shift(X, c(0.05,0.05))
plot(Y, add=TRUE, col="blue")

shift(Y, origin="mid")
```

Description

This point pattern data set was simulated (using the Metropolis-Hastings algorithm) from a model fitted to the Numata Japanese black pine data set referred to in Baddeley and Turner (2000).

Usage

`data(simdat)`

Format

An object of class "ppp" in a square window of size 10 by 10 units.

See [ppp.object](#) for details of the format of a point pattern object.

Source

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

References

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

simplenet*Simple Example of Linear Network*

Description

A simple, artificially created, example of a linear network.

Usage

```
data(simplenet)
```

Format

`simplenet` is an object of class "linnet".

Source

Created by Adrian Baddeley.

simplify.owin*Approximate a Polygon by a Simpler Polygon*

Description

Given a polygonal window, this function finds a simpler polygon that approximates it.

Usage

```
simplify.owin(W, dmin)
```

Arguments

- | | |
|-------------------|---|
| <code>W</code> | The polygon which is to be simplified. An object of class "owin". |
| <code>dmin</code> | Numeric value. The smallest permissible length of an edge. |

Details

This function simplifies a polygon `W` by recursively deleting the shortest edge of `W` until all remaining edges are longer than the specified minimum length `dmin`, or until there are only three edges left.

The argument `W` must be a window (object of class "owin"). It should be of type "polygonal". If `W` is a rectangle, it is returned without alteration.

The simplification algorithm is not yet implemented for binary masks. If `W` is a mask, an error is generated.

Value

Another window (object of class "owin") of type "polygonal".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[owin](#)

Examples

```
data(letterR)
plot(letterR, col="red")
plot(simplify.owin(letterR, 0.3), col="blue", add=TRUE)
data(chorley)
W <- chorley>window
plot(W)
WS <- simplify.owin(W, 2)
plot(WS, add=TRUE, border="green")
points(vertices(WS))
```

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, verbose=TRUE)
```

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.
...	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).

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` or `rLGCP` depending on the model.

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`kppm`, `rThomas`, `rMatClust`, `rLGCP`, `simulate.ppm`, `simulate`

Examples

```
data(redwood)
fit <- kppm(redwood, ~1, "Thomas")
simulate(fit, 2)
```

`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, ...,
         start = NULL,
         control = default.rmhcontrol(object),
         project=TRUE,
         verbose=FALSE, progress=(nsim > 1))
```

Arguments

<code>object</code>	Fitted point process model. An object of class "ppm".
<code>nsim</code>	Number of simulated realisations.
<code>start</code>	Data determining the initial state of the Metropolis-Hastings algorithm. See <code>rmhstart</code> for description of these arguments. Defaults to <code>list(x.start=data.ppm(model))</code> .
<code>control</code>	Data controlling the running of the Metropolis-Hastings algorithm. See <code>rmhcontrol</code> for description of these arguments.
<code>...</code>	Further arguments passed to <code>rmhcontrol</code> , or to <code>rmh.default</code> , or to covariate functions in the model.
<code>project</code>	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 <code>project=TRUE</code> the closest valid model will be simulated; if <code>project=FALSE</code> an error will occur.
<code>verbose</code>	Logical flag indicating whether to print progress reports from <code>rmh.ppm</code> during the simulation of each point pattern.
<code>progress</code>	Logical flag indicating whether to print progress reports for the sequence of simulations.

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`.

Value

A list of length `nsim` containing simulated point patterns (objects of class "ppp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`ppm`, `simulate.kppm`, `simulate`

Examples

```
fit <- ppm(japanesepines, ~1, Strauss(0.1))
simulate(fit, 2)
```

<code>simulate.slrm</code>	<i>Simulate a Fitted Spatial Logistic Regression Model</i>
----------------------------	--

Description

Generates simulated realisations from a fitted spatial logistic regression model

Usage

```
## S3 method for class 'slrm'
simulate(object, nsim = 1, seed=NULL, ...,
          window=NULL, covariates=NULL, verbose=TRUE)
```

Arguments

<code>object</code>	Fitted spatial logistic regression model. An object of class "slrm".
<code>nsim</code>	Number of simulated realisations.
<code>seed</code>	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 <code>set.seed</code> before simulating the point patterns.
<code>...</code>	Ignored.
<code>window</code>	Optional. Window (object of class "owin") in which the model should be simulated.
<code>covariates</code>	Optional. A named list containing new values for the covariates in the model.
<code>verbose</code>	Logical. Whether to print progress reports (when <code>nsim > 1</code>).

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`slrm`, `rpoispp`, `simulate.ppm`, `simulate.kppm`, `simulate`

Examples

```
X <- copper$SouthPoints
fit <- slrm(X ~ 1)
simulate(fit, 2)
fitxy <- slrm(X ~ x+y)
simulate(fitxy, 2, window=square(2))
```

slrm

Spatial Logistic Regression

Description

Fits a spatial logistic regression model to a spatial point pattern.

Usage

```
slrm(formula, ..., data = NULL, offset = TRUE, link = "logit",
      dataAtPoints=NULL, splitby=NULL)
```

Arguments

formula	The model formula. See Details.
...	Optional arguments passed to pixellate 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")`.

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`.

Author(s)

Adrian Baddeley <adrian@maths.uwa.edu.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

References

- Agterberg, F.P. (1974) Automatic contouring of geological maps to detect target areas for mineral exploration. *Journal of the International Association for Mathematical Geology* **6**, 373–395.
- Baddeley, A., Berman, M., Fisher, N.I., Hardegen, A., Milne, R.K., Schuhmacher, D., Shah, R. and Turner, R. (2010) Spatial logistic regression and change-of-support for spatial Poisson point processes. *Electronic Journal of Statistics* **4**, 1151–1201. doi: 10.1214/10-EJS581
- Tukey, J.W. (1972) Discussion of paper by F.P. Agterberg and S.C. Robinson. *Bulletin of the International Statistical Institute* **44** (1) p. 596. Proceedings, 38th Congress, International Statistical Institute.

See Also

`anova.slrn`, `coef.slrn`, `fitted.slrn`, `logLik.slrn`, `plot.slrn`, `predict.slrn`

Examples

```
X <- copper$SouthPoints
slrm(X ~ 1)
slrm(X ~ x+y)

slrm(X ~ x+y, link="cloglog")
# specify a grid of 1 km square pixels
slrm(X ~ 1, eps=1)

Y <- copper$SouthLines
Z <- distmap(Y)
slrm(X ~ Z)
slrm(X ~ Z, dataAtPoints=list(Z=nncross(X,Y)$dist))

dat <- list(A=X, V=Z)
slrm(A ~ V, data=dat)
```

`smooth.fv`

Apply Smoothing to Function Values

Description

Applies spline smoothing to the values in selected columns of a function value table.

Usage

```
smooth.fv(x, which = "*", ...)
```

Arguments

- | | |
|--------------------|---|
| <code>x</code> | Values to be smoothed. A function value table (object of class "fv", see fv.object). |
| <code>which</code> | Character vector identifying which columns of the table should be smoothed. Either a vector containing names of columns, or one of the wildcard strings "*" or "." explained below. |
| <code>...</code> | Extra arguments passed to <code>smooth.spline</code> to control the smoothing. |

Details

`smooth.spline` is applied to each of the selected columns in turn (using the function argument as the x coordinate and the selected column as the y coordinate). The original function values are then replaced by the corresponding smooth interpolated function values.

The argument which specifies which of the columns of function values in x will be smoothed. The default (indicated by the wildcard `which="*"`) is to smooth all function values, i.e.\ all columns except the function argument. Alternatively `which=". "` designates the subset of function values that are displayed in the default plot. Alternatively `which` can be a character vector containing the names of columns of x .

Value

Another function value table (object of class "fv") of the same format.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`with.fv`, `fv.object`, `smooth.spline`

Examples

```
data(cells)
G <- Gest(cells)
plot(G)
plot(smooth.fv(G, df=9), add=TRUE)
```

`smooth.msr`

Smooth a Signed or Vector-Valued Measure

Description

Apply kernel smoothing to a signed measure or vector-valued measure.

Usage

```
smooth.msr(X, ...)
```

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.

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

For signed measures, a pixel image (object of class "im"). For vector-valued measures, a list of pixel images; the list also belongs to the class "listof" so that it can be printed and plotted.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

References

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

Examples

```
example(msr)
plot(smooth.msr(rp))
plot(smooth.msr(rs))
```

smooth.ppp

Spatial smoothing of observations at irregular points

Description

Performs spatial smoothing of numeric values observed at a set of irregular locations. Uses Gaussian kernel smoothing and least-squares cross-validated bandwidth selection.

Usage

```
smooth.ppp(X, ..., weights = rep(1, npoints(X)), at="pixels")
markmean(X, ...)
markvar(X, ...)
```

Arguments

- | | |
|---------|--|
| X | A marked point pattern (object of class "ppp"). |
| ... | Arguments passed to bw.smoothppp and density.ppp to control the kernel smoothing and the pixel resolution of the result. |
| weights | Optional weights attached to the observations. |
| at | String specifying whether to compute the smoothed values at a grid of pixel locations (at="pixels") or only at the points of X (at="points"). |

Details

The function `smooth.hpp` performs spatial smoothing of numeric values observed at a set of irregular locations. The functions `markmean` and `markvar` are wrappers for `smooth.hpp` which compute the spatially-varying mean and variance of the marks of a point pattern.

Smoothing is performed by Gaussian kernel weighting. If the observed values are v_1, \dots, v_n at locations x_1, \dots, x_n respectively, then the smoothed value at a location u is (ignoring edge corrections)

$$g(u) = \frac{\sum_i k(u - x_i)v_i}{\sum_i k(u - x_i)}$$

where k is a Gaussian kernel. This is known as the Nadaraya-Watson smoother (Nadaraya, 1964, 1989; Watson, 1964). By default, the smoothing kernel bandwidth is chosen by least squares cross-validation (see below).

The argument `X` must be a marked point pattern (object of class "ppp", see [ppp.object](#)). The points of the pattern are taken to be the observation locations x_i , and the marks of the pattern are taken to be the numeric values v_i observed at these locations.

The marks are allowed to be a data frame (in `smooth.hpp` and `markmean`). Then the smoothing procedure is applied to each column of marks.

The numerator and denominator are computed by `density.hpp`. The arguments ... control the smoothing kernel parameters and determine whether edge correction is applied. The smoothing kernel bandwidth can be specified by either of the arguments `sigma` or `varcov` which are passed to `density.hpp`. If neither of these arguments is present, then by default the bandwidth is selected by least squares cross-validation, using `bw.smooth.hpp`.

The optional argument `weights` allows numerical weights to be applied to the data. If a weight w_i is associated with location x_i , then the smoothed function is (ignoring edge corrections)

$$g(u) = \frac{\sum_i k(u - x_i)v_iw_i}{\sum_i k(u - x_i)w_i}$$

An alternative to kernel smoothing is inverse-distance weighting, which is performed by `idw`.

Value

If `X` has a single column of marks:

- If `at="pixels"` (the default), the result is a pixel image (object of class "im"). Pixel values are values of the interpolated function.
- If `at="points"`, the result is a numeric vector of length equal to the number of points in `X`. Entries are values of the interpolated function at the points of `X`.

If `X` has a data frame of marks:

- If `at="pixels"` (the default), the result is a named list of pixel images (object of class "im"). There is one image for each column of marks. This list also belongs to the class `listof`, for which there is a plot method.
- If `at="points"`, the result is a data frame with one row for each point of `X`, and one column for each column of marks. Entries are values of the interpolated function at the points of `X`.

The return value has attributes "sigma" and "varcov" which report the smoothing bandwidth that was used.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

References

- Nadaraya, E.A. (1964) On estimating regression. *Theory of Probability and its Applications* **9**, 141–142.
- Nadaraya, E.A. (1989) *Nonparametric estimation of probability densities and regression curves*. Kluwer, Dordrecht.
- Watson, G.S. (1964) Smooth regression analysis. *Sankhya A* **26**, 359–372.

See Also

`density.ppp`, `bw.smoothppp`, `ppp.object`, `im.object`.

See `idw` for inverse-distance weighted smoothing.

To perform interpolation, see also the `akima` package.

Examples

```
# Longleaf data - tree locations, marked by tree diameter
data(longleaf)
# Local smoothing of tree diameter (automatic bandwidth selection)
Z <- smooth.ppp(longleaf)
# Kernel bandwidth sigma=5
plot(smooth.ppp(longleaf, 5))
# mark variance
plot(markvar(longleaf, sigma=5))
# data frame of marks: trees marked by diameter and height
data(finpines)
plot(smooth.ppp(finpines, sigma=2))
```

Description

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

Usage

```
Softcore(kappa)
```

Arguments

kappa	The exponent κ of the Soft Core interaction
-------	--

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.

Thus the process has probability density

$$f(x_1, \dots, x_n) = \alpha \beta^{n(x)} \exp \left\{ - \sum_{i < j} \left(\frac{\sigma}{||x_i - x_j||} \right)^{2/\kappa} \right\}$$

where x_1, \dots, 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 interpoint interaction decreasing smoothly with distance. The strength of interaction is controlled by the parameter σ , a positive real number, with larger values corresponding to stronger interaction; and by the exponent κ in the range $(0, 1)$, with larger values corresponding to weaker interaction. If $\sigma = 0$ the model reduces to the Poisson point process. If $\sigma > 0$, the process is well-defined only for κ in $(0, 1)$. The limit of the model as $\kappa \rightarrow 0$ is the hard core process with hard core distance $h = \sigma$.

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.

Note the only 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()`.

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

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

```
data(cells)
ppm(cells, ~1, Softcore(kappa=0.5), correction="isotropic")
# fit the stationary Soft Core process to 'cells'
```

solutionset

Evaluate Logical Expression Involving Pixel Images and Return Region Where Expression is True

Description

Given a logical expression involving one or more pixel images, find all pixels where the expression is true, and assemble these pixels into a window.

Usage

```
solutionset(..., envir)
```

Arguments

- | | |
|-------|--|
| ... | An expression in the R language, involving one or more pixel images. |
| envir | Optional. The environment in which to evaluate the expression. |

Details

Given a logical expression involving one or more pixel images, this function will find all pixels where the expression is true, and assemble these pixels into a spatial window.

Pixel images in `spatstat` are represented by objects of class "im" (see [im.object](#)). These are essentially matrices of pixel values, with extra attributes recording the pixel dimensions, etc.

Suppose X is a pixel image. Then `eval.im(abs(X) > 3)` will find all the pixels in X for which the pixel value is greater than 3 in absolute value, and return a window containing all these pixels.

Suppose X and Y are two pixel images with compatible dimensions: they have the same number of pixels, the same physical size of pixels, and the same bounding box. Then `eval.im(X > Y)` will find all pixels for which the pixel value of X is greater than the corresponding pixel value of Y , and return a window containing these pixels.

In general, expr can be any logical expression involving (a) the *names* of pixel images, (b) scalar constants, and (c) functions which are vectorised. See the Examples.

The expression expr is evaluated by `eval.im`. The expression expr must be vectorised. There must be at least one pixel image in the expression. All images must have compatible dimensions.

Value

A spatial window (object of class "owin", see [owin.object](#)).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[im.object](#), [owin.object](#), [eval.im](#), [levelset](#)

Examples

```
# test images
X <- as.im(function(x,y) { x^2 - y^2 }, unit.square())
Y <- as.im(function(x,y) { 3 * x + y - 1}, unit.square())

W <- solutionset(abs(X) > 0.1)
W <- solutionset(X > Y)
W <- solutionset(X + Y >= 1)

area.owin(solutionset(X < Y))
```

Description

Allows the user to examine and reset the values of global parameters which control actions in the **spatstat** package.

Usage

```
spatstat.options(...)
reset.spatstat.options()
```

Arguments

... Either empty, or a succession of parameter names in quotes, or a succession of name=value pairs. See below for the parameter names.

Details

The function `spatstat.options` allows the user to examine and reset the values of global parameters which control actions in the **spatstat** package. It is analogous to the system function [options](#).

The function `reset.spatstat.options` resets all the global parameters in **spatstat** to their original, default values.

The global parameters are:

scalable Logical flag indicating whether the new code in `rmh.default` which makes the results scalable (invariant to change of units) should be used. In order to recover former behaviour (so that previous results can be reproduced) set this option equal to FALSE. See the “Warning” section in the help for `rmh()` for more detail.

npixel Default number of pixels in a binary mask or pixel image. Either an integer, or a pair of integers, giving the number of pixels in the x and y directions respectively.

- maxedgewt** Edge correction weights will be trimmed so as not to exceed this value. This applies to the weights computed by `edge.Trans` or `edge.Ripley` and used in `Kest` and its relatives.
- par.binary** List of arguments to be passed to the function `image` when displaying a binary image mask (in `plot.owin` or `plot.ppp`). Typically used to reset the colours of foreground and background.
- par.persp** List of arguments to be passed to the function `persp` when displaying a real-valued image, such as the fitted surfaces in `plot.ppm`.
- par.points** List of arguments controlling the plotting of point patterns by `plot.ppp`.
- par.contour** List of arguments controlling contour plots of pixel images by `contour.im`.
- par.fv** List of arguments controlling the plotting of functions by `plot.fv` and its relatives.
- ndummy.min** The minimum number of dummy points in a quadrature scheme created by `default.dummy`. Either an integer or a pair of integers giving the minimum number of dummy points in the x and y directions respectively.
- ngrid.disc** Number of points in the square grid used to compute a discrete approximation to the areas of discs in `areaLoss` and `areaGain` when exact calculation is not available. A single integer.
- image.colfun** Function determining the default colour map for `plot.im`. When called with one integer argument n, this function should return a character vector of length n specifying n different colours.
- progress** Character string determining the style of progress reports printed by `progressreport`. Either "tty" or "txtbar".
- checkpolygons** Logical flag indicating whether the functions `owin` and `as.owin` should check the validity of polygon data. It is advisable to leave this set to TRUE. If you set `checkpolygons=FALSE`, no checking will be performed, making it possible to create window objects whose topology is garbled. This can be useful for inspecting spatial data that contain errors, for example, when converting data from shapefiles. However, other functions in `spatstat` may return incorrect answers on these data.
- checksegments** Logical flag indicating whether the functions `psp` and `as.psp` should check the validity of line segment data (in particular, checking that the endpoints of the line segments are inside the specified window). It is advisable to leave this flag set to TRUE.
- gpclib** Logical flag indicating whether the polygon clipping library `gpclib` can be used. The default is FALSE, which means that the package `gpclib` is not loaded, and certain polygon operations (such as the intersection and union of polygons) will be approximated by raster operations. If this option is re-set to TRUE, then the package `gpclib` will be loaded, and polygon operations such as the union and intersection of polygons will result in a polygon. The licence for `gpclib` is not Free Open Source and explicitly forbids commercial use. See `library(help=gpclib)`. If you are using `spatstat` for commercial purposes, this option should remain set to FALSE.
- maxmatrix** The maximum permitted size (rows times columns) of matrices generated by `spatstat`'s internal code. Used by `ppm` and `predict.ppm` (for example) to decide when to split a large calculation into blocks. Defaults to $2^{24}=16777216$.
- huge.npoints** The maximum value of n for which `runif(n)` will not generate an error (possible errors include failure to allocate sufficient memory, and integer overflow of n). An attempt to generate more than this number of random points triggers a warning from `runifpoint` and other functions. Defaults to 1e6.
- expand** The default expansion factor (area inflation factor) for expansion of the simulation window in `rmh` (see `rmhcontrol`). Initialised to 2.
- fasteval** One of the strings 'off', 'on' or 'test' determining whether to use accelerated C code to evaluate the conditional intensity of a Gibbs model. Initialised to 'on'.

density Logical. Indicates whether to use accelerated C code to evaluate density.ppp(X, at="points")
Initialised to TRUE.

n.bandwidth Integer. Number of trial values of smoothing bandwidth to use for cross-validation
in bw.relrisk and similar functions.

psstG.remove.zeroes Logical value, determining whether the algorithm in psstG removes or retains the contributions to the function from pairs of points that are identical. If these are retained then the function has a jump at $r = 0$. Initialised to TRUE.

Kcom.remove.zeroes Logical value, determining whether the algorithm in Kcom and Kres removes or retains the contributions to the function from pairs of points that are identical. If these are retained then the function has a jump at $r = 0$. Initialised to TRUE.

psstA.ngrid Single integer, controlling the accuracy of the discrete approximation of areas computed in the function psstA. The area of a disc is approximated by counting points on an $n \times n$ grid. Initialised to 32.

psstA.nr Single integer, determining the number of distances r at which the function psstA will be evaluated (in the default case where argument r is absent). Initialised to 30.

project.fast Logical. If TRUE, the algorithm of project.ppm will be accelerated using a shortcut.
Initialised to FALSE.

exactdt.checks.data,closepairs.newcode Logical. For software development purposes only. Do not change these values, unless you are Adrian Baddeley.

rmh.p, rmh.q, rmh.nrep New default values for the parameters p, q and nrep in the Metropolis-Hastings simulation algorithm. These override the defaults in rmhcontrol.default.

print.ppm.SE Default rule used by print.ppm to decide whether to calculate and print standard errors of the estimated coefficients of the model. One of the strings "always", "never" or "poisson" (the latter indicating that standard errors will be calculated only for Poisson models). The default is "poisson" because the calculation for non-Poisson models can take a long time.

nvoxel Default number of voxels in a 3D image, typically for calculating the distance transform in F3est. Initialised to 4 megavoxels: nvoxel = $2^{22} = 4194304$.

fastK.lgcp Logical. Whether to use fast or slow algorithm to compute the (theoretical) K -function of a log-Gaussian Cox process for use in lgcp.estK or Kmodel. The slow algorithm uses accurate numerical integration; the fast algorithm uses Simpson's Rule for numerical integration, and is about two orders of magnitude faster. Initialised to FALSE.

If no arguments are given, the current values of all parameters are returned, in a list.

If one parameter name is given, the current value of this parameter is returned (not in a list, just the value).

If several parameter names are given, the current values of these parameters are returned, in a list.

If name=value pairs are given, the named parameters are reset to the given values, and the previous values of these parameters are returned, in a list.

Value

Either a list of parameters and their values, or a single value. See Details.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also[options](#)**Examples**

```
# save current values
oldopt <- spatstat.options()

spatstat.options("npixel")
spatstat.options(npixel=150)
spatstat.options(npixel=c(100,200))

spatstat.options(par.binary=list(col=grey(c(0.5,1)))))

spatstat.options(par.persp=list(theta=-30,phi=40,d=4))
# see help(persp.default) for other options

# revert
spatstat.options(oldopt)
```

split.im*Divide Image Into Sub-images***Description**

Divides a pixel image into several sub-images according to the value of a factor, or according to the tiles of a tessellation.

Usage

```
## S3 method for class 'im'
split(x, f, ..., drop = FALSE)
```

Arguments

<code>x</code>	Pixel image (object of class "im").
<code>f</code>	Splitting criterion. Either a tessellation (object of class "tess") or a pixel image with factor values.
<code>...</code>	Ignored.
<code>drop</code>	Logical value determining whether each subset should be returned as a pixel images (<code>drop=FALSE</code>) or as a one-dimensional vector of pixel values (<code>drop=TRUE</code>).

Details

This is a method for the generic function [split](#) for the class of pixel images. The image `x` will be divided into subsets determined by the data `f`. The result is a list of these subsets.

The splitting criterion may be either

- a tessellation (object of class "tess"). Each tile of the tessellation delineates a subset of the spatial domain.

- a pixel image (object of class "im") with factor values. The levels of the factor determine subsets of the spatial domain.

If `drop=FALSE` (the default), the result is a list of pixel images, each one a subset of the pixel image `x`, obtained by restricting the pixel domain to one of the subsets. If `drop=TRUE`, then the pixel values are returned as numeric vectors.

Value

If `drop=FALSE`, a list of pixel images (objects of class "im"). It is also of class "listof" so that it can be plotted immediately.

If `drop=TRUE`, a list of numeric vectors.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[by.im](#), [tess](#), [im](#)

Examples

```
W <- square(1)
X <- as.im(function(x,y){sqrt(x^2+y^2)}, W)
Y <- dirichlet(runifpoint(12, W))
plot(split(X,Y))
```

split.ppp

Divide Point Pattern into Sub-patterns

Description

Divides a point pattern into several sub-patterns, according to their marks, or according to any user-specified grouping.

Usage

```
## S3 method for class 'ppp'
split(x, f = marks(x), drop=FALSE, un=NULL, ...)
## S3 replacement method for class 'ppp'
split(x, f = marks(x), drop=FALSE, un=missing(f), ...) <- value
```

Arguments

<code>x</code>	A two-dimensional point pattern. An object of class "ppp".
<code>f</code>	Data determining the grouping. Either a factor, a pixel image with factor values, a tessellation, or the name of one of the columns of marks.
<code>drop</code>	Logical. Determines whether empty groups will be deleted.
<code>un</code>	Logical. Determines whether the resulting subpatterns will be unmarked (i.e. whether marks will be removed from the points in each subpattern).

...	Other arguments are ignored.
value	List of point patterns.

Details

The function `split.ppp` divides up the points of the point pattern x into several sub-patterns according to the values of f . The result is a list of point patterns.

The argument f may be

- a factor, of length equal to the number of points in x . The levels of f determine the destination of each point in x . The i th point of x will be placed in the sub-pattern `split.ppp(x)$l` where $l = f[i]$.
- a pixel image (object of class "im") with factor values. The pixel value of f at each point of x will be used as the classifying variable.
- a tessellation (object of class "tess"). Each point of x will be classified according to the tile of the tessellation into which it falls.
- a character string, matching the name of one of the columns of `marks`, if `marks(x)` is a data frame. This column should be a factor.

If f is missing, then it will be determined by the marks of the point pattern. The pattern x can be either

- a multitype point pattern (a marked point pattern whose marks vector is a factor). Then f is taken to be the marks vector. The effect is that the points of each type are separated into different point patterns.
- a marked point pattern with a data frame of marks, containing at least one column that is a factor. The first such column will be used to determine the splitting factor f .

Some of the sub-patterns created by the split may be empty. If `drop=TRUE`, then empty sub-patterns will be deleted from the list. If `drop=FALSE` then they are retained.

The argument `un` determines how to handle marks in the case where x is a marked point pattern. If `un=TRUE` then the marks of the points will be discarded when they are split into groups, while if `un=FALSE` then the marks will be retained.

If f and `un` are both missing, then the default is `un=TRUE` for multitype point patterns and `un=FALSE` for marked point patterns with a data frame of marks.

The result of `split.ppp` has class "splittppp" and can be plotted using `plot.splittppp`.

The assignment function `split<-.ppp` updates the point pattern x so that it satisfies `split(x, f, drop, un) = value`. The argument `value` is expected to be a list of point patterns, one for each level of f . These point patterns are expected to be compatible with the type of data in the original pattern x .

Splitting can also be undone by the function `superimpose`.

Value

The value of `split.ppp` is a list of point patterns. The components of the list are named by the levels of f . The list also has the class "splittppp".

The assignment form `split<-.ppp` returns the updated point pattern x .

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[cut.ppp](#), [plot.splitppp](#), [superimpose](#), [im](#), [tess](#), [ppp.object](#)

Examples

```
# (1) Splitting by marks

# Multitype point pattern: separate into types
data(amacrine)
u <- split(amacrine)

# plot them
plot(split(amacrine))

# the following are equivalent:
amon <- split(amacrine)$on
amon <- unmark(amacrine[amacrine$marks == "on"])

# the following are equivalent:
amon <- split(amacrine, un=FALSE)$on
amon <- amacrine[amacrine$marks == "on"]

# Scramble the locations of the 'on' cells
u <- split(amacrine)
u$on <- runifpoint(amon$n, amon>window)
split(amacrine) <- u

# Point pattern with continuous marks
data(longleaf)

# cut the range of tree diameters into three intervals
# using cut.ppp
long3 <- cut(longleaf, breaks=3)
# now split them
long3split <- split(long3)

# (2) Splitting by a factor

# Unmarked point pattern
data(swedishpines)
# cut & split according to nearest neighbour distance
f <- cut(nndist(swedishpines), 3)
u <- split(swedishpines, f)

# (3) Splitting over a tessellation
tes <- tess(xgrid=seq(0,96,length=5),ygrid=seq(0,100,length=5))
v <- split(swedishpines, tes)
```

Description

Divides a multidimensional point pattern into several sub-patterns, according to their marks, or according to any user-specified grouping.

Usage

```
## S3 method for class 'ppx'
split(x, f = marks(x), drop=FALSE, un=NULL, ...)
```

Arguments

<code>x</code>	A multi-dimensional point pattern. An object of class "ppx".
<code>f</code>	Data determining the grouping. Either a factor, or the name of one of the columns of marks.
<code>drop</code>	Logical. Determines whether empty groups will be deleted.
<code>un</code>	Logical. Determines whether the resulting subpatterns will be unmarked (i.e. whether marks will be removed from the points in each subpattern).
<code>...</code>	Other arguments are ignored.

Details

The generic command `split` allows a dataset to be separated into subsets according to the value of a grouping variable.

The function `split.ppx` is a method for the generic `split` for the class "ppx" of multidimensional point patterns. It divides up the points of the point pattern `x` into several sub-patterns according to the values of `f`. The result is a list of point patterns.

The argument `f` may be

- a factor, of length equal to the number of points in `x`. The levels of `f` determine the destination of each point in `x`. The `i`th point of `x` will be placed in the sub-pattern `split.ppx(x)$l` where `l = f[i]`.
- a character string, matching the name of one of the columns of marks, if `marks(x)` is a data frame. This column should be a factor.

If `f` is missing, then it will be determined by the marks of the point pattern. The pattern `x` can be either

- a multitype point pattern (a marked point pattern whose marks vector is a factor). Then `f` is taken to be the marks vector. The effect is that the points of each type are separated into different point patterns.
- a marked point pattern with a data frame or hyperframe of marks, containing at least one column that is a factor. The first such column will be used to determine the splitting factor `f`.

Some of the sub-patterns created by the `split` may be empty. If `drop=TRUE`, then empty sub-patterns will be deleted from the list. If `drop=FALSE` then they are retained.

The argument `un` determines how to handle marks in the case where `x` is a marked point pattern. If `un=TRUE` then the marks of the points will be discarded when they are split into groups, while if `un=FALSE` then the marks will be retained.

If `f` and `un` are both missing, then the default is `un=TRUE` for multitype point patterns and `un=FALSE` for marked point patterns with a data frame of marks.

The result of `split.ppx` has class "splitppx" and "listof". There are methods for `print`, `summary` and `plot`.

Value

A list of point patterns. The components of the list are named by the levels of *f*. The list also has the class "splitppx" and "listof".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppx](#), [plot.listof](#)

Examples

```
df <- data.frame(x=runif(4),y=runif(4),t=runif(4),
                  age=rep(c("old", "new"), 2),
                  size=runif(4))
X <- ppx(data=df, coord.type=c("s","s","t","m","m"))
X
split(X)
```

spokes

Spokes pattern of dummy points

Description

Generates a pattern of dummy points in a window, given a data point pattern. The dummy points lie on the radii of circles emanating from each data point.

Usage

```
spokes(x, y, nrad = 3, nper = 3, fctr = 1.5, Mdefault = 1)
```

Arguments

<i>x</i>	Vector of <i>x</i> coordinates of data points, or a list with components <i>x</i> and <i>y</i> , or a point pattern (an object of class ppp).
<i>y</i>	Vector of <i>y</i> coordinates of data points. Ignored unless <i>x</i> is a vector.
<i>nrad</i>	Number of radii emanating from each data point.
<i>nper</i>	Number of dummy points per radius.
<i>fctr</i>	Scale factor. Length of largest spoke radius is <i>fctr</i> * <i>M</i> where <i>M</i> is the mean nearest neighbour distance for the data points.
<i>Mdefault</i>	Value of <i>M</i> to be used if <i>x</i> has length 1.

Details

This function is useful in creating dummy points for quadrature schemes (see [quadscheme](#)).

Given the data points, the function creates a collection of $nrad * nper * \text{length}(x)$ dummy points.

Around each data point $(x[i], y[i])$ there are $nrad * nper$ dummy points, lying on $nrad$ radii emanating from $(x[i], y[i])$, with $nper$ dummy points equally spaced along each radius.

The (equal) spacing of dummy points along each radius is controlled by the factor `fctr`. The distance from a data point to the furthest of its associated dummy points is `fctr * M` where `M` is the mean nearest neighbour distance for the data points.

If there is only one data point the nearest neighbour distance is infinite, so the value `Mdefault` will be used in place of `M`.

If `x` is a point pattern, then the value returned is also a point pattern, which is clipped to the window of `x`. Hence there may be fewer than $nrad * nper * \text{length}(x)$ dummy points in the pattern returned.

Value

If argument `x` is a point pattern, a point pattern with window equal to that of `x`. Otherwise a list with two components `x` and `y`. In either case the components `x` and `y` of the value are numeric vectors giving the coordinates of the dummy points.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[quad.object](#), [quadscheme](#), [inside.owin](#), [gridcentres](#), [stratrand](#)

Examples

```
dat <- runifrect(10)
## Not run:
plot(dat)

## End(Not run)
dum <- spokes(dat$x, dat$y)
## Not run:
points(dum$x, dum$y, pch=".") 

## End(Not run)
Q <- quadscheme(dat, dum)
```

spruces	<i>Spruces Point Pattern</i>
---------	------------------------------

Description

The data give the locations of Norwegian spruce trees in a natural forest stand in Saxonia, Germany. Each tree is marked with its diameter at breast height.

Usage

```
data(spruces)
```

Format

An object of class "ppp" representing the point pattern of tree locations in a 56 x 38 metre sampling region. Each tree is marked with its diameter at breast height. All values are given in metres.

See [ppp.object](#) for details of the format of a point pattern object. The marks are numeric.

These data have been analysed by Fiksel (1984, 1988), Stoyan et al (1987), Penttinen et al (1992) and Goulard et al (1996).

Source

Stoyan et al (1987). Original source unknown.

References

- Fiksel, T. (1984) Estimation of parameterized pair potentials of marked and nonmarked Gibbsian point processes. *Elektron. Informationsverarb. u. Kybernet.* **20**, 270–278.
- Fiksel, T. (1988) Estimation of interaction potentials of Gibbsian point processes. *Statistics* **19**, 77–86
- Goulard, M., S\"arkk\"a, A. and Grabarnik, P. (1996) Parameter estimation for marked Gibbs point processes through the maximum pseudolikelihood method. *Scandinavian Journal of Statistics* **23**, 365–379.
- Penttinen, A., Stoyan, D. and Henttonen, H. (1992) Marked point processes in forest statistics. *Forest Science* **38**, 806–824.
- Stoyan, D., Kendall, W.S. and Mecke, J. (1987) *Stochastic Geometry and its Applications*. Wiley.

Examples

```
data(spruces)
plot(spruces)
# To reproduce Goulard et al. Figure 3
plot(spruces, maxsize=5*max(spruces$marks))
plot(unmark(spruces), add=TRUE)
```

square*Square Window***Description**

Creates a square window

Usage

```
square(r=1)
unit.square()
```

Arguments

r Numeric. The side length of the square, or a vector giving the minimum and maximum coordinate values.

Details

If **r** is a number, **square(r)** is a shortcut for creating a window object representing the square $[0, r] \times [0, r]$. It is equivalent to the command **owin(c(0,r),c(0,r))**.

If **r** is a vector of length 2, then **square(r)** creates the square with x and y coordinates ranging from **r[1]** to **r[2]**.

unit.square creates the unit square $[0, 1] \times [0, 1]$. It is equivalent to **square(1)** or **square()** or **owin(c(0,1),c(0,1))**.

These commands are included mainly to improve the readability of some code.

Value

An object of class "owin" (see [owin.object](#)) specifying a window.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[owin.object](#), [owin](#)

Examples

```
W <- square(10)
W <- square(c(-1,1))
```

stieltjes*Compute Integral of Function Against Cumulative Distribution*

Description

Computes the Stieltjes integral of a function f with respect to a function M .

Usage

```
stieltjes(f, M, ...)
```

Arguments

<code>f</code>	The integrand. A function in the R language.
<code>M</code>	The cumulative function against which <code>f</code> will be integrated. An object of class "fv".
<code>...</code>	Additional arguments passed to <code>f</code> .

Details

This command computes the Stieltjes integral

$$I = \int f(x)dM(x)$$

of a real-valued function $f(x)$ with respect to a nondecreasing function $M(x)$.

One common use of the Stieltjes integral is to find the mean value of a random variable from its cumulative distribution function $F(x)$. The mean value is the Stieltjes integral of $f(x) = x$ with respect to $F(x)$.

The argument `f` should be a function in the R language. It should accept a numeric vector argument `x` and should return a numeric vector of the same length.

The argument `M` should be a function value table (object of class "fv", see [fv.object](#)). Such objects are returned by the commands `link{Kest}`, [Gest](#), etc.

Value

A list containing the value of the Stieltjes integral computed using each of the versions of the function `M`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[fv.object](#), [Gest](#)

Examples

```
data(redwood)
# estimate cdf of nearest neighbour distance
G <- Gest(redwood)
# compute estimate of mean nearest neighbour distance
stieljes(function(x){x}, G)
# estimated probability of a distance in the interval [0.1,0.2]
stieljes(function(x,a,b){ (x >= a) & (x <= b)}, G, a=0.1, b=0.2)
```

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

$$f(x_1, \dots, x_n) = \alpha \beta^{n(x)} \gamma^{s(x)}$$

where x_1, \dots, 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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

References

- Kelly, F.P. and Ripley, B.D. (1976) On Strauss's model for clustering. *Biometrika* **63**, 357–360.
Strauss, D.J. (1975) A model for clustering. *Biometrika* **63**, 467–475.

See Also

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

Examples

```
Strauss(r=0.1)
# prints a sensible description of itself
data(cells)

## Not run:
ppm(cells, ~1, Strauss(r=0.07))
# fit the stationary Strauss process to 'cells'

## End(Not run)

ppm(cells, ~polynom(x,y,3), Strauss(r=0.07))
# fit a nonstationary Strauss process with log-cubic polynomial trend
```

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)
```

Arguments

- | | |
|----|---|
| r | The interaction radius of the Strauss interaction |
| hc | The hard core distance |

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

$$f(x_1, \dots, x_n) = \alpha \beta^{n(x)} \gamma^{s(x)}$$

where x_1, \dots, 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 process reduces to a classical hard core 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 .

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()`.

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

- Baddeley, A. and Turner, R. (2000) Practical maximum pseudolikelihood for spatial point patterns. *Australian and New Zealand Journal of Statistics* **42**, 283–322.
- Ripley, B.D. (1981) *Spatial statistics*. John Wiley and Sons.
- Strauss, D.J. (1975) A model for clustering. *Biometrika* **63**, 467–475.

See Also

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

Examples

```

StraussHard(r=1, hc=0.02)
# prints a sensible description of itself

data(cells)

## Not run:
ppm(cells, ~1, StraussHard(r=0.1, hc=0.05))
# fit the stationary Strauss/hard core process to 'cells'

## End(Not run)

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

```

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppm](#)

Examples

```
data(swedishpines)
fitS <- ppm(swedishpines, ~1, Strauss(7))
X <- rpoispp(summary(swedishpines)$intensity, win=swedishpines$window)
suffstat(fitS, X)
```

summary.im *Summarizing a Pixel Image*

Description

summary method for class "im".

Usage

```
## S3 method for class 'im'  
summary(object, ...)  
## S3 method for class 'summary.im'  
print(x, ...)
```

Arguments

object	A pixel image.
...	Ignored.
x	Object of class "summary.im" as returned by summary.im.

Details

This is a method for the generic `summary` for the class "im". An object of class "im" describes a pixel image. See `im.object`) for details of this class.

`summary.im` extracts information about the pixel image, and `print.summary.im` prints this information in a comprehensible format.

In normal usage, `print.summary.im` is invoked implicitly when the user calls `summary.im` without assigning its value to anything. See the examples.

The information extracted by `summary.im` includes

range The range of the image values.

mean The mean of the image values.

integral The “integral” of the image values, calculated as the sum of the image values multiplied by the area of one pixel.

dim The dimensions of the pixel array: `dim[1]` is the number of rows in the array, corresponding to the `y` coordinate.

Value

`summary.im` returns an object of class "summary.im", while `print.summary.im` returns NULL.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

Examples

```
# make an image
X <- as.im(function(x,y) {x^2}, unit.square())
# summarize it
summary(X)
# save the summary
s <- summary(X)
# print it
print(X)
s
# extract stuff
X$dim
X$range
X$integral
```

summary.listof *Summary of a List of Things*

Description

Prints a useful summary of each item in a list of things.

Usage

```
## S3 method for class 'listof'
summary(object, ...)
```

Arguments

object	An object of class "listof".
...	Ignored.

Details

This is a method for the generic function [summary](#).

An object of the class "listof" is effectively a list of things which are all of the same class.

This function extracts a useful summary of each of the items in the list.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[summary](#), [plot.listof](#)

Examples

```
x <- list(A=runif(10), B=runif(10), C=runif(10))
class(x) <- c("listof", class(x))
summary(x)
```

summary.owin *Summary of a Spatial Window*

Description

Prints a useful description of a window object.

Usage

```
## S3 method for class 'owin'  
summary(object, ...)
```

Arguments

object	Window (object of class "owin").
...	Ignored.

Details

A useful description of the window object is printed.

This is a method for the generic function [summary](#).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[summary](#), [summary.ppp](#), [print.owin](#)

Examples

```
summary(owin()) # the unit square  
  
data(demopat)  
W <- demopat$window # weird polygonal window  
summary(W) # describes it  
  
summary(as.mask(W)) # demonstrates current pixel resolution
```

summary.ppm*Summarizing a Fitted Point Process Model*

Description

`summary` method for class "ppm".

Usage

```
## S3 method for class 'ppm'
summary(object, ..., quick=FALSE)
## S3 method for class 'summary.ppm'
print(x, ...)
```

Arguments

<code>object</code>	A fitted point process model.
<code>...</code>	Ignored.
<code>quick</code>	Logical flag controlling the scope of the summary.
<code>x</code>	Object of class "summary.ppm" as returned by <code>summary.ppm</code> .

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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

Examples

```

# invent some data
X <- rpoispp(42)
# fit a model to it
fit <- ppm(X, ~x, Strauss(r=0.1))
# summarize the fitted model
summary(fit)
# 'quick' option
summary(fit, quick=TRUE)

# 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

## Not run:
# multitype pattern
data(demopat)
fit <- ppm(demopat, ~marks, Poisson())
summary(fit)

## End(Not run)

# model with external covariates
fitX <- ppm(X, ~Z, covariates=list(Z=function(x,y){x+y}))
summary(fitX)

```

summary.ppp

Summary of a Point Pattern Dataset

Description

Prints a useful summary of a point pattern dataset.

Usage

```
## S3 method for class 'ppp'
summary(object, ..., checkdup=TRUE)
```

Arguments

- | | |
|----------|---|
| object | Point pattern (object of class "ppp"). |
| ... | Ignored. |
| checkdup | Logical value indicating whether to check for the presence of duplicate points. |

Details

A useful summary of the point pattern object is printed.

This is a method for the generic function [summary](#).

If `checkdup=TRUE`, the pattern will be checked for the presence of duplicate points, using [duplicated.ppp](#). This can be time-consuming if the pattern contains many points, so the checking can be disabled by setting `checkdup=FALSE`.

If the point pattern was generated by simulation using [rmh](#), the parameters of the algorithm are printed.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[summary](#), [summary.owin](#), [print.ppp](#)

Examples

```
summary(cells) # plain vanilla point pattern

# multitype point pattern

summary(lansing) # tabulates frequencies of each mark

# numeric marks

summary(longleaf) # prints summary.default(x$marks)

# weird polygonal window
summary(demopat) # describes it
```

Description

Prints a useful summary of a line segment pattern dataset.

Usage

```
## S3 method for class 'psp'
summary(object, ...)
```

Arguments

object	Line segment pattern (object of class "psp").
...	Ignored.

Details

A useful summary of the line segment pattern object is printed.

This is a method for the generic function [summary](#).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[summary](#), [summary.owin](#), [print.psp](#)

Examples

```
a <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
summary(a) # describes it
```

summary.quad

Summarizing a Quadrature Scheme

Description

`summary` method for class "quad".

Usage

```
## S3 method for class 'quad'
summary(object, ..., checkdup=FALSE)
## S3 method for class 'summary.quad'
print(x, ..., dp=3)
```

Arguments

object	A quadrature scheme.
...	Ignored.
checkdup	Logical value indicating whether to test for duplicated points.
dp	Number of significant digits to print.
x	Object of class "summary.quad" returned by <code>summary.quad</code> .

Details

This is a method for the generic [summary](#) for the class "quad". An object of class "quad" describes a quadrature scheme, used to fit a point process model. See [quad.object](#)) for details of this class. `summary.quad` extracts information about the quadrature scheme, and `print.summary.quad` prints this information in a comprehensible format.

In normal usage, `print.summary.quad` is invoked implicitly when the user calls `summary.quad` without assigning its value to anything. See the examples.

Value

`summary.quad` returns an object of class "summary.quad", while `print.summary.quad` returns NULL.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

Examples

```
# make a quadrature scheme
Q <- quadscheme(rpoispp(42))
# summarize it
summary(Q)
# save the summary
s <- summary(Q)
# print it
print(s)
s
# extract total quadrature weight
s$w$all$sum
```

summary.splitppp

*Summary of a Split Point Pattern***Description**

Prints a useful summary of a split point pattern.

Usage

```
## S3 method for class 'splitppp'
summary(object, ...)
```

Arguments

- | | |
|--------|--|
| object | Split point pattern (object of class "splitppp", effectively a list of point patterns, usually created by split.ppp). |
| ... | Ignored. |

Details

This is a method for the generic function [summary](#).

An object of the class "splitppp" is effectively a list of point patterns (objects of class "ppp") representing different sub-patterns of an original point pattern.

This function extracts a useful summary of each of the sub-patterns.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[summary](#), [split](#), [split.ppp](#)

Examples

```
data(amacrine)      # multitype point pattern
summary(split(amacrine))
```

sumouter

Compute Quadratic Forms

Description

Calculates certain quadratic forms of matrices.

Usage

```
sumouter(x, w)
quadform(x, v)
```

Arguments

- x A matrix, whose rows are the vectors in the quadratic form.
- w Vector of weights
- v Matrix determining the quadratic form

Details

The matrix x will be interpreted as a collection of row vectors. The command sumouter computes the sum of the outer products of these vectors, weighted by the entries of w:

$$M = \sum_i w_i x_i x_i^\top$$

where the sum is over all rows of x (after removing any rows containing NA or other non-finite values). The result is a $p \times p$ matrix where $p = \text{ncol}(x)$.

The command quadform evaluates the quadratic form, defined by the matrix v, for each of the row vectors of x:

$$y_i = x_i V x_i^\top$$

The result y is a numeric vector of length n where $n = \text{nrow}(x)$. If $x[i,]$ contains NA or other non-finite values, then $y[i] = \text{NA}$.

Value

A vector or matrix.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

Examples

```

x <- matrix(1:12, 4, 3)
dimnames(x) <- list(c("Wilma", "Fred", "Barney", "Betty"), letters[1:3])
x

w <- 4:1
sumouter(x, w)
v <- matrix(1, 3, 3)
quadform(x, v)

# See what happens with NA's
x[3,2] <- NA
sumouter(x, w)
quadform(x, v)

```

superimpose

Superimpose Several Geometric Patterns

Description

Superimpose any number of point patterns or line segment patterns.

Usage

```

superimpose(...)
## S3 method for class 'ppp'
superimpose(..., W=NULL, check=TRUE)
## S3 method for class 'psp'
superimpose(..., W=NULL, check=TRUE)
## Default S3 method:
superimpose(..., W=NULL, check=TRUE)

```

Arguments

...	Any number of arguments, each of which represents either a point pattern or a line segment pattern.
W	Optional. Data determining the window for the resulting pattern. Either a window (object of class "owin", or something acceptable to as.owin), or a function which returns a window, or one of the strings "convex", "rectangle", "bbox" or "none".
check	Logical value (passed to ppp or psp as appropriate) determining whether to check the geometrical validity of the resulting pattern.

Details

This function is used to superimpose several geometric patterns of the same kind, producing a single pattern of the same kind.

The function `superimpose` is generic, with methods for the class `ppp` of point patterns, the class `psp` of line segment patterns, and a default method. The dispatch to a method is determined by the class of the first argument in `...`.

- **default:** If the first argument is *not* an object of class `ppp` or `psp`, then the default method `superimpose.default` is executed. All arguments in `...` must have components `x` and `y` representing spatial coordinates. They may include objects of class `ppp`. The result will be either a `list(x,y)` or a point pattern (objects of class `ppp`) as explained below.
- **ppp:** If the first `...` argument is an object of class `ppp` then the method `superimpose.ppp` is executed. All arguments in `...` must be either `ppp` objects or lists with components `x` and `y`. The result will be an object of class `ppp`.
- **psp:** If the first `...` argument is an object of class `psp` then the `psp` method is dispatched and all `...` arguments must be `psp` objects. The result is a `psp` object.

The patterns are *not* required to have the same window of observation.

The window for the superimposed pattern is controlled by the argument `W`.

- If `W` is a window (object of class "`W`" or something acceptable to `as.owin`) then this determines the window for the superimposed pattern.
- If `W` is `NULL`, or the character string "`none`", then windows are extracted from the geometric patterns, as follows. For `superimpose.psp`, all arguments `...` are line segment patterns (objects of class "`psp`"); their observation windows are extracted; the union of these windows is computed; and this union is taken to be the window for the superimposed pattern. For `superimpose.ppp` and `superimpose.default`, the arguments `...` are inspected, and any arguments which are point patterns (objects of class "`ppp`") are selected; their observation windows are extracted, and the union of these windows is taken to be the window for the superimposed point pattern. For `superimpose.default` if none of the arguments is of class "`ppp`" then no window is computed and the result of `superimpose` is a `list(x,y)`.
- If `W` is one of the strings "`convex`", "`rectangle`" or "`bbox`" then a window for the superimposed pattern is computed from the coordinates of the points or the line segments as follows.
 - `"bbox"`: the bounding box of the points or line segments (see `bounding.box.xy`);
 - `"convex"`: the Ripley-Rasson estimator of a convex window (see `ripras`);
 - `"rectangle"`: the Ripley-Rasson estimator of a rectangular window (using `ripras` with argument `shape="rectangle"`).
- If `W` is a function, then this function is used to compute a window for the superimposed pattern from the coordinates of the points or the line segments. The function should accept input of the form `list(x,y)` and is expected to return an object of class "`owin`". Examples of such functions are `ripras` and `bounding.box.xy`.

The arguments `...` may be *marked* patterns. The marks of each component pattern must have the same format. Numeric and character marks may be "mixed". If there is such mixing then the numeric marks are coerced to character in the combining process. If the mark structures are all data frames, then these data frames must have the same number of columns and identical column names.

If the arguments `...` are given in the form `name=value`, then the names will be used as an extra column of marks attached to the elements of the corresponding patterns.

Value

For `superimpose.ppp`, a point pattern (object of class "`ppp`"). For `superimpose.default`, either a point pattern (object of class "`ppp`") or a `list(x,y)`. For `superimpose.psp`, a line segment pattern (object of class "`psp`").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[concatxy](#), [quadscheme](#).

Examples

```
# superimposing point patterns
p1 <- runifrect(30)
p2 <- runifrect(42)
s1 <- superimpose(p1,p2) # Unmarked pattern.
p3 <- list(x=rnorm(20),y=rnorm(20))
s2 <- superimpose(p3,p2,p1) # Default method gets called.
s2a <- superimpose(p1,p2,p3) # Same as s2 except for order of points.
s3 <- superimpose(clyde=p1,irving=p2) # Marked pattern; marks a factor
# with levels "clyde" and "irving";
# warning given.
marks(p1) <- factor(sample(LETTERS[1:3],30,TRUE))
marks(p2) <- factor(sample(LETTERS[1:3],42,TRUE))
s5 <- superimpose(clyde=p1,irving=p2) # Marked pattern with extra column
marks(p2) <- data.frame(a=marks(p2),b=runif(42))
s6 <- try(superimpose(p1,p2)) # Gives an error.
marks(p1) <- data.frame(a=marks(p1),b=1:30)
s7 <- superimpose(p1,p2) # O.K.

# how to make a 2-type point pattern with types "a" and "b"
u <- superimpose(a = rpoispp(10), b = rpoispp(20))

# how to make a 2-type point pattern with types 1 and 2
u <- superimpose("1" = rpoispp(10), "2" = rpoispp(20))

# superimposing line segment patterns
X <- rpoisline(10)
Y <- as.psp(matrix(runif(40), 10, 4), window=owin())
Z <- superimpose(X, Y)

# being unreasonable
## Not run:
crud <- superimpose(p1,p2,X,Y) # Gives an error, of course!

## End(Not run)
```

Description

The data give the locations of pine saplings in a Swedish forest.

Usage

`data(swedishpines)`

Format

An object of class "ppp" representing the point pattern of tree locations in a 10 x 10 metre square. Cartesian coordinates are in decimetres (multiples of 0.1 metre).

See [ppp.object](#) for details of the format of a point pattern object.

Note

For previous analyses see Ripley (1981, pp. 172-175), Venables and Ripley (1997, p. 483), Baddeley and Turner (2000).

Source

Strand (1975), Ripley (1981)

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.
- Strand, L. (1972). A model for stand growth. *IUFRO Third Conference Advisory Group of Forest Statisticians*, INRA, Institut National de la Recherche Agronomique, Paris. Pages 207–216.
- Venables, W.N. and Ripley, B.D. (1997) *Modern applied statistics with S-PLUS*. Second edition. Springer Verlag.

tess

Create a Tessellation

Description

Creates an object of class "tess" representing a tessellation of a spatial region.

Usage

```
tess(..., xgrid = NULL, ygrid = NULL, tiles = NULL, image = NULL,
      window=NULL, keepempty=FALSE)
```

Arguments

...	Ignored.
xgrid, ygrid	Cartesian coordinates of vertical and horizontal lines determining a grid of rectangles. Incompatible with other arguments.
tiles	List of tiles in the tessellation. A list, each of whose elements is a window (object of class "owin"). Incompatible with other arguments.
image	Pixel image which specifies the tessellation. Incompatible with other arguments.
window	Optional. The spatial region which is tessellated (i.e. the union of all the tiles). An object of class "owin".
keepempty	Logical flag indicating whether empty tiles should be retained or deleted.

Details

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

Three types of tessellation are supported:

rectangular: tiles are rectangles, with sides parallel to the x and y axes. They may or may not have equal size and shape. The arguments `xgrid` and `ygrid` determine the positions of the vertical and horizontal grid lines, respectively. (See [quadrats](#) for another way to do this.)

tile list: tiles are arbitrary spatial regions. The argument `tiles` is a list of these tiles, which are objects of class "owin".

pixel image: Tiles are subsets of a fine grid of pixels. The argument `image` is a pixel image (object of class "im") with factor values. Each level of the factor represents a different tile of the tessellation. The pixels that have a particular value of the factor constitute a tile.

The optional argument `window` specifies the spatial region formed by the union of all the tiles. In other words it specifies the spatial region that is divided into tiles by the tessellation. If this argument is missing or `NULL`, it will be determined by computing the set union of all the tiles. This is a time-consuming computation. For efficiency it is advisable to specify the window. Note that the validity of the window will not be checked.

Empty tiles may occur, either because one of the entries in the list `tiles` is an empty window, or because one of the levels of the factor-valued pixel image `image` does not occur in the pixel data. When `keepempty=TRUE`, empty tiles are permitted. When `keepempty=FALSE` (the default), tiles are not allowed to be empty, and any empty tiles will be removed from the tessellation.

There are methods for `print`, `plot`, `[` and `[<-` for tessellations. Use `tiles` to extract the list of tiles in a tessellation, or `tile.areas` to compute their areas.

Tessellations can be used to classify the points of a point pattern, in `split.ppp`, `cut.ppp` and `by.ppp`.

Value

An object of class "tess" representing the tessellation.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[plot.tess](#), [\[.tess](#), [as.tess](#), [tiles](#), [intersect.tess](#), [split.ppp](#), [cut.ppp](#), [by.ppp](#), [quadrats](#), [bdist.tiles](#), [tile.areas](#).

Examples

```
A <- tess(xgrid=0:4,ygrid=0:4)
A
B <- A[c(1, 2, 5, 7, 9)]
B
v <- as.im(function(x,y){factor(round(5 * (x^2 + y^2)))}, W=owin())
levels(v) <- letters[seq(length(levels(v)))]
E <- tess(image=v)
E
```

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, sigma2=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 <code>optim</code> 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 Moller 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 .

The theoretical K -function of the Thomas process is

$$K(r) = \pi r^2 + \frac{1}{\kappa} \left(1 - \exp\left(-\frac{r^2}{4\sigma^2}\right)\right).$$

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:

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

Author(s)

Rasmus Waagepetersen <rwm@math.auc.dk> Adapted for [spatstat](#) by Adrian Baddeley <Adrian.Baddeley@csiro.au>
<http://www.maths.uwa.edu.au/~adrian/>

References

- Moller, 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](#), [matclust.estK](#), [mincontrast](#), [Kest](#), [rThomas](#) to simulate the fitted model.

Examples

```
data(redwood)
u <- thomas.estK(redwood, c(kappa=10, sigma2=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,sigma2=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 Moller 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 .

The theoretical pair correlation function of the Thomas process is

$$g(r) = 1 + \frac{1}{4\pi\kappa\sigma^2} \exp\left(-\frac{r^2}{4\sigma^2}\right).$$

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:

<code>par</code>	Vector of fitted parameter values.
<code>fit</code>	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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

References

- Moller, 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

[thomas.estK](#) [mincontrast](#), [pcf](#), [rThomas](#) to simulate the fitted model.

Examples

```
data(redwood)
u <- thomas.estpcf(redwood, c(kappa=10, sigma2=0.1))
u
plot(u, legendpos="topright")
u2 <- thomas.estpcf(redwood, c(kappa=10, sigma2=0.1),
pcfargs=list(stoyan=0.12))
```

tile.areas*Compute Areas of Tiles in a Tessellation*

Description

Computes the area of each tile in a tessellation.

Usage

```
tile.areas(x)
```

Arguments

x A tessellation (object of class "tess").

Details

A tessellation is a collection of disjoint spatial regions (called *tiles*) that fit together to form a larger spatial region. See [tess](#).

This command computes the area of each of the tiles that make up the tessellation x. The result is a numeric vector in the same order as the tiles would be listed by [tiles\(x\)](#).

Value

A numeric vector.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[tess](#), [tiles](#)

Examples

```
A <- tess(xgrid=0:2,ygrid=0:2)
tile.areas(A)
v <- as.im(function(x,y){factor(round(x^2 + y^2))}, W=owin())
E <- tess(image=v)
tile.areas(E)
```

tiles	<i>Extract List of Tiles in a Tessellation</i>
--------------	--

Description

Extracts a list of the tiles that make up a tessellation.

Usage

```
tiles(x)
```

Arguments

x A tessellation (object of class "tess").

Details

A tessellation is a collection of disjoint spatial regions (called *tiles*) that fit together to form a larger spatial region. See [tess](#).

The tiles that make up the tessellation **x** are returned in a list.

Value

A list of windows (objects of class "owin").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[tess](#), [tile.areas](#)

Examples

```
A <- tess(xgrid=0:2,ygrid=0:2)
tiles(A)
v <- as.im(function(x,y){factor(round(x^2 + y^2))}, W=owin())
E <- tess(image=v)
tiles(E)
```

transect.im*Pixel Values Along a Transect*

Description

Extract the pixel values of a pixel image at each point along a linear transect.

Usage

```
transect.im(X, ..., from="bottomleft", to="topright",
            click=FALSE, add=FALSE)
```

Arguments

X	A pixel image (object of class "im").
...	Ignored.
from,to	Optional. Start point and end point of the transect. Pairs of (x, y) coordinates in a format acceptable to <code>xy.coords</code> , or keywords "bottom", "left", "top", "right", "bottomleft" etc.
click	Optional. Logical value. If TRUE, the linear transect is determined interactively by the user, who clicks two points on the current plot.
add	Logical. If click=TRUE, this argument determines whether to perform interactive tasks on the current plot (add=TRUE) or to start by plotting X (add=FALSE).

Details

The pixel values of the image X along a line segment will be extracted. The result is a function table ("fv" object) which can be plotted directly.

If `click=TRUE`, then the user is prompted to click two points on the plot of X. These endpoints define the transect.

Otherwise, the transect is defined by the endpoints `from` and `to`. The default is a diagonal transect from bottom left to top right of the frame.

Value

An object of class "fv" which can be plotted.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`im`

Examples

```
Z <- density(redwood)
plot(transect.im(Z))
## Not run:
plot(transect.im(Z, click=TRUE))

## End(Not run)
```

trim.rectangle *Cut margins from rectangle*

Description

Trims a margin from a rectangle.

Usage

```
trim.rectangle(W, xmargin=0, ymargin=xmargin)
```

Arguments

W	A window (object of class "owin"). Must be of type "rectangle".
xmargin	Width of horizontal margin to be trimmed. A single nonnegative number, or a vector of length 2 indicating margins of unequal width at left and right.
ymargin	Height of vertical margin to be trimmed. A single nonnegative number, or a vector of length 2 indicating margins of unequal width at bottom and top.

Details

This is a simple convenience function to trim off a margin of specified width and height from each side of a rectangular window. Unequal margins can also be trimmed.

Value

Another object of class "owin" representing the window after margins are trimmed.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[owin.object](#)

Examples

```
w <- square(10)
# trim a margin of width 1 from all four sides
square9 <- trim.rectangle(w, 1)

# trim margin of width 3 from the right side
# and margin of height 4 from top edge.
v <- trim.rectangle(w, c(0,3), c(0,4))
```

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.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

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

See Also

[Triplets](#) to create the triplet interaction process structure.

Other families: [pairwise.family](#), [pairsat.family](#), [inforder.family](#), [ord.family](#).

Triplets *The Triplet Point Process Model*

Description

Creates an instance of Geyer's triplet interaction point process model which can then be fitted to point pattern data.

Usage

`Triplets(r)`

Arguments

`r` The interaction radius of the Triplets process

Details

The (stationary) Geyer triplet process (Geyer, 1999) with interaction radius r and parameters β and γ is the point process in which each point contributes a factor β to the probability density of the point pattern, and each triplet of close points contributes a factor γ to the density. A triplet of close points is a group of 3 points, each pair of which is closer than r units apart.

Thus the probability density is

$$f(x_1, \dots, x_n) = \alpha \beta^{n(x)} \gamma^{s(x)}$$

where x_1, \dots, 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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

Geyer, C.J. (1999) Likelihood Inference for Spatial Point Processes. Chapter 3 in O.E. Barndorff-Nielsen, W.S. Kendall and M.N.M. Van Lieshout (eds) *Stochastic Geometry: Likelihood and Computation*, Chapman and Hall / CRC, Monographs on Statistics and Applied Probability, number 80. Pages 79–140.

See Also

`ppm`, `triplet.family`, `ppm.object`

Examples

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

## Not run:
ppm(cells, ~1, Triplets(r=0.1))
```

```
# fit the stationary Triplets process to 'cells'

## End(Not run)

ppm(cells, ~polynom(x,y,3), Triplets(r=0.1))
# fit a nonstationary Triplets process with log-cubic polynomial trend
```

Tstat*Third order summary statistic***Description**

Computes the third order summary statistic $T(r)$ of a spatial point pattern.

Usage

```
Tstat(X, ..., r = NULL, rmax = NULL,
      correction = c("border", "translate"), ratio = FALSE, verbose=TRUE)
```

Arguments

<code>X</code>	The observed point pattern, from which an estimate of $T(r)$ will be computed. An object of class "ppp", or data in any format acceptable to as.ppp() .
<code>...</code>	Ignored.
<code>r</code>	Optional. Vector of values for the argument r at which $T(r)$ should be evaluated. Users are advised <i>not</i> to specify this argument; there is a sensible default.
<code>rmax</code>	Optional. Numeric. The maximum value of r for which $T(r)$ should be estimated.
<code>correction</code>	Optional. A character vector containing any selection of the options "none", "border", "bord.modif", "translate", or "best". It specifies the edge correction(s) to be applied.
<code>ratio</code>	Logical. If TRUE, the numerator and denominator of each edge-corrected estimate will also be saved, for use in analysing replicated point patterns.
<code>verbose</code>	Logical. If TRUE, an estimate of the computation time is printed.

Details

This command calculates the third-order summary statistic $T(r)$ for a spatial point patterns, defined by Schladitz and Baddeley (2000).

The definition of $T(r)$ is similar to the definition of Ripley's K function $K(r)$, except that $K(r)$ counts pairs of points while $T(r)$ counts triples of points. Essentially $T(r)$ is a rescaled cumulative distribution function of the diameters of triangles in the point pattern. The diameter of a triangle is the length of its longest side.

Value

An object of class "fv", see [fv.object](#), which can be plotted directly using [plot.fv](#).

Computation time

If the number of points is large, the algorithm can take a very long time to inspect all possible triangles. A rough estimate of the total computation time will be printed at the beginning of the calculation. If this estimate seems very large, stop the calculation using the user interrupt signal, and call `Tstat` again, using `rmax` to restrict the range of `r` values, thus reducing the number of triangles to be inspected.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

References

Schladitz, K. and Baddeley, A. (2000) A third order point process characteristic. *Scandinavian Journal of Statistics* **27** (2000) 657–671.

See Also

[Kest](#)

Examples

```
plot(Tstat(redwood))
```

`union.quad`

Union of Data and Dummy Points

Description

Combines the data and dummy points of a quadrature scheme into a single point pattern.

Usage

```
union.quad(Q)
```

Arguments

`Q` A quadrature scheme (an object of class "quad").

Details

The argument `Q` should be a quadrature scheme (an object of class "quad", see [quad.object](#) for details).

This function combines the data and dummy points of `Q` into a single point pattern. If either the data or the dummy points are marked, the result is a marked point pattern.

The function [as.ppp](#) will perform the same task.

Value

A point pattern (of class "ppp").

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[quad.object](#), [as.ppp](#)

Examples

```
data(simdat)
Q <- quadscheme(simdat, default.dummy(simdat))
U <- union.quad(Q)
## Not run: plot(U)
# equivalent:
U <- as.ppp(Q)
```

unique.ppp

Extract Unique Points from a Spatial Point Pattern

Description

Removes any points that are identical to other points in a spatial point pattern.

Usage

```
## S3 method for class 'ppp'
unique(x, ...)
```

Arguments

x	A spatial point pattern (object of class "ppp").
...	Ignored.

Details

This is a method for the generic function `unique` for point pattern datasets (of class "ppp", see [ppp.object](#)).

Two points in a point pattern are deemed to be identical if their x, y coordinates are the same, and their marks are also the same (if they carry marks). The Examples section illustrates how it is possible for a point pattern to contain a pair of identical points.

This function removes duplicate points in `x`, and returns a point pattern.

Value

Another point pattern object.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppp.object](#), [duplicated.ppp](#), [multiplicity.ppp](#)

Examples

```
X <- ppp(c(1,1,0.5), c(2,2,1), window=square(3))
unique(X)
```

unitname	<i>Name for Unit of Length</i>
----------	--------------------------------

Description

Inspect or change the name of the unit of length in a spatial dataset.

Usage

```
unitname(x)
## S3 method for class 'im'
unitname(x)
## S3 method for class 'kppm'
unitname(x)
## S3 method for class 'minconfit'
unitname(x)
## S3 method for class 'owin'
unitname(x)
## S3 method for class 'ppm'
unitname(x)
## S3 method for class 'ppp'
unitname(x)
## S3 method for class 'psp'
unitname(x)
## S3 method for class 'quad'
unitname(x)
## S3 method for class 'slrm'
unitname(x)
unitname(x) <- value
## S3 replacement method for class 'im'
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 'owin'
unitname(x) <- value
## S3 replacement method for class 'ppm'
unitname(x) <- value
## S3 replacement method for class 'ppp'
unitname(x) <- value
## S3 replacement method for class 'psp'
unitname(x) <- value
```

```
## S3 replacement method for class 'quad'
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 quadrature scheme (object of class "quad"), or a fitted point process model (object of class "ppm" or "kppm" or "slrm" 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 "units" containing the name of the unit of length in `x`. There are methods for `print` and `summary`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

`rescale`, `owin`, `ppp`

Examples

```
X <- runifpoint(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)
```

unmark*Remove Marks***Description**

Remove the mark information from a spatial dataset.

Usage

```
unmark(X)
## S3 method for class 'ppp'
unmark(X)
## S3 method for class 'splitppp'
unmark(X)
## S3 method for class 'psp'
unmark(X)
## S3 method for class 'ppx'
unmark(X)
```

Arguments

X A point pattern (object of class "ppp"), a split point pattern (object of class "splitppp"), a line segment pattern (object of class "psp") or a multidimensional space-time point pattern (object of class "ppx").

Details

A ‘mark’ is a value attached to each point in a spatial point pattern, or attached to each line segment in a line segment pattern, etc.

The function `unmark` is a simple way to remove the marks from such a dataset.

Value

An object of the same class as X with any mark information deleted.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppp.object](#), [psp.object](#)

Examples

```
data(lansing)
hicks <- lansing[lansing$marks == "hickory", ]
## Not run:
plot(hicks) # still a marked point pattern, but only 1 value of marks
plot(unmark(hicks)) # unmarked

## End(Not run)
```

unnormdensity	<i>Weighted kernel smoother</i>
---------------	---------------------------------

Description

An unnormalised version of kernel density estimation where the weights are not required to sum to 1. The weights may be positive, negative or zero.

Usage

```
unnormdensity(x, ..., weights = NULL)
```

Arguments

x	Numeric vector of data
...	Arguments passed to <code>density.default</code> . Arguments must be <i>named</i> . ‘
weights	Optional numeric vector of weights for the data.

Details

This is an alternative to the standard R kernel density estimation function `density.default`.

The standard `density.default` requires the weights to be nonnegative numbers that add up to 1, and returns a probability density (a function that integrates to 1).

This function `unnormdensity` does not impose any requirement on the weights except that they be finite. Individual weights may be positive, negative or zero. The result is a function that does not necessarily integrate to 1 and may be negative. The result is the convolution of the kernel k with the weighted data,

$$f(x) = \sum_i w_i k(x - x_i)$$

where x_i are the data points and w_i are the weights.

The algorithm first selects the kernel bandwidth by applying `density.default` to the data x with normalised, positive weight vector $w = \text{abs}(\text{weights})/\text{sum}(\text{abs}(\text{weights}))$ and extracting the selected bandwidth. Then the result is computed by applying `density.default` to x twice using the normalised positive and negative parts of the weights.

Note that the arguments \dots must be passed by name, i.e. in the form (name=value). Arguments that do not match an argument of `density.default` will be ignored *silently*.

Value

Object of class "density" as described in `density.default`.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

`density.default`

Examples

```
d <- unnormdensity(1:3, weights=c(-1,0,1))
if(interactive()) plot(d)
```

update.kppm

Update a Fitted Cluster Point Process Model

Description

update method for class "kppm".

Usage

```
## S3 method for class 'kppm'
update(object, trend = ~1, ..., clusters = NULL)
```

Arguments

- | | |
|----------|---|
| object | Fitted cluster point process model. An object of class "kppm", obtained from kppm . |
| trend | A formula without a left hand side, determining the form of the intensity of the model. |
| ... | Other arguments passed to kppm . |
| clusters | The type of cluster mechanism. A character string. See 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.

The argument trend determines the formula for the intensity in the model. It should be an R formula without a left hand side. It may include the symbols . and + or - to specify addition or deletion of terms in the current model formula, as shown in the Examples below.

The model is refitted using [kppm](#).

Value

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

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[kppm](#), [plot.kppm](#), [predict.kppm](#), [simulate.kppm](#), [methods.kppm](#), [vcov.kppm](#)

Examples

```
data(redwood)
fit <- kppm(redwood, ~1, "Thomas")
fitx <- update(fit, ~ . + x)
fitM <- update(fit, clusters="MatClust")
```

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=parent.frame())
```

Arguments

<code>object</code>	An existing fitted point process model, typically produced by ppm .
<code>...</code>	Arguments to be updated in the new call to ppm .
<code>fixdummy</code>	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.
<code>use.internal</code>	Optional. Logical flag indicating whether the model should be refitted using the internally saved data (<code>use.internal=TRUE</code>) or by re-evaluating these data in the current frame (<code>use.internal=FALSE</code>).
<code>envir</code>	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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

Examples

```
data(nztrees)
data(cells)

# fit the stationary Poisson process
fit <- ppm(nztrees, ~ 1)

# fit a nonstationary Poisson process
fitP <- update(fit, trend=~x)
fitP <- update(fit, ~x)

# change the trend formula: add another term to the trend
```

```

fitPxy <- update(fitP, ~ . + y)
# change the trend formula: remove the x variable
fitPy <- update(fitPxy, ~ . - x)

# fit a stationary Strauss process
fitS <- update(fit, interaction=Strauss(13))
fitS <- update(fit, Strauss(13))

# refit using a different edge correction
fitS <- update(fitS, correction="isotropic")

# re-fit the model to a subset
# of the original point pattern
nzw <- owin(c(0,148),c(0,95))
nzsub <- nztrees[,nzw]
fut <- update(fitS, Q=nzsub)
fut <- update(fitS, nzsub)

# WARNING: the point pattern argument is called 'Q'

ranfit <- ppm(rpoispp(42), ~1, Poisson())
ranfit
# different random data!
update(ranfit)
# the original data
update(ranfit, use.internal=TRUE)

```

update.rmhcontrol*Update Control Parameters of Metropolis-Hastings Algorithm***Description**

update method for class "rmhcontrol".

Usage

```
## S3 method for class 'rmhcontrol'
update(object, ...)
```

Arguments

- | | |
|--------|---|
| object | Object of class "rmhcontrol" containing control parameters for a Metropolis-Hastings algorithm. |
| ... | Arguments to be updated in the new call to rmhcontrol . |

Details

This is a method for the generic function [update](#) for the class "rmhcontrol". An object of class "rmhcontrol" describes a set of control parameters for the Metropolis-Hastings simulation algorithm. See [rmhcontrol](#)).

`update.rmhcontrol` will modify the parameters specified by `object` according to the new arguments given.

Value

Another object of class "rmhcontrol".

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

Examples

```
a <- rmhcontrol(expand=1)
update(a, expand=2)
```

urkiola

*Urkiola Woods Point Pattern***Description**

Locations of birch (*Betula celtiberica*) and oak (*Quercus robur*) trees in a secondary wood in Urkiola Natural Park (Basque country, northern Spain). They are part of a more extensive dataset collected and analysed by Laskurain (2008). The coordinates of the trees are given in meters.

Usage

```
data(urkiola)
```

Format

An object of class "ppp" representing the point pattern of tree locations. Entries include

- x** Cartesian x-coordinate of tree
- y** Cartesian y-coordinate of tree
- marks** factor indicating species of each tree

The levels of **marks** are birch and oak. See [ppp.object](#) for details of the format of a ppp object.

Source

N.A. Laskurain. Kindly formatted and communicated by M. de la Cruz Rot

References

Laskurain, N. A. (2008) *Dinámica espacio-temporal de un bosque secundario en el Parque Natural de Urkiola (Bizkaia)*. Tesis Doctoral. Universidad del País Vasco /Euskal Herriko Unibertsitatea.

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

```
valid.ppm(object)
```

Arguments

object Fitted point process model (object of class "ppm").

Details

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

Logical.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[ppm](#), [project.ppm](#)

Examples

```
fit1 <- ppm(cells, ~1, Strauss(0.1))
valid.ppm(fit1)
fit2 <- ppm(redwood, ~1, Strauss(0.1))
valid.ppm(fit2)
```

varblock

Estimate Variance of Summary Statistic by Subdivision

Description

This command estimates the variance of any summary statistic (such as the K -function) by spatial subdivision of a single point pattern dataset.

Usage

```
varblock(X, fun = Kest, blocks = quadrats(X, nx = nx, ny = ny), ...,
         nx = 3, ny = nx)
```

Arguments

X	Point pattern dataset (object of class "ppp").
fun	Function that computes the summary statistic.
blocks	Optional. A tessellation that specifies the division of the space into blocks.
...	Arguments passed to fun.
nx, ny	Optional. Number of rectangular blocks in the x and y directions. Incompatible with blocks.

Details

This command computes an estimate of the variance of the summary statistic $\text{fun}(X)$ from a single point pattern dataset X using a subdivision method. It can be used to plot **confidence intervals** for the true value of a summary function such as the K -function.

The window containing X is divided into pieces by an $nx * ny$ array of rectangles (or is divided into pieces of more general shape, according to the argument `blocks` if it is present). The summary statistic `fun` is applied to each of the corresponding sub-patterns of X as described below. Then the pointwise sample mean, sample variance and sample standard deviation of these summary statistics are computed. The two-standard-deviation confidence intervals are computed.

The variance is estimated by equation (4.21) of Diggle (2003, page 52). This assumes that the point pattern X is stationary. For further details see Diggle (2003, pp 52–53).

The estimate of the summary statistic from each block is computed as follows. For most functions `fun`, the estimate from block B is computed by finding the subset of X consisting of points that fall inside B , and applying `fun` to these points, by calling `fun(X[B])`.

However if `fun` is the K -function `Kest`, or any function which has an argument called `domain`, the estimate for each block B is computed by calling `fun(X, domain=B)`. In the case of the K -function this means that the estimate from block B is computed by counting pairs of points in which the *first* point lies in B , while the second point may lie anywhere.

Value

A function value table (object of class "fv") that contains the result of `fun(X)` as well as the sample mean, sample variance and sample standard deviation of the block estimates, together with the upper and lower two-standard-deviation confidence limits.

Errors

If the blocks are too small, there may be insufficient data in some blocks, and the function `fun` may report an error. If this happens, you need to take larger blocks.

An error message about incompatibility may occur. The different function estimates may be incompatible in some cases, for example, because they use different default edge corrections (typically because the tiles of the tessellation are not the same kind of geometric object as the window of `X`, or because the default edge correction depends on the number of points). To prevent this, specify the choice of edge correction, in the `correction` argument to `fun`, if it has one. Some edge corrections are only available if you have set `spatstat.options(gpclib=TRUE)`.

An alternative to `varblock` is Loh's mark bootstrap [lohboot](#).

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

References

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

See Also

[tess](#), [quadrats](#) for basic manipulation.

[lohboot](#) for an alternative bootstrap technique.

Examples

```
v <- varblock(amacrine, Kest, nx=4, ny=2)
v <- varblock(amacrine, Kcross, nx=4, ny=2)
if(interactive()) plot(v, iso ~ r, shade=c("hiiso", "loiso"))
```

`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,eta=1), nu.ker = -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.ker	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, 2011) 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 (2011). 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 (2011).

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:

- | | |
|------------------|---|
| <code>par</code> | Vector of fitted parameter values. |
| <code>fit</code> | Function value table (object of class " <code>fv</code> ") containing the observed values of the summary statistic (<code>observed</code>) 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@csiro.au>
<http://www.maths.uwa.edu.au/~adrian/>

References

- Jalilian, A., Guan, Y. and Waagepetersen, R. (2011) Decomposition of variance for spatial Cox processes. Manuscript submitted for publication.
- 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

```
u <- vargamma.estK(redwood)
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,eta=1), nu.ker = -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.ker	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, 2011) 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 (2011). 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 (2011).

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:

<code>par</code>	Vector of fitted parameter values.
<code>fit</code>	Function value table (object of class "fv") containing the observed values of the summary statistic (<code>observed</code>) 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@csiro.au>
<http://www.maths.uwa.edu.au/~adrian/>

References

- Jalilian, A., Guan, Y. and Waagepetersen, R. (2011) Decomposition of variance for spatial Cox processes. Manuscript submitted for publication.
- 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")
```

Description

Returns the variance-covariance matrix of the estimates of the parameters of a fitted cluster point process model.

Usage

```
## S3 method for class 'kppm'
vcov(object, ...,
      what=c("vcov", "corr", "fisher", "internals"))
```

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.

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 $n \times n$ matrix where $n = \text{length}(\text{coef}(\text{model}))$.

Value

A square matrix.

Author(s)

Abdollah Jalilian and Rasmus Waagepetersen. Ported to **spatstat** by Adrian Baddeley <Adrian.Baddeley@csiro.au>

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

```
data(redwood)
fit <- kppm(redwood, ~ x + y)
vc <- vcov(fit)
sd <- sqrt(diag(vc))
t(coef(fit) + 1.96 * outer(sd, c(lower=-1, upper=1)))
vcov(fit, what="corr")
```

Description

Returns the variance-covariance matrix of the estimates of the parameters of a fitted point process model.

Usage

```
## S3 method for class 'ppm'
vcov(object, ..., what = "vcov", verbose = TRUE,
      gam.action=c("warn", "fatal", "silent"),
      matrix.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.
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 <code>gam</code> .
matrix.action	String indicating what to do if the matrix is ill-conditioned (so that its inverse 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 maximum pseudolikelihood (which is the default in `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.

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 `gamaction` determines what to do in this case. If `gamaction="fatal"`, an error is generated. If `gamaction="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 `gamaction="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 percent confidence intervals for the coefficients can also be obtained using `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. This 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@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>. New code for stationary Gibbs point processes was generously contributed by Ege Rubak and Jean-Francois Coeurjolly. New code for generic Gibbs process written by Adrian Baddeley.

References

Coeurjolly, J.-F. and Rubak, E. (2012) *Fast covariance estimation for innovations computed from a spatial Gibbs point process*. Research Report, Centre for Stochastic Geometry and Bioimaging, Denmark, 2012. www.csgb.dk

Kutoyants, Y.A. (1998) **Statistical Inference for Spatial Poisson Processes**, Lecture Notes in Statistics 134. New York: Springer 1998.

Examples

```
X <- rpoispp(42)
fit <- ppm(X, ~ x + y)
vcov(fit)
vcov(fit, what="Fish")

# example of singular system
data(demopat)
m <- ppm(demopat, ~polynom(x,y,2))
## Not run:
try(v <- vcov(m))

## End(Not run)
# rescale x, y coordinates to range [0,1] x [0,1] approximately
demopat <- rescale(demopat, 10000)
m <- ppm(demopat, ~polynom(x,y,2))
v <- vcov(m)

# Gibbs example
fitS <- ppm(swedishpines, ~1, Strauss(9))
coef(fitS)
sqrt(diag(vcov(fitS)))
```

vertices

Vertices of a Window

Description

Finds the vertices of a window

Usage

```
vertices(w)
```

Arguments

w A window.

Details

This function computes the vertices ('corners') of a spatial window.

The argument w should be a window (an object of class "owin", see [owin.object](#) for details).

If w is a rectangle, the coordinates of the four corner points are returned.

If w is a polygonal window (consisting of one or more polygons), the coordinates of the vertices of all polygons are returned.

If w is a binary mask, then a 'boundary pixel' is defined to be a pixel inside the window which has at least one neighbour outside the window. The coordinates of the centres of all boundary pixels are returned.

Value

A list with components x and y giving the coordinates of the vertices.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[owin.object](#).

Examples

```
data(letterR)
vert <- vertices(letterR)

plot(letterR, main="Polygonal vertices")
points(vert)
plot(letterR, main="Boundary pixels")
points(vertices(as.mask(letterR)))
```

volume

Volume of an Object

Description

Computes the volume of a spatial object such as a three-dimensional box.

Usage

```
volume(x)
```

Arguments

x An object whose volume will be computed.

Details

This function computes the volume of an object such as a three-dimensional box.

The function **volume** is generic, with methods for the classes "box3" (three-dimensional boxes) and "boxx" (multi-dimensional boxes). There is also a method for the class "owin" (two-dimensional windows), which is identical to [area.owin](#).

Value

The numerical value of the volume of the object.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[area.owin](#), [volume.box3](#), [volume.boxx](#)

which.max.im*Identify Pixelwise Maximum of Several Pixel Images*

Description

Given a list of pixel images, this function identifies the image that has the largest value at each pixel, and returns an image.

Usage

```
which.max.im(x)
```

Arguments

x A list of images (objects of class "im").

Details

x should be a list of pixel images. All images must have compatible dimensions.

For each pixel, the algorithm identifies which of the images in the list **x** has the largest value at that pixel. The index of this image becomes the pixel value in the output image. If **names(x)** is not null, then the indices are replaced by these names.

Value

An image (object of class "im") with factor values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[eval.im](#), [im.object](#)

Examples

```
# test images
X <- as.im(function(x,y) { x^2 - y^2 }, unit.square())
Y <- as.im(function(x,y) { x - y }, unit.square())
which.max.im(list(X=X, Y=Y))
```

<code>whist</code>	<i>Weighted Histogram</i>
--------------------	---------------------------

Description

Computes the weighted histogram of a set of observations with a given set of weights.

Usage

```
whist(x, breaks, weights = NULL)
```

Arguments

<code>x</code>	Numeric vector of observed values.
<code>breaks</code>	Vector of breakpoints for the histogram.
<code>weights</code>	Numeric vector of weights for the observed values.

Details

This low-level function computes (but does not plot) the weighted histogram of a vector of observations `x` using a given vector of `weights`.

The arguments `x` and `weights` should be numeric vectors of equal length. They may include NA or infinite values.

The argument `breaks` should be a numeric vector whose entries are strictly increasing. These values define the boundaries between the successive histogram cells. The `breaks` *do not* have to span the range of the observations.

There are $N-1$ histogram cells, where $N = \text{length}(\text{breaks})$. An observation $x[i]$ falls in the j th cell if $\text{breaks}[j] \leq x[i] < \text{breaks}[j+1]$ (for $j < N-1$) or $\text{breaks}[j] \leq x[i] \leq \text{breaks}[j+1]$ (for $j = N-1$). The weighted histogram value $h[j]$ for the j th cell is the sum of `weights[i]` for all observations `x[i]` that fall in the cell.

Note that, in contrast to the function `hist`, the function `whist` does not require the breakpoints to span the range of the observations `x`. Values of `x` that fall outside the range of `breaks` are handled separately; their total weight is returned as an attribute of the histogram.

Value

A numeric vector of length $N-1$ containing the histogram values, where $N = \text{length}(\text{breaks})$.

The return value also has attributes "low" and "high" giving the total weight of all observations that are less than the lowest breakpoint, or greater than the highest breakpoint, respectively.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz> with thanks to Peter Dalgaard.

Examples

```
x <- rnorm(100)
b <- seq(-1,1,length=21)
w <- runif(100)
whist(x,b,w)
```

will.expand	<i>Test Expansion Rule</i>
-------------	----------------------------

Description

Determines whether an expansion rule will actually expand the window or not.

Usage

```
will.expand(x)
```

Arguments

x Expansion rule. An object of class "rmhexpand".

Details

An object of class "rmhexpand" describes a rule for expanding a simulation window. See [rmhexpand](#) for details.

One possible expansion rule is to do nothing, i.e. not to expand the window.

This command inspects the expansion rule x and determines whether it will or will not actually expand the window. It returns TRUE if the window will be expanded.

Value

Logical value.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and Rolf Turner <r.turner@auckland.ac.nz>

See Also

[rmhexpand](#), [expand.owin](#)

Examples

```
x <- rmhexpand(distance=0.2)
y <- rmhexpand(area=1)
will.expand(x)
will.expand(y)
```

with.fv*Evaluate an Expression in a Function Table***Description**

Evaluate an R expression in a function value table (object of class "fv").

Usage

```
## S3 method for class 'fv'
with(data, expr, ..., drop = TRUE)
```

Arguments

<code>data</code>	A function value table (object of class "fv") in which the expression will be evaluated.
<code>expr</code>	The expression to be evaluated. An R language expression, which may involve the names of columns in <code>data</code> , the special abbreviations <code>. , .x</code> and <code>.y</code> , and global constants or functions.
<code>...</code>	Ignored.
<code>drop</code>	Logical value. If the result of evaluating the expression <code>expr</code> is a vector (rather than a matrix or data frame) then the result will be returned as a vector if <code>drop=TRUE</code> . Otherwise it will be returned as another function value table (object of class "fv").

Details

This is a method for the generic command [with](#) for an object of class "fv" (function value table). An object of class "fv" is a convenient way of storing and plotting several different estimates of the same function. It is effectively a data frame with extra attributes. See [fv.object](#) for further explanation.

This command makes it possible to perform computations that involve different estimates of the same function. For example we use it to compute the arithmetic difference between two different edge-corrected estimates of the K function of a point pattern.

The argument `expr` should be an R language expression. The expression may involve

- the name of any column in `data`, referring to one of the estimates of the function;
- the symbol `.` which stands for all the available estimates of the function;
- the symbol `.y` which stands for the recommended estimate of the function (in an "fv" object, one of the estimates is always identified as the recommended estimate);
- the symbol `.x` which stands for the argument of the function;
- global constants or functions.

See the Examples.

The expression should be capable of handling vectors and matrices. If the result of evaluating the expression is a matrix or data frame, then it is returned as a new function value table (object of class "fv"). If the result of evaluation is a vector and `drop=TRUE` then the result is returned as a vector.

To perform calculations involving *several* objects of class "fv", use [eval.fv](#).

Value

Either a function value table (object of class "fv") or a vector.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[with](#), [fv.object](#), [eval.fv](#), [Kest](#)

Examples

```
# compute 4 estimates of the K function
X <- rpoispp(42)
K <- Kest(X)
plot(K)

# derive 4 estimates of the L function L(r) = sqrt(K(r)/pi)
L <- with(K, sqrt(./pi))
plot(L)

# compute 4 estimates of V(r) = L(r)/r
V <- with(L, ./x)
plot(V)

# compute the maximum absolute difference between
# the isotropic and translation correction estimates of K(r)
D <- with(K, max(abs(iso - trans)))
```

[with.hyperframe](#)

Evaluate an Expression in Each Row of a Hyperframe

Description

An expression, involving the names of columns in a hyperframe, is evaluated separately for each row of the hyperframe.

Usage

```
## S3 method for class 'hyperframe'
with(data, expr, ...,
      simplify = TRUE,
      ee = NULL, enclos=NULL)
```

Arguments

- | | |
|------|--|
| data | A hyperframe (object of class "hyperframe") containing data. |
| expr | An R language expression to be evaluated. |
| ... | Ignored. |

simplify	Logical. If TRUE, the return value will be simplified to a vector whenever possible.
ee	Alternative form of expr, as an object of class "expression".
enclos	An environment in which to search for objects that are not found in the hyperframe. Defaults to <code>parent.frame()</code> .

Details

This function evaluates the expression `expr` in each row of the hyperframe data. It is a method for the generic function `with`.

The argument `expr` should be an R language expression in which each variable name is either the name of a column in the hyperframe data, or the name of an object in the parent frame (the environment in which `with` was called.) The argument `ee` can be used as an alternative to `expr` and should be an expression object (of class "expression").

For each row of data, the expression will be evaluated so that variables which are column names of data are interpreted as the entries for those columns in the current row.

For example, if a hyperframe `h` has columns called `A` and `B`, then `with(h, A != B)` inspects each row of data in turn, tests whether the entries in columns `A` and `B` are equal, and returns the n logical values.

Value

Normally a list of length n (where n is the number of rows) containing the results of evaluating the expression for each row. If `simplify=TRUE` and each result is a single atomic value, then the result is a vector or factor containing the same values.

Author(s)

Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/> and
Rolf Turner <r.turner@auckland.ac.nz>

See Also

[hyperframe](#), [plot.hyperframe](#)

Examples

```
# generate Poisson point patterns with intensities 10 to 100
H <- hyperframe(L=seq(10,100, by=10))
X <- with(H, rpoispp(L))
```

Description

Modifies a pixel image, identifying those pixels that have values very close to zero, and replacing the value by zero.

Usage

```
zapsmall.im(x, digits)
```

Arguments

- | | |
|--------|---|
| x | Pixel image (object of class "im"). |
| digits | Argument passed to <code>zapsmall</code> indicating the precision to be used. |

Details

The function `zapsmall` is applied to each pixel value of the image `x`.

Value

Another pixel image.

Author(s)

Ege Rubak and Adrian Baddeley <Adrian.Baddeley@csiro.au> <http://www.maths.uwa.edu.au/~adrian/>

See Also

`zapsmall`

Examples

```
data(cells)
D <- density(cells)
zapsmall.im(D)
```

Index

- *Topic **IO**
 - scanpp, 840
- *Topic **arith**
 - whist, 932
- *Topic **array**
 - sumouter, 891
- *Topic **attribute**
 - bind.fv, 99
 - fasp.object, 262
 - fv.object, 281
 - im.object, 327
 - owin.object, 557
 - ppm.object, 652
 - ppp.object, 657
 - pppmatching.object, 663
 - psp.object, 685
 - quad.object, 697
- *Topic **classes**
 - fv, 279
- *Topic **classif**
 - nnclean, 528
- *Topic **color**
 - colourmap, 143
 - colourtools, 145
 - plot.colourmap, 600
- *Topic **datagen**
 - box3, 105
 - boxx, 106
 - clickjoin, 134
 - default.dummy, 175
 - default.expand, 176
 - default.rmhcontrol, 177
 - disc, 203
 - gridcentres, 306
 - gridweights, 307
 - im, 325
 - inflne, 330
 - owin, 555
 - pixelquad, 597
 - pp3, 644
 - ppp, 655
 - pppmatching, 662
 - ppx, 664
- psp, 684
- quadratresample, 705
- quadrats, 706
- quadscheme, 708
- rCauchy, 713
- rcell, 714
- rDGS, 715
- rDiggleGratton, 717
- rGaussPoisson, 737
- rgbim, 738
- rHardcore, 739
- rjitter, 746
- rlabel, 748
- rLGCP, 749
- rlinegrid, 751
- rMatClust, 752
- rMaternI, 753
- rMaternII, 754
- rmh, 755
- rmh.default, 757
- rmh.ppm, 765
- rmhcontrol, 769
- rmhexpand, 772
- rmhmodel, 774
- rmhmodel.default, 775
- rmhmodel.list, 781
- rmhmodel.ppm, 783
- rmhstart, 785
- rMosaicField, 787
- rMosaicSet, 788
- rmpoint, 789
- rmpoispp, 792
- rNeymanScott, 795
- rpoint, 801
- rpoisline, 802
- rpoislinetess, 803
- rpoislpp, 804
- rpoispp, 805
- rpoispp3, 806
- rpoisppOnLines, 807
- rpoisppx, 809
- rPoissonCluster, 810
- rshift, 811

rshift.hpp, 812
rshift.hpp, 815
rshift.splitppp, 816
RSSI, 817
rstrat, 818
rStrauss, 819
rStraussHard, 821
rsyst, 823
rthin, 824
rThomas, 825
runifdisc, 826
runiflpp, 827
runifpoint, 828
runifpoint3, 829
runifpointOnLines, 830
runifpointx, 831
rVarGamma, 832
spokes, 873
square, 876
tess, 895

*Topic **datasets**

- amacrine, 44
- anemones, 45
- ants, 51
- bei, 95
- betacells, 98
- bramblecanes, 107
- bronzefilter, 108
- cells, 123
- chicago, 125
- chorley, 127
- copper, 161
- demopat, 180
- finpines, 268
- flu, 274
- ganglia, 283
- gorillas, 302
- hamster, 308
- heather, 313
- humberside, 317
- japanesepines, 361
- lansing, 419
- letterR, 432
- longleaf, 461
- murchison, 521
- nbfires, 523
- nztrees, 548
- osteo, 553
- ponderosa, 639
- redwood, 721
- redwoodfull, 722
- residualspaper, 736

shapley, 843
simdat, 850
simplesnet, 851
spruces, 875
swedishpines, 894
urkiola, 918

*Topic **distribution**

- rknn, 747

*Topic **documentation**

- latest.news, 420

*Topic **hplot**

- contour.im, 155
- contour.listof, 156
- diagnose.ppm, 188
- envelope, 222
- envelope.envelope, 229
- envelope.pp3, 233
- iplot, 343
- istat, 360
- layered, 421
- lurking, 464
- pairs.im, 567
- persp.im, 591
- plot.bermantest, 599
- plot.colourmap, 600
- plot.envelope, 601
- plot.fasp, 602
- plot.fv, 604
- plot.hyperframe, 607
- plot.im, 608
- plot.kstest, 613
- plot.layered, 614
- plot.listof, 618
- plot.msr, 620
- plot.owin, 621
- plot.plotppm, 623
- plot.pp3, 625
- plot.ppm, 626
- plot.ppp, 628
- plot.psp, 631
- plot.quad, 632
- plot.slrm, 634
- plot.splitppp, 635
- plot.tess, 636
- pool.envelope, 641
- pool.fasp, 642
- qqplot.ppm, 693

*Topic **htest**

- bermantest, 96
- clf.test, 132
- envelope, 222
- envelope.envelope, 229

envelope.ppp, 233
 kstest.ppm, 415
 pool.envelope, 641
 pool.fasp, 642
 quadrat.test, 700
 quadrat.test.splitppp, 702
 scan.test, 838
***Topic iplot**
 clickpoly, 135
 clickppp, 136
 identify.hppp, 320
 identify.psp, 321
 transect.im, 903
***Topic iteration**
 applynbd, 54
 envelope, 222
 envelope.envelope, 229
 envelope.ppp, 233
 pool.envelope, 641
 pool.fasp, 642
***Topic manip**
 append.psp, 53
 as.box3, 63
 as.data.frame.hyperframe, 64
 as.data.frame.hppp, 66
 as.data.frame.psp, 67
 as.hyperframe, 69
 as.hyperframe.ppx, 70
 as.im, 71
 as.mask, 75
 as.mask.psp, 76
 as.owin, 79
 as.polygonal, 82
 as.hppp, 83
 as.psp, 85
 as.rectangle, 88
 as.tess, 89
 by.im, 115
 by.hppp, 116
 cbind.hyperframe, 122
 collapse.fv, 142
 commonGrid, 146
 compatible, 148
 compatible.fasp, 149
 compatible.fv, 150
 compatible.im, 151
 concatxy, 153
 coords, 160
 crossing.psp, 170
 data.hppp, 174
 delaunay, 178
 dirichlet, 201
 discretise, 205
 edges2triangles, 215
 endpoints.psp, 221
 eval.fasp, 237
 eval.fv, 239
 eval.im, 240
 expand.owin, 244
 Extract.fasp, 245
 Extract.fv, 246
 Extract.im, 247
 Extract.listof, 249
 Extract.lpp, 250
 Extract.msr, 251
 Extract.hppp, 252
 Extract.hppx, 254
 Extract.psp, 255
 Extract.quad, 257
 Extract.splitppp, 258
 Extract.tess, 259
 harmonise.im, 312
 hyperframe, 318
 im, 325
 interp.im, 339
 is.convex, 346
 is.empty, 347
 is.im, 348
 is.marked, 348
 is.marked.hppp, 349
 is.marked.ppp, 350
 is.multitype, 351
 is.multitype.hppp, 352
 is.multitype.ppp, 354
 is.owin, 355
 is.hppp, 356
 is.ppp, 356
 is.rectangle, 357
 levelset, 432
 lut, 467
 marks, 476
 marks.psp, 478
 nearest.raster.point, 526
 npoints, 546
 nsegments, 547
 periodify, 590
 pixellate, 593
 pixellate.owin, 594
 pixellate.hppp, 595
 pixellate.psp, 596
 pointsOnLines, 637
 quad.hppp, 698
 raster.x, 711
 rat, 712

Replace.im, 726
rescue.rectangle, 732
rgbim, 738
selfcrossing.psp, 841
shift, 845
shift.im, 846
shift.owin, 847
shift.ppp, 848
shift.psp, 849
solutionset, 864
split.im, 868
split.ppp, 869
split.ppx, 871
superimpose, 892
tile.areas, 901
tiles, 902
transect.im, 903
trim.rectangle, 904
union.quad, 908
unitname, 910
unmark, 912
which.max.im, 931
will.expand, 933
with.fv, 934
with.hyperframe, 935

*Topic **math**

affine, 35
affine.im, 36
affine.owin, 37
affine.ppp, 38
affine.psp, 39
angles.psp, 46
area.owin, 57
areaGain, 58
areaLoss, 62
bdist.pixels, 92
bdist.points, 93
bdist.tiles, 94
border, 102
centroid.owin, 123
chop.tess, 126
circumradius, 128
clip.inflne, 137
closing, 138
complement.owin, 151
connected, 154
convolve.im, 159
crossdist, 164
crossdist.default, 165
crossdist.pp3, 166
crossdist.ppp, 167
crossdist.ppx, 168
crossdist.psp, 169
deltametric, 179
diameter, 192
diameter.box3, 193
diameter.boxx, 194
diameter.owin, 195
dilated.areas, 199
dilation, 200
discpartarea, 204
distfun, 206
distmap, 207
distmap.owin, 208
distmap.ppp, 210
distmap.psp, 211
dummify, 212
eroded.areas, 235
erosion, 236
flipxy, 273
imcov, 328
incircle, 329
inside.owin, 334
integral.im, 335
intersect.owin, 340
intersect.tess, 342
issubset.owin, 359
LambertW, 418
lengths.psp, 428
matchingdist, 483
midpoints.psp, 505
nearestsegment, 527
nncross, 532
nndist, 534
nndist.pp3, 536
nndist.ppx, 537
nndist.psp, 539
nnwhich, 540
nnwhich.pp3, 542
nnwhich.ppx, 543
opening, 548
pairdist, 558
pairdist.default, 559
pairdist.pp3, 561
pairdist.ppp, 562
pairdist.ppx, 563
pairdist.psp, 564
perimeter, 589
pppdist, 659
project2segment, 682
quadratcount, 703
reflect, 723
rescale, 727
rescale.im, 728

rescale.owin, 729
 rescale.hpp, 730
 rescale.psp, 731
 rotate, 797
 rotate.owin, 798
 rotate.hpp, 799
 rotate.psp, 800
 scalardilate, 836
 setcov, 842
 simplify.owin, 851
 stieljes, 877
 vertices, 929
 volume, 930
***Topic methods**
 adaptive.density, 34
 anova.lppm, 47
 anova.ppm, 48
 anova.slrm, 50
 as.data.frame.im, 65
 as.function.fv, 68
 as.matrix.im, 77
 as.matrix.owin, 78
 bw.diggle, 109
 bw.relrisk, 110
 bw.scott, 112
 bw.smoothppp, 113
 bw.stoyan, 114
 by.im, 115
 by.hpp, 116
 coef.ppm, 139
 coef.slrm, 141
 cut.im, 171
 cut.hpp, 172
 density.hpp, 181
 density.psp, 184
 density.splitppp, 185
 duplicated.hpp, 214
 fitted.ppm, 271
 fitted.slrm, 272
 formula.ppm, 276
 hist.im, 316
 idw, 322
 logLik.slrm, 458
 mean.im, 489
 methods.box3, 490
 methods.boxx, 491
 methods.distfun, 492
 methods.kppm, 494
 methods.linnet, 495
 methods.lpp, 496
 methods.pp3, 499
 methods.rho2hat, 501
 methods.rhohat, 502
 methods.slrm, 503
 methods.units, 504
 predict.slrm, 672
 quantile.im, 710
 relrisk, 724
 residuals.hpp, 733
 scaletointerval, 837
 smooth.hpp, 860
 split.im, 868
 split.hpp, 869
 split.ppx, 871
 summary.im, 883
 summary.listof, 884
 summary.owin, 885
 summary.ppm, 886
 summary.hpp, 887
 summary.psp, 888
 summary.quad, 889
 summary.splitppp, 890
 unique.hpp, 909
 update.ppm, 915
 update.rmhcontrol, 917
 vcov.kppm, 925
 vcov.ppm, 926
 zapsmall.im, 936
***Topic models**
 anova.lppm, 47
 anova.ppm, 48
 anova.slrm, 50
 AreaInter, 59
 as.interact, 74
 BadGey, 90
 cauchy.estK, 118
 cauchy.estpcf, 120
 coef.ppm, 139
 coef.slrm, 141
 compareFit, 147
 data.ppm, 174
 dfbetas.ppm, 186
 diagnose.ppm, 188
 DiggleGatesStibbard, 196
 DiggleGratton, 197
 dummy.ppm, 213
 eem, 216
 effectfun, 217
 exactMLEstrauss, 243
 Fiksel, 267
 fitin.ppm, 269
 fitted.ppm, 271
 fitted.slrm, 272
 Gcom, 284

Geyer, 296
Gres, 304
Hardcore, 309
harmonic, 310
influence.ppm, 332
inforder.family, 333
intensity, 336
intensity.ppm, 337
ippm, 344
is.marked.ppm, 349
is.multitype.ppm, 352
is.ppm, 356
is.stationary, 358
Kcom, 374
Kmodel, 402
kppm, 409
Kres, 411
LennardJones, 429
leverage.ppm, 434
lgcp.estK, 435
lgcp.estpcf, 438
logLik.ppm, 457
logLik.slrm, 458
lppm, 463
lurking, 464
matclust.estK, 485
matclust.estpcf, 487
methods.lppm, 498
mincontrast, 506
model.depends, 509
model.frame.ppm, 511
model.images, 512
model.matrix.ppm, 513
msr, 515
MultiHard, 517
MultiStrauss, 519
MultiStraussHard, 520
Ord, 550
ord.family, 551
OrdThresh, 552
PairPiece, 565
pairsat.family, 568
Pairwise, 569
pairwise.family, 571
plot.influence.ppm, 611
plot.kppm, 612
plot.leverage.ppm, 615
plot.plotppm, 623
plot.ppm, 626
plot.slrm, 634
Poisson, 638
ppm, 645
predict.kppm, 666
predict.lppm, 667
predict.ppm, 668
predict.slrm, 672
print.ppm, 675
profilepl, 678
project.ppm, 681
psst, 686
psstA, 688
psstG, 691
qqplot.ppm, 693
quad.ppm, 698
reach, 718
residuals.ppm, 733
residualspaper, 736
rho2hat, 740
rhohat, 742
rmh.ppm, 765
SatPiece, 833
Saturated, 835
simulate.kppm, 852
simulate.ppm, 853
simulate.slrm, 855
slrm, 856
smooth.msr, 859
Softcore, 862
Strauss, 878
StraussHard, 879
suffstat, 881
summary.ppm, 886
thomas.estK, 897
thomas.estpcf, 899
triplet.family, 905
Triplets, 905
update.kppm, 914
update.ppm, 915
update.rmhcontrol, 917
valid.ppm, 919
vargamma.estK, 921
vargamma.estpcf, 923
vcov.kppm, 925
vcov.ppm, 926
***Topic nonparametric**
allstats, 40
alltypes, 41
blur, 101
clarkevans, 129
clarkevans.test, 131
Emark, 219
ewcdf, 242
F3est, 260
Fest, 263

fryplot, 277
 G3est, 282
 Gcross, 287
 Gdot, 291
 Gest, 293
 Gfox, 298
 Gmulti, 299
 Hest, 314
 Iest, 323
 intensity.ppp, 338
 Jcross, 362
 Jdot, 364
 Jest, 367
 Jmulti, 370
 K3est, 372
 kaplan.meier, 373
 Kcross, 378
 Kcross.inhom, 380
 Kdot, 383
 Kdot.inhom, 386
 Kest, 389
 Kest.fft, 393
 Kinhom, 394
 km.rs, 398
 Kmeasure, 400
 Kmuli, 403
 Kmuli.inhom, 406
 Kscaled, 412
 Lcross, 422
 Lcross.inhom, 423
 Ldot, 425
 Ldot.inhom, 426
 Lest, 430
 linearK, 442
 linearKinhom, 443
 linearpcf, 444
 linearpcf.inhom, 446
 Linhom, 447
 localK, 451
 localKinhom, 453
 localpcf, 455
 lohboot, 459
 markconnect, 468
 markcorr, 470
 markcorrint, 474
 markvario, 481
 miplot, 508
 nncorr, 529
 npfun, 545
 pcf, 571
 pcf.fasp, 573
 pcf.fv, 575
 pcf.ppp, 576
 pcf3est, 578
 pcfcross, 580
 pcfcross.inhom, 582
 pcfdot, 584
 pcfdot.inhom, 585
 pcfinhom, 587
 pool.rat, 643
 reduced.sample, 720
 sharpen, 844
 smooth.fv, 858
 Tstat, 907
 varblock, 920
***Topic package**
 spatstat-package, 14
***Topic print**
 print.im, 673
 print.owin, 674
 print.ppm, 675
 print.ppp, 676
 print.psp, 677
 print.quad, 677
 progressreport, 680
***Topic programming**
 applynbd, 54
 eval.fasp, 237
 eval.fv, 239
 eval.im, 240
 levelset, 432
 markstat, 479
 marktable, 480
 solutionset, 864
 which.max.im, 931
 with.fv, 934
 with.hyperframe, 935
***Topic smooth**
 adaptive.density, 34
 bw.diggle, 109
 bw.relrisk, 110
 bw.scott, 112
 bw.smoothppp, 113
 bw.stoyan, 114
 density.ppp, 181
 density.psp, 184
 density.splitppp, 185
 idw, 322
 relrisk, 724
 smooth.ppp, 860
 unnormdensity, 913
***Topic spatial**
 adaptive.density, 34
 affine, 35

affine.im, 36
affine.owin, 37
affine.ppp, 38
affine.psp, 39
allstats, 40
alltypes, 41
amacrine, 44
anemones, 45
angles.psp, 46
anova.lppm, 47
anova.ppm, 48
anova.slrm, 50
ants, 51
append.psp, 53
applynbd, 54
area.owin, 57
areaGain, 58
AreaInter, 59
areaLoss, 62
as.box3, 63
as.data.frame.hyperframe, 64
as.data.frame.im, 65
as.data.frame.ppp, 66
as.data.frame.psp, 67
as.function.fv, 68
as.hyperframe, 69
as.hyperframe.ppx, 70
as.im, 71
as.interact, 74
as.mask, 75
as.mask.psp, 76
as.matrix.im, 77
as.matrix.owin, 78
as.owin, 79
as.polygonal, 82
as.ppp, 83
as.psp, 85
as.rectangle, 88
as.tess, 89
BadGey, 90
bdist.pixels, 92
bdist.points, 93
bdist.tiles, 94
bei, 95
bermantest, 96
betacells, 98
bind.fv, 99
blur, 101
border, 102
bounding.box, 103
bounding.box.xy, 104
box3, 105
boxx, 106
bramblecanes, 107
bronzefilter, 108
bw.diggle, 109
bw.rerisk, 110
bw.scott, 112
bw.smoothppp, 113
bw.stoyan, 114
by.im, 115
by.ppp, 116
cauchy.estK, 118
cauchy.estpcf, 120
cbind.hyperframe, 122
cells, 123
centroid.owin, 123
chicago, 125
chop.tess, 126
chorley, 127
circumradius, 128
clarkevans, 129
clarkevans.test, 131
clf.test, 132
clickjoin, 134
clickpoly, 135
clickppp, 136
clip.inflne, 137
closing, 138
coef.ppm, 139
coef.slrm, 141
collapse.fv, 142
colourmap, 143
commonGrid, 146
compareFit, 147
compatible, 148
compatible.fasp, 149
compatible.fv, 150
compatible.im, 151
complement.owin, 151
concatxy, 153
connected, 154
contour.im, 155
contour.listof, 156
convexhull, 157
convexhull.xy, 158
convolve.im, 159
coords, 160
copper, 161
corners, 163
crossdist, 164
crossdist.default, 165
crossdist.pp3, 166
crossdist.ppp, 167

crossdist.ppx, 168
 crossdist.psp, 169
 crossing.psp, 170
 cut.im, 171
 cut.ppp, 172
 data.ppm, 174
 default.dummy, 175
 default.expand, 176
 default.rmhcontrol, 177
 delaunay, 178
 deltametric, 179
 demopat, 180
 density.ppp, 181
 density.psp, 184
 density.splitppp, 185
 dfbetas.ppm, 186
 diagnose.ppm, 188
 diameter, 192
 diameter.box3, 193
 diameter.boxx, 194
 diameter.owin, 195
 DiggleGatesStibbard, 196
 DiggleGratton, 197
 dilated.areas, 199
 dilation, 200
 dirichlet, 201
 dirichlet.weights, 202
 disc, 203
 discpartarea, 204
 discretise, 205
 distfun, 206
 distmap, 207
 distmap.owin, 208
 distmap.ppp, 210
 distmap.psp, 211
 dummy.ppm, 213
 duplicated.ppp, 214
 edges2triangles, 215
 eem, 216
 effectfun, 217
 Emark, 219
 endpoints.psp, 221
 envelope, 222
 envelope.envelope, 229
 envelope.lpp, 231
 envelope.pp3, 233
 eroded.areas, 235
 erosion, 236
 eval.fasp, 237
 eval.fv, 239
 eval.im, 240
 exactMPLEstrauss, 243
 expand.owin, 244
 Extract.fasp, 245
 Extract.fv, 246
 Extract.im, 247
 Extract.listof, 249
 Extract.lpp, 250
 Extract.msr, 251
 Extract.ppp, 252
 Extract.ppx, 254
 Extract.psp, 255
 Extract.quad, 257
 Extract.splitppp, 258
 Extract.tess, 259
 F3est, 260
 fasp.object, 262
 Fest, 263
 Fiksel, 267
 finpines, 268
 fitin.ppm, 269
 fitted.ppm, 271
 fitted.slrm, 272
 flipxy, 273
 formula.ppm, 276
 fryplot, 277
 fv, 279
 fv.object, 281
 G3est, 282
 ganglia, 283
 Gcom, 284
 Gcross, 287
 Gdot, 291
 Gest, 293
 Geyer, 296
 Gfox, 298
 Gmulti, 299
 gpc2owin, 303
 Gres, 304
 gridcentres, 306
 gridweights, 307
 hamster, 308
 Hardcore, 309
 harmonic, 310
 harmonise.im, 312
 heather, 313
 Hest, 314
 hist.im, 316
 humberside, 317
 hyperframe, 318
 identify.ppp, 320
 identify.psp, 321
 idw, 322
 Iest, 323

im, 325
im.object, 327
imcov, 328
incircle, 329
inflne, 330
influence.ppm, 332
inforder.family, 333
inside.owin, 334
integral.im, 335
intensity, 336
intensity.ppm, 337
intensity.ppp, 338
interp.im, 339
intersect.owin, 340
intersect.tess, 342
iplot, 343
ippm, 344
is.convex, 346
is.empty, 347
is.im, 348
is.marked, 348
is.marked.ppm, 349
is.marked.ppp, 350
is.multitype, 351
is.multitype.ppm, 352
is.multitype.ppp, 354
is.owin, 355
is.ppm, 356
is.ppp, 356
is.rectangle, 357
is.stationary, 358
is.subset.owin, 359
istat, 360
japanesepines, 361
Jcross, 362
Jdot, 364
Jest, 367
Jmulti, 370
K3est, 372
kaplan.meier, 373
Kcom, 374
Kcross, 378
Kcross.inhom, 380
Kdot, 383
Kdot.inhom, 386
Kest, 389
Kest.fft, 393
Kinhom, 394
km.rs, 398
Kmeasure, 400
Kmodel, 402
Kmulti, 403
Kmulti.inhom, 406
kppm, 409
Kres, 411
Kscaled, 412
kstest.ppm, 415
lansing, 419
layered, 421
Lcross, 422
Lcross.inhom, 423
Ldot, 425
Ldot.inhom, 426
lengths.psp, 428
LennardJones, 429
Lest, 430
letterR, 432
levelset, 432
leverage.ppm, 434
lgcp.estK, 435
lgcp.estpcf, 438
lineardisc, 441
linearK, 442
linearKinhom, 443
linearppcf, 444
linearpcfinhom, 446
Linhom, 447
linim, 448
linnet, 450
localK, 451
localKinhom, 453
localpcf, 455
logLik.ppm, 457
logLik.slrm, 458
lohboot, 459
longleaf, 461
lpp, 462
lppm, 463
lurking, 464
lut, 467
markconnect, 468
markcorr, 470
markcorrint, 474
marks, 476
marks.psp, 478
markstat, 479
marktable, 480
markvario, 481
matchingdist, 483
matclust.estK, 485
matclust.estpcf, 487
mean.im, 489
methods.box3, 490
methods.boxx, 491

methods.distfun, 492
 methods.kppm, 494
 methods.linnet, 495
 methods.lpp, 496
 methods.lppm, 498
 methods.pp3, 499
 methods.ppx, 500
 methods.rho2hat, 501
 methods.rhohat, 502
 methods.slrm, 503
 methods.units, 504
 midpoints.psp, 505
 mincontrast, 506
 miplot, 508
 model.depends, 509
 model.frame.ppm, 511
 model.images, 512
 model.matrix.ppm, 513
 msr, 515
 MultiHard, 517
 multiplicity.ppp, 518
 MultiStrauss, 519
 MultiStraussHard, 520
 murchison, 521
 nbfires, 523
 nearest.raster.point, 526
 nearestsegment, 527
 nncclean, 528
 nncorr, 529
 nncross, 532
 nndist, 534
 nndist.pp3, 536
 nndist.ppx, 537
 nndist.psp, 539
 nnwhich, 540
 nnwhich.pp3, 542
 nnwhich.ppx, 543
 npfun, 545
 npoints, 546
 nsegments, 547
 nztrees, 548
 opening, 548
 Ord, 550
 ord.family, 551
 OrdThresh, 552
 owin, 555
 owin.object, 557
 pairdist, 558
 pairdist.default, 559
 pairdist.lpp, 560
 pairdist.pp3, 561
 pairdist.ppp, 562
 pairdist.ppx, 563
 pairdist.psp, 564
 PairPiece, 565
 pairs.im, 567
 pairsat.family, 568
 Pairwise, 569
 pairwise.family, 571
 pcf, 571
 pcf.fasp, 573
 pcf.fv, 575
 pcf.ppp, 576
 pcf3est, 578
 pcfcross, 580
 pcfcross.inhom, 582
 pcfdot, 584
 pcfdot.inhom, 585
 pcfinhom, 587
 perimeter, 589
 periodify, 590
 persp.im, 591
 pixellate, 593
 pixellate.owin, 594
 pixellate.ppp, 595
 pixellate.psp, 596
 pixelquad, 597
 plot.bermantest, 599
 plot.colourmap, 600
 plot.envelope, 601
 plot.fasp, 602
 plot.fv, 604
 plot.hyperframe, 607
 plot.im, 608
 plot.influence.ppm, 611
 plot.kppm, 612
 plot.kstest, 613
 plot.layered, 614
 plot.leverage.ppm, 615
 plot.linim, 616
 plot.linnet, 617
 plot.listof, 618
 plot.msr, 620
 plot.owin, 621
 plot.plotppm, 623
 plot.pp3, 625
 plot.ppm, 626
 plot.ppp, 628
 plot.psp, 631
 plot.quad, 632
 plot.slrm, 634
 plot.splitppp, 635
 plot.tess, 636
 pointsOnLines, 637

Poisson, 638
ponderosa, 639
pool, 640
pool.envelope, 641
pool.fasp, 642
pool.rat, 643
pp3, 644
ppm, 645
ppm.object, 652
ppp, 655
ppp.object, 657
pppdist, 659
pppmatching, 662
pppmatching.object, 663
ppx, 664
predict.kppm, 666
predict.lppm, 667
predict.ppm, 668
predict.sirm, 672
print.im, 673
print.owin, 674
print.ppm, 675
print.ppp, 676
print.psp, 677
print.quad, 677
profilepl, 678
project.ppm, 681
project2segment, 682
psp, 684
psp.object, 685
psst, 686
pssta, 688
psstG, 691
qqplot.ppm, 693
quad.object, 697
quad.ppm, 698
quadrat.test, 700
quadrat.test.splitppp, 702
quadratcount, 703
quadratresample, 705
quadscheme, 708
quantile.im, 710
raster.x, 711
rat, 712
rCauchy, 713
rcell, 714
rDGS, 715
rDiggleGratton, 717
reach, 718
reduced.sample, 720
redwood, 721
redwoodfull, 722
reflect, 723
relrisk, 724
Replace.im, 726
rescale, 727
rescale.im, 728
rescale.owin, 729
rescale.ppp, 730
rescale.psp, 731
rescue.rectangle, 732
residuals.ppm, 733
residualspaper, 736
rGaussPoisson, 737
rgbim, 738
rHardcore, 739
rho2hat, 740
rhohat, 742
ripras, 744
rjitter, 746
rknn, 747
rlabel, 748
rLGCP, 749
rlinegrid, 751
rMatClust, 752
rMaternI, 753
rMaternII, 754
rmh, 755
rmh.default, 757
rmh.ppm, 765
rmhcontrol, 769
rmhexpand, 772
rmhmodel, 774
rmhmodel.default, 775
rmhmodel.list, 781
rmhmodel.ppm, 783
rmhstart, 785
rMosaicField, 787
rMosaicSet, 788
rmpoint, 789
rmpoispp, 792
rNeymanScott, 795
rotate, 797
rotate.owin, 798
rotate.ppp, 799
rotate.psp, 800
rpoint, 801
rpoisline, 802
rpoislinetess, 803
rpoislpp, 804
rpoispp, 805
rpoispp3, 806
rpoisppOnLines, 807
rpoisppx, 809

rPoissonCluster, 810
 rshift, 811
 rshift.ppp, 812
 rshift.psp, 815
 rshift.splitppp, 816
 RSSI, 817
 rstrat, 818
 rStrauss, 819
 rStraussHard, 821
 rsyst, 823
 rthin, 824
 rThomas, 825
 runifdisc, 826
 runiflpp, 827
 runifpoint, 828
 runifpoint3, 829
 runifpointOnLines, 830
 runifpointx, 831
 rVarGamma, 832
 SatPiece, 833
 Saturated, 835
 scalardilate, 836
 scaletointerval, 837
 scan.test, 838
 scanpp, 840
 selfcrossing.psp, 841
 setcov, 842
 shapley, 843
 sharpen, 844
 shift, 845
 shift.im, 846
 shift.owin, 847
 shift.ppp, 848
 shift.psp, 849
 simdat, 850
 simplenet, 851
 simplify.owin, 851
 simulate.kppm, 852
 simulate.ppm, 853
 simulate.slrm, 855
 slrm, 856
 smooth.fv, 858
 smooth.msr, 859
 smooth.ppp, 860
 Softcore, 862
 solutionset, 864
 spatstat-package, 14
 spatstat.options, 865
 split.im, 868
 split.ppp, 869
 split.ppx, 871
 spokes, 873
 spruces, 875
 square, 876
 stieljes, 877
 Strauss, 878
 StraussHard, 879
 suffstat, 881
 summary.im, 883
 summary.listof, 884
 summary.owin, 885
 summary.ppm, 886
 summary.ppp, 887
 summary.psp, 888
 summary.quad, 889
 summary.splitppp, 890
 superimpose, 892
 swedishpines, 894
 tess, 895
 thomas.estK, 897
 thomas.estpcf, 899
 tile.areas, 901
 tiles, 902
 transect.im, 903
 trim.rectangle, 904
 triplet.family, 905
 Triplets, 905
 Tstat, 907
 union.quad, 908
 unique.ppp, 909
 unitname, 910
 unmark, 912
 update.kppm, 914
 update.ppm, 915
 update.rmhcontrol, 917
 valid.ppm, 919
 varblock, 920
 vargamma.estK, 921
 vargamma.estpcf, 923
 vcov.kppm, 925
 vcov.ppm, 926
 vertices, 929
 volume, 930
 which.max.im, 931
 will.expand, 933
 with.fv, 934
 with.hyperframe, 935
 zapsmall.im, 936
 *Topic **univar**
 ewcdf, 242
 mean.im, 489
 quantile.im, 710
 scaletointerval, 837
 zapsmall.im, 936

***Topic utilities**

bounding.box, 103
 bounding.box.xy, 104
 convexhull, 157
 convexhull.xy, 158
 corners, 163
 dirichlet.weights, 202
 dummy.ppm, 213
 gpc2owin, 303
 multiplicity.ppp, 518
 quadrats, 706
 ripras, 744
 .Random.seed, 758, 760
 [, 250, 252, 255–257
 [.data.frame, 246
 [.fasp, 262, 263
 [.fasp (Extract.fasp), 245
 [.fv (Extract.fv), 246
 [.im, 20, 171, 326, 328, 335, 336, 727
 [.im (Extract.im), 247
 [.lpp (Extract.lpp), 250
 [.msr, 516
 [.msr (Extract.msr), 251
 [.ppp, 18, 173, 257, 658
 [.ppp (Extract.ppp), 252
 [.ppx (Extract.ppx), 254
 [.psp, 21, 685, 686
 [.psp (Extract.psp), 255
 [.quad (Extract.quad), 257
 [.splitppp (Extract.splitppp), 258
 [.tess, 21, 896
 [.tess (Extract.tess), 259
 [<- .im, 20
 [← .im (Replace.im), 726
 [← .listof (Extract.listof), 249
 [← .ppp (Extract.ppp), 252
 [← .quad (Extract.quad), 257
 [← .splitppp (Extract.splitppp), 258
 [← .tess, 21
 [← .tess (Extract.tess), 259
 %mark% (marks), 476

 abline, 331
 adaptive.density, 34, 183, 184
 add1, 653
 affine, 18, 19, 35, 36–40, 244, 245, 273, 274,
 557, 724, 728, 730–732, 827, 836,
 837, 846, 848–850
 affine.im, 36, 36, 37, 39, 40
 affine.owin, 35, 36, 37, 37–40, 558
 affine.ppp, 35–37, 38, 40
 affine.psp, 21, 36, 37, 39, 39
 AIC, 28, 457

allstats, 24, 40
 allltypes, 25, 41, 149, 238, 262, 263, 470,
 572, 574, 576, 603, 642, 643
 amacrine, 17, 44, 349, 351–354, 658
 anemones, 17, 45
 angles.psp, 21, 46, 428, 506, 686
 anova, 47, 49, 50
 anova.glm, 47, 49, 50
 anova.lppm, 30, 47
 anova.ppm, 28, 32, 48, 653, 701, 703
 anova.slrm, 30, 50, 858
 ants, 17, 51
 append.psp, 53
 apply, 54, 55, 480
 applynbd, 25, 54, 480, 481
 area.owin, 19, 57, 196, 557, 558, 589, 930
 areaGain, 58, 63, 866
 AreaInter, 28, 59, 59, 62, 63, 333, 571, 647,
 651, 679, 689, 760, 767, 780, 785
 areaLoss, 59, 62, 866
 array, 78
 as.array.im (as.matrix.im), 77
 as.box3, 22, 63, 106, 194, 644
 as.data.frame, 64, 66, 67, 69, 70
 as.data.frame.default, 65
 as.data.frame.hyperframe, 23, 64, 319
 as.data.frame.im, 20, 65
 as.data.frame.ppp, 66
 as.data.frame.ppx (as.hyperframe.ppx),
 70
 as.data.frame.psp, 21, 67
 as.function, 68
 as.function.fv, 68, 280, 281
 as.hyperframe, 22, 23, 69, 70, 71, 122, 319
 as.hyperframe.ppx, 69, 70, 70, 319
 as.im, 20, 34, 71, 79, 146, 207, 241, 312, 326,
 328, 493, 593–596, 839
 as.im.function, 493
 as.im.leverage.ppm, 434
 as.im.owin, 19
 as.im.ppp, 18, 73
 as.im.ppp (pixellate.ppp), 595
 as.interact, 28, 74, 270, 653
 as.interact.ppm, 653
 as.linnet (methods.linnet), 495
 as.linnet.lppm (methods.lppm), 498
 as.mask, 19, 20, 36, 37, 72, 75, 77–79, 83, 92,
 138, 146, 179–181, 183, 185, 200,
 203, 205, 206, 208–211, 236, 237,
 298, 308, 314, 315, 322, 323, 340,
 342, 360, 526, 549, 557, 558,
 594–598, 627, 667, 695, 711, 742,

787, 798, 838, 842
`as.mask.psp`, 21, 76, 597
`as.matrix.im`, 20, 77, 79, 326, 328
`as.matrix.owin`, 78, 78
`as.matrix.ppx` (`as.hyperframe.ppx`), 70
`as.owin`, 18, 57, 75, 79, 83–85, 87, 88, 102,
 104, 105, 124, 139, 157, 159, 163,
 195, 196, 199, 201, 235–237, 276,
 304, 306, 329, 334, 340, 360, 391,
 433, 458, 495, 549, 555–558, 589,
 597, 598, 621, 653, 655, 656, 684,
 685, 707, 713, 714, 733, 737, 745,
 751–754, 757, 779, 784, 793, 795,
 802, 803, 805, 810, 818, 819, 823,
 825, 828, 832, 842, 866, 892, 893
`as.owin.linnet` (`methods.linnet`), 495
`as.owin.ppm`, 653
`as.polygonal`, 19, 20, 76, 82, 589
`as.ppp`, 16, 54, 83, 175, 204, 206, 219, 220,
 247, 248, 263, 264, 277, 288, 291,
 294, 300, 324, 363, 365, 367, 368,
 370, 378, 381, 384, 387, 389, 390,
 393, 395, 400, 404, 407, 412, 431,
 462, 468, 469, 471, 472, 474, 482,
 497, 508, 628, 656, 658, 708, 709,
 726, 727, 758, 786, 841, 907–909
`as.ppp.influence.ppm`, 332
`as.ppp.lpp` (`methods.lpp`), 496
`as.psp`, 21, 54, 85, 495, 497, 631, 684–686,
 866
`as.psp.linnet` (`methods.linnet`), 495
`as.psp.lpp` (`methods.lpp`), 496
`as.rectangle`, 76, 88, 104, 557, 558
`as.tess`, 21, 89, 342, 896
`atan2`, 46
`axis`, 609

`BadGey`, 28, 90, 91, 647, 651, 679, 760, 780,
 785, 835
`barplot`, 316, 317
`bdist.pixels`, 19, 92, 93, 94, 557, 558
`bdist.points`, 19, 92, 93, 94, 557, 558
`bdist.tiles`, 20, 22, 92, 93, 94, 896
`bei`, 17, 95
`bermantest`, 32, 96, 417, 599, 653
`bermantest.ppm`, 653
`betacells`, 17, 98, 284, 658
`bind.fv`, 99, 280, 281
`blur`, 20, 101, 396, 413
`border`, 19, 102
`bounding.box`, 19, 88, 103, 341, 557, 558
`bounding.box.xy`, 104, 104, 159, 745, 893
`box3`, 22, 63, 64, 105, 491, 505, 645, 807, 830

`boxx`, 22, 106, 195, 492, 809, 832
`bramblecanes`, 17, 107, 658
`bronzefilter`, 17, 108
`bw.diggle`, 109, 112, 182–184
`bw.relrisk`, 110, 115, 724, 725, 867
`bw.scott`, 110, 112
`bw.smoothppp`, 113, 860–862
`bw.stoyan`, 111, 114
`by`, 116, 117
`by.im`, 115, 869
`by.ppp`, 18, 116, 896

`cauchy.estK`, 27, 118, 121, 409, 410, 713,
 714, 832, 833, 923
`cauchy.estpcf`, 27, 119, 120, 409, 410, 713,
 714, 832, 833, 925
`cbind`, 100, 122
`cbind.fv`, 142, 143, 280, 281
`cbind.fv` (`bind.fv`), 99
`cbind.hyperframe`, 23, 122, 319
`cells`, 17, 123, 658
`centroid.owin`, 20, 123, 330, 848
`chicago`, 17, 23, 125
`chisq.test`, 701, 703
`chop.tess`, 21, 126, 138
`chorley`, 17, 127
`chull`, 346
`circumradius`, 128
`clarkevans`, 23, 129, 131, 132
`clarkevans.test`, 32, 130, 131
`clf.test`, 32, 132, 227
`clickjoin`, 22, 134
`clickpoly`, 135, 137
`clickppp`, 16, 135, 136, 136, 321
`clip.inflne`, 126, 137
`clip.psp`, 685
`closing`, 19, 138, 549, 557, 558
`coef`, 140, 141, 494, 498
`coef.kppm` (`methods.kppm`), 494
`coef.lppm` (`methods.lppm`), 498
`coef.ppm`, 28, 139, 276, 458, 653, 654, 927
`coef.slrm`, 30, 141, 504, 858
`col2hex` (`colourtools`), 145
`col2rgb`, 145
`collapse.fv`, 142, 147, 148
`colourmap`, 143, 468, 592, 600, 601, 609, 611
`colours`, 144
`colourtools`, 144, 145, 739
`commonGrid`, 19, 20, 146, 151, 312
`compareFit`, 32, 147
`compatible`, 148, 149, 150, 505, 712
`compatible.fasp`, 149, 149
`compatible.fv`, 149, 150

compatible.im, 20, 146, 149, 151, 241, 312
compatible.units, 149
compatible.units (methods.units), 504
compileK, 443
complement.owin, 19, 151, 347, 556–558
concatxy, 153, 894
confint, 653
connected, 19, 20, 154
contour, 493, 624, 626, 627
contour.default, 73, 155, 156
contour.distfun (methods.distfun), 492
contour.im, 20, 155, 493, 592, 611, 866
contour.listof, 156, 620
convexhull, 18–20, 157, 159
convexhull.xy, 105, 158, 158, 179, 347, 745
convolve.im, 20, 159, 329
coords, 18, 22, 160
coords.ppx, 168, 538, 543, 563
coords<- (coords), 160
copper, 17, 161, 736
cor, 530, 531
corners, 29, 163, 175, 176, 708, 709
countends (lineardisc), 441
Covariance, 436, 437, 439, 440, 749
crossdist, 24, 164, 165–169, 559, 560,
 562–565
crossdist.default, 164, 165, 168
crossdist.pp3, 26, 166
crossdist.ppp, 164, 165, 167
crossdist.ppx, 26, 168
crossdist.psp, 164, 165, 168, 169
crossing.psp, 21, 170, 841
cut, 171, 173
cut.default, 171, 172
cut.im, 20, 171, 326, 328, 710
cut.ppp, 18, 25, 117, 172, 253, 529, 871, 896
data, 17, 658
data.frame, 319
data.ppm, 174, 216, 217, 654
default.dummy, 29, 175, 648, 708, 866
default.expand, 176, 178, 227
default.rmhcontrol, 177, 766, 767
delaunay, 18, 22, 178, 201
deltametric, 179
demopat, 17, 180, 658
density, 185, 219, 220, 469, 471, 472, 482,
 577, 578
density.default, 445, 446, 582, 586, 587,
 742, 743, 913
density.ppp, 18, 20, 24, 34, 35, 102, 109,
 110, 112, 181, 186, 189, 323, 382,
 387, 395, 396, 406, 407, 413, 414,
 453, 455, 582, 586, 588, 596, 724,
 725, 741, 844, 845, 859–862
density.psp, 21, 184
density.splitppp, 185, 619, 620
dfbetas, 187
dfbetas.ppm, 32, 186, 333, 435
diagnose.ppm, 32, 188, 216, 217, 464–466,
 693–696, 734, 735
diameter, 192, 558
diameter.box3, 22, 106, 193, 193
diameter.boxx, 22, 107, 193, 194
diameter.linnet (circumradius), 128
diameter.owin, 19, 57, 193, 195, 557, 589
DiggleGatesStibbard, 28, 196, 647, 651,
 679, 717, 760, 780, 785
DiggleGratton, 28, 197, 197, 647, 651, 679,
 718, 719, 760, 766, 767, 775, 780,
 782, 785
dilated.areas, 19, 63, 199
dilation, 19, 102, 103, 139, 177, 199, 200,
 237, 549, 557, 558
dilation.owin, 773
dirichlet, 18, 21, 34, 35, 178, 179, 201
dirichlet.weights, 29, 202, 308, 709
disc, 19, 203, 205, 556, 827
discpartarea, 204
discretise, 18, 205
distfun, 24, 72, 206, 208, 210–212, 493
distfun.owin, 20
distfun.psp, 21
distmap, 24, 179, 180, 199, 207, 207,
 209–212, 315, 328, 330
distmap.owin, 20, 208, 208, 211, 212
distmap.ppp, 208, 209, 210, 212
distmap.psp, 21, 208, 209, 211, 211, 527, 686
dknn (rknn), 747
drop1, 28, 653
dummify, 212
dummy.ppm, 213, 654
duplicated.ppp, 18, 214, 518, 888, 910
ecdf, 242
edge.Ripley, 866
edge.Trans, 866
edges2triangles, 215
eem, 190–192, 216, 466, 696, 734
effectfun, 28, 217
Emark, 25, 219
empty.space (Fest), 263
endpoints.psp, 21, 221
envelope, 24, 29, 32, 41–43, 133, 134, 176,
 177, 222, 229, 230, 232, 234, 280,

- 300, 371, 391, 404, 407, 601, 602,
 641, 642, 653
envelope.envelope, 226, 227, 229, 641, 642
envelope.lpp, 26, 231
envelope.lppm, 30
envelope.lppm(envelope.lpp), 231
envelope.pp3, 22, 26, 224, 233
envelope.ppm, 653
eroded.areas, 19, 199, 235, 237, 557, 558
eroded.volumes, 22, 106, 107
eroded.volumes(diameter.box3), 193
eroded.volumes.boxx, 22, 107
eroded.volumes.boxx(diameter.boxx), 194
erosion, 19, 92, 93, 102, 103, 139, 201, 235,
 236, 236, 329, 347, 549, 557, 558,
 814, 842
eval.fasp, 24, 150, 237, 262, 263
eval.fv, 24, 150, 238, 239, 934, 935
eval.im, 20, 151, 240, 326, 328, 335, 336,
 725, 729, 864, 865, 931
ewcdf, 242, 416
exactdt, 24
exactMLEstrauss, 243
expand.owin, 244, 774, 933
Extract.fasp, 245
Extract.fv, 246
Extract.im, 247
Extract.listof, 249
Extract.lpp, 250
Extract.msr, 251
Extract.ppp, 252
Extract.ppx, 254
Extract.psp, 255
Extract.quad, 257
Extract.splitppp, 258
Extract.tess, 259
extractAIC, 457, 498
extractAIC.lppm(methods.lppm), 498
extractAIC.ppm, 276, 653
extractAIC.ppm(logLik.ppm), 457
F3est, 26, 235, 260, 283, 373, 867
fasp, 643
fasp.object, 40, 42, 43, 238, 245, 246, 262,
 572, 574, 603
Fest, 24, 40–43, 61, 227, 239, 263, 281, 296,
 299, 315, 316, 360, 363, 365–369,
 371, 392
fft, 400
Fiksel, 28, 267, 571, 647, 651, 679, 760, 780,
 785
finpines, 17, 268
fitin, 28, 75, 653
fitin(fitin.ppm), 269
fitin.ppm, 269, 653
fitted, 273
fitted.ppm, 28, 271, 276, 458, 653, 654, 671
fitted.slrm, 30, 272, 858
flipxy, 18, 19, 21, 36, 39, 40, 273, 273, 724
flu, 17, 274
formula, 276, 494, 498, 503, 509
formula.kppm(methods.kppm), 494
formula.lppm(methods.lppm), 498
formula.ppm, 28, 276, 458, 653
formula.slrm(methods.slrm), 503
fryplot, 23, 277, 400, 401
frypoints(fryplot), 277
fv, 68, 100, 279
fv.object, 41, 68, 142, 143, 218, 220, 226,
 227, 240, 246, 247, 265, 279, 280,
 281, 287, 289, 292, 295, 301, 305,
 315, 324, 363, 365, 368, 371, 376,
 379, 382, 385, 388, 391, 393, 397,
 404, 408, 411, 414, 423, 424, 426,
 427, 431, 448, 452, 454, 456, 470,
 473, 475, 482, 507, 572, 575, 576,
 581, 585, 606, 688, 690, 692, 858,
 859, 877, 907, 934, 935
G3est, 26, 166, 235, 262, 282, 373, 536, 537
gam, 311
gam.control, 646
ganglia, 98, 99, 283, 284, 658
GaussRF, 749, 750
Gcom, 32, 148, 284, 304, 305, 377
Gcross, 24, 42, 43, 287, 291, 293, 300, 301,
 363
Gdot, 24, 42, 43, 288, 290, 291, 300, 301, 363,
 366, 371
Gest, 24, 40, 41, 43, 130, 165, 168, 170, 227,
 266, 281, 287–292, 293, 293,
 299–301, 305, 360, 367–369, 392,
 535, 877
Geyer, 28, 90, 91, 296, 296, 551, 568, 569,
 571, 647, 651, 679, 691, 719, 760,
 766, 767, 780, 785, 834, 835
Gfox, 26, 298
glm, 15, 311, 509, 646, 647
glm.control, 646
Gmulti, 24, 25, 288, 290, 291, 293, 299, 363,
 365
gorillas, 17, 302
gpc2owin, 303
Gres, 32, 148, 286, 287, 304, 411, 688, 690,
 692
gridcenters(gridcentres), 306

- gridcentres, 29, 175, 176, 306, 708, 709, 874
gridweights, 29, 202, 307, 709

hamster, 17, 308
Hardcore, 28, 309, 571, 647, 651, 679, 740, 760, 780, 785
harmonic, 310
harmonise.im, 20, 146, 151, 241, 312
harmonize.im(harmonise.im), 312
has.offset(model.depends), 509
heather, 313
Hest, 26, 298, 299, 314
hist, 264, 289, 292, 294, 300, 316, 317, 371, 404, 407, 932
hist.default, 316, 317
hist.im, 20, 316, 328
hsv, 738, 739
hsvim, 20
hsvim(rgbim), 738
humberside, 17, 317
hyperframe, 23, 64, 69–71, 122, 274, 275, 318, 477, 553, 607, 608, 665, 936

identify, 320–322
identify.default, 320
identify.ppp, 18, 135–137, 320, 322
identify.psp, 321
idw, 322, 861, 862
Iest, 25, 323
im, 16, 20, 116, 117, 325, 328, 449, 513, 568, 596, 729, 869, 871, 903
im.object, 35, 72, 95, 155, 156, 171, 184–186, 241, 247, 248, 316, 317, 323, 326, 327, 348, 401, 433, 490, 513, 592, 609, 611, 647, 670, 674, 710, 726, 727, 739, 779, 790, 793, 795, 801, 805, 810, 862, 864, 865, 883, 931
image, 624, 626, 627, 866
image.default, 73, 600, 609–611, 622, 623
image.im(plot.im), 608
image.listof, 619, 620
image.listof(contour.listof), 156
imcov, 20, 160, 328, 842
incircle, 19, 329
inflne, 126, 138, 330
influence, 332
influence.ppm, 32, 187, 332, 435, 611, 612
inforder.family, 333, 569, 571, 905
inside.owin, 19, 306, 334, 874
integral.im, 20, 335, 401
intensity, 336, 337–339
intensity.ppm, 337, 337, 339
intensity.ppp, 337, 338, 338
interp.im, 20, 102, 339
intersect.owin, 19, 340, 342, 347
intersect.tess, 21, 342, 896
iplot, 18, 343, 361, 630
ippm, 187, 332, 344, 434, 651
is.convex, 20, 158, 346
is.empty, 347
is.im, 20, 348
is.marked, 348, 350, 351, 359, 653
is.marked.ppm, 349, 349, 351, 653
is.marked.ppp, 349, 350, 350
is.mask, 20
is.mask(is.rectangle), 357
is.multitype, 351, 353, 354, 653
is.multitype.ppm, 352, 352, 354, 653
is.multitype.ppp, 352, 353, 354
is.owin, 355
is.poisson, 653
is.poisson(is.stationary), 358
is.poisson.ppm, 653
is.polygonal, 20
is.polygonal(is.rectangle), 357
is.ppm, 356
is.ppp, 356
is.psp, 21
is.rectangle, 20, 357
is.stationary, 358, 653
is.stationary.ppm, 653
is.subset.owin, 20, 341, 359
istat, 23, 344, 360

japanesepines, 17, 361
Jcross, 25, 42, 43, 362, 365, 366, 370, 371
Jdot, 25, 42, 43, 362, 364, 364, 370, 371
Jest, 24, 40–43, 227, 266, 281, 296, 299, 324, 325, 360, 362–366, 367, 370, 371, 392
Jfox, 26
Jfox (Gfox), 298
Jmulti, 25, 362, 364–366, 370

K3est, 26, 235, 262, 283, 372, 562, 579, 580
kaplan.meier, 266, 296, 369, 373, 399, 721
Kcom, 32, 148, 287, 374, 411, 867
Kcross, 25, 42, 43, 221, 378, 381, 383, 384, 391, 392, 404, 405, 422, 423, 470, 473, 481, 483, 572–576, 581, 584
Kcross.inhom, 25, 380, 388, 408, 423, 424
Kdot, 25, 42, 43, 221, 378, 380, 383, 385, 387, 388, 391, 392, 404, 405, 425, 426, 470, 473, 483, 572–576, 581, 584, 585

Kdot.inhom, 25, 383, 386, 408, 426, 427
 Kest, 24, 40–43, 68, 118, 119, 133, 150, 220,
 227, 230, 238–240, 266, 278, 281,
 296, 360, 369, 375–380, 382, 384,
 385, 388, 389, 393–395, 397, 401,
 403–405, 409–411, 414, 431, 432,
 436, 437, 447, 448, 451–453, 460,
 469, 472, 485, 486, 507, 559, 560,
 563, 572–578, 581, 606, 643, 644,
 715, 866, 897, 898, 908, 920, 922,
 923, 935
 Kest.fft, 24, 393
 Kinhom, 24, 381, 383, 387, 388, 390, 392, 394,
 407, 409, 410, 447, 448, 454, 460,
 572–576, 588, 589
 km.rs, 266, 296, 369, 374, 398, 721
 Kmeasure, 20, 24, 278, 328, 393, 394, 400
 Kmodel, 119, 402, 867, 923
 Kmodel.kppm, 26, 410
 Kmuli, 25, 378, 380, 384, 385, 391, 392, 403,
 407, 408, 572, 574–576
 Kmuli.inhom, 383, 388, 406
 kppm, 26, 31, 119, 121, 359, 402, 403, 409,
 486, 488, 489, 494, 508, 512, 514,
 613, 666, 713, 714, 750, 752, 753,
 825, 826, 832, 833, 853, 898, 900,
 914, 922–926
 Kres, 32, 148, 305, 377, 411, 688, 690, 692,
 867
 ks.test, 415–417
 Kscaled, 412
 kstest, 32, 97, 613, 614, 653, 701, 703
 kstest (kstest.ppm), 415
 kstest.ppm, 415, 653

 labels, 494, 503
 labels.kppm (methods.kppm), 494
 labels.slrm (methods.slrm), 503
 LambertW, 418
 lansing, 17, 419, 658
 latest.news, 420
 layered, 23, 421, 615
 Lcross, 25, 42, 43, 422, 424, 426
 Lcross.inhom, 25, 423, 427
 Ldot, 25, 42, 43, 423, 425, 427
 Ldot.inhom, 25, 426
 legend, 605
 lengths.psp, 21, 46, 428, 506, 686
 LennardJones, 28, 429, 571, 647, 651, 679,
 719, 760, 777, 780, 784, 785
 Lest, 24, 42, 43, 133, 360, 423, 426, 430, 448,
 452
 letterR, 19, 432

 levelset, 20, 432, 865
 leverage (leverage.ppm), 434
 leverage.ppm, 32, 187, 333, 434, 615, 616
 lgcp.estK, 27, 119, 409, 410, 435, 440, 486,
 489, 506, 508, 750, 867, 898, 923
 lgcp.estpcf, 27, 121, 409, 410, 437, 438, 925
 lineardisc, 22, 441
 linearK, 25, 232, 442, 444, 445
 linearKinhom, 25, 443, 447
 linearpcf, 25, 444, 446, 447
 linearpcfinhom, 25, 445, 446
 lines, 465
 Linhom, 24, 424, 427, 447, 453, 454
 linim, 30, 448, 617, 668
 linnet, 22, 129, 135, 442, 449, 450, 462, 496,
 618, 804, 805, 828
 lm, 15, 311, 509
 load, 916
 localK, 24, 392, 451, 453, 454, 456, 460
 localKinhom, 24, 452, 453, 456, 460
 localL, 24, 453, 454
 localL (localK), 451
 localLinhom, 24, 452
 localLinhom (localKinhom), 453
 localpcf, 24, 455, 460
 localpcfinhom, 24, 460
 localpcfinhom (localpcf), 455
 locator, 135–137
 locfit, 742, 743
 loess, 219, 469, 471, 482
 logLik, 457, 458, 498
 logLik.lppm (methods.lppm), 498
 logLik.ppm, 28, 276, 457, 653
 logLik.slrm, 30, 458, 858
 lohboot, 24, 31, 391, 459, 578, 921
 longleaf, 17, 349, 351–354, 461, 658
 lpp, 16, 23, 125, 251, 443–445, 447, 462, 463,
 464, 497, 561, 668, 805, 828
 lppm, 29, 47, 48, 463, 498, 512, 514, 667
 Lscaled (Kscaled), 412
 lurking, 191, 192, 464, 696
 lut, 144, 467

 mad.test, 32, 225, 227
 mad.test (clf.test), 132
 markconnect, 25, 221, 468, 473, 483, 581, 585
 markcorr, 25, 221, 470, 470, 475, 476, 483
 markcorrint, 25, 473, 474
 markmean, 25
 markmean (smooth.ppp), 860
 marks, 18, 476, 478, 479
 marks.psp, 21, 478, 685
 marks<-, 16

marks<- (marks), 476
 marks<- .lpp (methods.lpp), 496
 marks<- .psp, 21
 marks<- .psp (marks.psp), 478
 markstat, 25, 55, 479, 481
 marktable, 25, 55, 480, 480
 markvar, 25
 markvar (smooth.ppp), 860
 markvario, 25, 221, 470, 473, 481
 matchingdist, 483, 661, 662, 664
 matclust.estK, 27, 409, 410, 437, 485, 489, 506, 508, 752, 753, 898
 matclust.estpcf, 27, 409, 410, 440, 487
 matrix, 326
 max.im (mean.im), 489
 mean, 490
 mean.default, 489, 490
 mean.im, 20, 328, 489
 median, 490
 median.default, 490
 median.im (mean.im), 489
 methods.box3, 490
 methods.boxx, 491
 methods.distfun, 492
 methods.kppm, 410, 494, 914
 methods.linnet, 22, 450, 495
 methods.lpp, 23, 462, 496
 methods.lppm, 464, 498
 methods.pp3, 499
 methods.ppm (ppm.object), 652
 methods.ppx, 462, 497, 500
 methods.rho2hat, 501, 741
 methods.rhohat, 502, 744
 methods.slrm, 503
 methods.units, 504
 midpoints.psp, 21, 46, 222, 428, 505, 686
 min.im (mean.im), 489
 mincontrast, 27, 118–121, 409, 410, 436, 437, 439, 440, 485, 486, 488, 489, 506, 897–900, 922–925
 miplot, 23, 508, 704
 model.covariates (model.depends), 509
 model.depends, 28, 509
 model.frame, 511, 512
 model.frame.glm, 511
 model.frame.kppm (model.frame.ppm), 511
 model.frame.lppm (model.frame.ppm), 511
 model.frame.ppm, 28, 276, 458, 511, 653
 model.images, 28, 512, 514
 model.is.additive (model.depends), 509
 model.matrix, 212, 509, 510, 513, 514
 model.matrix.kppm (model.matrix.ppm), 513
 model.matrix.lm, 512–514
 model.matrix.lppm (model.matrix.ppm), 513
 model.matrix.ppm, 276, 458, 512, 513, 513, 653
 msr, 191, 251, 252, 466, 515, 620, 621, 735, 859, 860
 MultiHard, 28, 310, 517, 520, 521, 571, 760, 780
 multiplicity.ppp, 215, 518, 910
 MultiStrauss, 28, 517, 519, 520, 521, 571, 647, 651, 719, 760, 766, 775, 780, 782, 785
 MultiStraussHard, 28, 517, 520, 571, 647, 651, 719, 760, 766, 775, 780, 782, 785
 murchison, 17, 521
 nbextras (nbfires), 523
 nbfires, 17, 523
 nbw.rect (nbfires), 523
 nearest.neighbour (Gest), 293
 nearest.raster.point, 19, 526, 557, 558
 nearestsegment, 21, 212, 527, 683
 news, 420
 nnclean, 23, 528
 nncorr, 529
 nncross, 21, 24, 210–212, 532, 535, 536, 538, 541–544
 nnndist, 24, 130, 164–166, 168–170, 295, 296, 529, 533, 534, 537–544, 559, 560, 562–565
 nnndist.pp3, 26, 536
 nnndist.ppp, 540
 nnndist.ppx, 26, 537
 nnndist.psp, 535, 539
 nnmean, 25
 nnmean (nncorr), 529
 nnvario, 25
 nnvario (nncorr), 529
 nnwhich, 24, 295, 296, 535–538, 540, 543, 544
 nnwhich.ppp, 26, 541, 542
 nnwhich.ppx, 26, 543
 nobs, 457
 nobs.ppm, 653
 nobs.ppm (logLik.ppm), 457
 npfun, 545
 npoints, 18, 22, 546, 547
 nsegments, 547
 nztrees, 17, 548, 658
 offset, 510

opening, 19, 139, 347, 548, 557, 558
 optim, 118–121, 243, 435, 437–439, 485–488,
 506, 507, 897–900, 922, 924, 925
 options, 865, 868
 Ord, 29, 550, 551, 571, 647, 651, 719
 ord.family, 333, 551, 569, 571, 905
 OrdThresh, 28, 550, 551, 552, 571, 647, 651,
 679, 719, 784
 osteo, 17, 553
 overlap.owin, 341
 owin, 16, 18, 57, 81, 83, 88, 104, 105, 124,
 139, 152, 158, 159, 163, 196, 199,
 201, 203, 205, 236, 237, 256, 306,
 329, 334, 347, 357, 549, 555, 557,
 558, 621, 622, 655–658, 684, 685,
 711, 714, 745, 819, 823, 842, 852,
 866, 876, 911
 owin.object, 57, 72, 75, 76, 80, 81, 85, 87,
 92, 124, 152, 195, 203, 235, 248,
 253, 256, 334, 341, 355, 432, 433,
 526, 555, 556, 557, 589, 623, 655,
 656, 685, 711, 727, 733, 791, 794,
 798, 802, 806, 829, 842, 864, 865,
 876, 904, 929, 930
 owin2gpc (gpc2owin), 303

pairdist, 24, 164–166, 168–170, 535, 537,
 538, 558, 562–565
 pairdist.default, 559, 559, 563
 pairdist.lpp, 26, 560
 pairdist.pp3, 26, 561
 pairdist.ppp, 559, 562, 565
 pairdist.ppx, 26, 563
 pairdist.psp, 559, 563, 564
 PairPiece, 29, 91, 551, 565, 571, 647, 651,
 719, 760, 766, 767, 775, 780, 782,
 785, 834, 835
 pairs, 567, 568
 pairs.default, 567, 568
 pairs.im, 567
 pairsat.family, 91, 333, 551, 568, 571, 835,
 905
 Pairwise, 29, 198, 551, 569, 571, 647, 651,
 719
 pairwise.family, 61, 197, 268, 297, 310,
 333, 430, 517, 520, 521, 551, 566,
 569, 570, 571, 864, 879, 880, 905
 palette, 145
 paletteindex (colourtools), 145
 par, 136, 137, 568, 603, 607, 610, 619, 630,
 632, 633, 635
 parent.frame, 936

pcf, 24, 115, 120, 121, 227, 360, 379, 380,
 382–385, 388, 391, 392, 397,
 403–405, 408, 410, 414, 432, 439,
 440, 448, 456, 460, 487, 489, 571,
 573, 575, 576, 578, 580, 581, 584,
 585, 589, 899, 900, 924, 925
 pcf.fasp, 572, 573
 pcf.fv, 414, 572, 575
 pcf.ppp, 120, 438, 487, 572, 576, 580,
 581, 583–589, 899, 924
 pcf3est, 26, 235, 262, 283, 373, 578
 pcfcross, 25, 42, 470, 580, 583–585
 pcfcross.inhom, 25, 582, 587
 pcfdot, 25, 581, 584, 587
 pcfdot.inhom, 25, 583, 585
 pcfinhom, 24, 410, 456, 460, 583, 587, 587
 pcfmodel, 121, 925
 pcfmodel (Kmodel), 402
 pcfmodel.kppm, 27, 410
 perimeter, 19, 57, 196, 557, 589
 periodify, 18, 19, 21, 590, 846, 848–850
 persp, 493, 624, 626, 627, 866
 persp.default, 592
 persp.distfun (methods.distfun), 492
 persp.im, 20, 156, 328, 493, 591, 611
 pictex, 622
 pixellate, 20, 593, 595–597, 672, 856
 pixellate.owin, 19, 593, 594
 pixellate.ppp, 18, 593, 595, 595
 pixellate.psp, 21, 77, 593, 596
 pixelquad, 29, 597, 648
 pknn (rknn), 747
 plot, 493, 498, 500–502, 608, 613, 615, 619,
 621, 627, 630, 632, 634, 636, 679,
 698
 plot.bermantest, 97, 599
 plot.colourmap, 144, 600
 plot.default, 189, 465, 613, 628
 plot.diagppm (diagnose.ppm), 188
 plot.distfun, 207
 plot.distfun (methods.distfun), 492
 plot.ecdf, 599
 plot.envelope, 225–227, 601
 plot.fasp, 40, 42, 43, 263, 602
 plot.fv, 24, 41, 68, 226, 227, 239, 265, 280,
 281, 295, 315, 324, 368, 391, 423,
 426, 431, 448, 452, 454, 456, 501,
 502, 581, 585, 601–603, 604, 612,
 613, 866, 907
 plot.hyperframe, 23, 319, 607, 936
 plot.im, 20, 156, 248, 328, 493, 568, 592,
 608, 616, 617, 620, 621, 634, 636,

- 866
- `plot.inflne (inflne)`, 330
- `plot.influence.ppm`, 332, 333, 611
- `plot.kppm`, 26, 410, 494, 612, 666, 914
- `plot.kstest`, 416, 417, 613
- `plot.layered`, 23, 421, 614
- `plot.leverage.ppm`, 434, 435, 615
- `plot.lininm`, 30, 449, 616
- `plot.linnet`, 617
- `plot.listof`, 41, 157, 250, 513, 618, 873, 884
- `plot.lppm (methods.lppm)`, 498
- `plot.minconfit`, 613
- `plot.msr`, 516, 620, 734, 860
- `plot.owin`, 19, 557, 558, 621, 629–632, 636, 866
- `plot.plotppm`, 623, 627
- `plot.pp3`, 22, 625
- `plot.ppm`, 27, 276, 458, 612, 613, 624, 626, 653, 654, 670, 671, 675, 866
- `plot.ppp`, 18, 320, 343, 344, 611, 620, 621, 623, 628, 633, 635, 636, 656, 658, 866
- `plot.ppx (methods.ppx)`, 500
- `plot.psp`, 21, 321, 618, 631, 686
- `plot.qqppm`, 695
- `plot.quad`, 632, 678, 698
- `plot.rho2hat (methods.rho2hat)`, 501
- `plot.rhohat (methods.rhohat)`, 502
- `plot.slrm`, 30, 504, 634, 858
- `plot.splitppp`, 258, 635, 870, 871
- `plot.tess`, 21, 636, 896
- `points`, 568, 628–630
- `pointsOnLines`, 21, 637, 831
- `Poisson`, 28, 74, 551, 571, 638, 647, 651, 679, 719, 760, 766, 767, 780, 785
- `poly`, 650
- `polygon`, 617, 622, 623
- `polypath`, 622
- `ponderosa`, 17, 639
- `pool`, 640, 641–644, 712
- `pool.envelope`, 226, 227, 641, 641, 643
- `pool.fasp`, 641, 642, 642
- `pool.rat`, 641, 643
- `pp3`, 16, 22, 64, 106, 161, 235, 500, 625, 644, 665, 807, 830
- `ppm`, 27, 31, 47, 49, 50, 60, 61, 74, 75, 91, 97, 140, 147, 148, 174, 178, 188, 189, 192, 196–198, 213, 214, 216–218, 227, 243, 244, 267, 268, 270–272, 276, 285–287, 297, 305, 309–311, 344, 345, 349, 352, 353, 359, 375–377, 410, 411, 417, 429, 430, 457, 458, 463, 464, 466, 509, 510, 512–514, 517, 519–521, 551, 552, 566, 570, 598, 624, 626, 627, 638, 645, 653, 654, 668–671, 675, 678, 679, 681, 682, 687, 689–691, 694, 696, 698, 699, 708, 709, 719, 734, 735, 760, 765–767, 775, 780, 782, 784, 785, 834, 835, 854, 863, 864, 866, 878–882, 906, 915, 919, 927, 928
- `ppm.object`, 61, 140, 174, 192, 197, 198, 213, 214, 217, 268, 270, 272, 297, 310, 356, 430, 513, 514, 517, 520, 521, 551, 552, 566, 570, 626, 627, 649, 651, 652, 668, 669, 671, 675, 696, 699, 735, 765, 766, 835, 864, 879–881, 886, 906, 915
- `ppp`, 16, 84, 85, 117, 161, 179, 201, 206, 227, 450, 477, 556, 638, 651, 655, 658, 760, 808, 831, 841, 892, 911
- `ppp.object`, 44, 45, 52, 55, 84, 85, 93, 95, 98, 107, 109, 123, 127, 162, 171, 173–175, 181, 184, 186, 214, 215, 222, 248, 253, 264, 269, 284, 294, 309, 317, 322–324, 344, 361, 368, 390, 419, 461, 477, 480, 481, 518, 522, 546, 548, 556, 629, 630, 635, 636, 640, 648, 656, 657, 721, 722, 727, 758, 766, 767, 791, 794, 799, 802, 806, 829, 841, 843, 850, 861, 862, 871, 875, 895, 909, 910, 912, 918
- `pppdist`, 25, 484, 659, 662, 664
- `pppmatching`, 662, 664
- `pppmatching.object`, 484, 660–662, 663
- `ppx`, 16, 22, 70, 71, 107, 161, 255, 477, 501, 645, 664, 809, 832, 873
- `predict`, 502, 666, 667, 672
- `predict.glm`, 650, 671
- `predict.kppm`, 26, 410, 494, 666, 914
- `predict.lppm`, 30, 464, 667
- `predict.ppm`, 27, 218, 271, 272, 276, 337, 458, 626, 627, 647, 650, 653, 654, 666, 668, 675, 866
- `predict.rhohat (methods.rhohat)`, 502
- `predict.slrm`, 30, 504, 634, 672, 855, 858
- `predict.smooth.spline`, 574–576
- `print`, 331, 491–495, 497–503, 505, 673, 674, 676, 677, 679
- `print.box3 (methods.box3)`, 490
- `print.boxx (methods.boxx)`, 491
- `print.distfun (methods.distfun)`, 492

print.im, 328, [673](#)
print.inflne (inflne), [330](#)
print.kppm (methods.kppm), [494](#)
print.linnet (methods.linnet), [495](#)
print.listof, [619](#), [620](#)
print.lpp (methods.lpp), [496](#)
print.lppm (methods.lppm), [498](#)
print.owin, [558](#), [674](#), [676](#), [677](#), [885](#)
print.pp3, [546](#), [645](#)
print.pp3 (methods.pp3), [499](#)
print.ppm, [28](#), [140](#), [627](#), [647](#), [653](#), [654](#), [671](#),
[675](#), [867](#)
print.ppp, [674](#), [676](#), [888](#)
print.ppx, [546](#), [665](#)
print.ppx (methods.ppx), [500](#)
print.psp, [21](#), [677](#), [889](#)
print.qqppm, [695](#)
print.quad, [677](#)
print.rho2hat (methods.rho2hat), [501](#)
print.rhohat (methods.rhohat), [502](#)
print.slrm (methods.slrm), [503](#)
print.summary.im (summary.im), [883](#)
print.summary.lpp (methods.lpp), [496](#)
print.summary.pp3 (methods.pp3), [499](#)
print.summary.ppm (summary.ppm), [886](#)
print.summary.quad (summary.quad), [889](#)
print.units (methods.units), [504](#)
profilepl, [267](#), [649](#), [651](#), [678](#)
progressreport, [680](#), [866](#)
project.ppm, [28](#), [646](#), [650](#), [651](#), [681](#), [867](#), [919](#)
project2segment, [21](#), [211](#), [212](#), [527](#), [682](#)
psp, [16](#), [21](#), [54](#), [86](#), [87](#), [138](#), [450](#), [638](#), [684](#),
[685](#), [686](#), [751](#), [803](#), [808](#), [831](#), [866](#),
[892](#)
psp.object, [86](#), [87](#), [162](#), [171](#), [185](#), [222](#), [256](#),
[479](#), [522](#), [547](#), [632](#), [684](#), [685](#), [685](#),
[800](#), [841](#), [912](#)
psst, [32](#), [148](#), [287](#), [305](#), [377](#), [411](#), [545](#), [686](#),
[690](#), [692](#)
psstA, [32](#), [148](#), [287](#), [305](#), [377](#), [411](#), [688](#), [688](#),
[692](#), [867](#)
psstG, [32](#), [148](#), [287](#), [305](#), [377](#), [411](#), [688](#), [690](#),
[691](#), [867](#)

qknn (rknn), [747](#)
qqplot.ppm, [32](#), [176](#), [177](#), [189](#), [191](#), [192](#), [466](#),
[693](#)
quad, [29](#)
quad.object, [84](#), [164](#), [175](#), [176](#), [202](#), [257](#),
[306](#), [308](#), [598](#), [633](#), [648](#), [678](#), [697](#),
[699](#), [708](#), [709](#), [874](#), [889](#), [908](#), [909](#)
quad.ppm, [271](#), [466](#), [511](#), [514](#), [515](#), [654](#), [698](#),
[734](#)

quadform (sumouter), [891](#)
quadrat.test, [32](#), [89](#), [97](#), [417](#), [653](#), [700](#), [703](#),
[704](#), [707](#)
quadrat.test.ppm, [653](#)
quadrat.test.hpp, [702](#)
quadrat.test.splitppp, [700](#), [701](#), [702](#)
quadratcount, [24](#), [89](#), [509](#), [700](#), [701](#), [703](#),
[703](#), [706](#), [707](#), [743](#)
quadratresample, [17](#), [31](#), [32](#), [701](#), [703](#), [704](#),
[705](#), [707](#)
quadrats, [21](#), [701](#), [703](#), [704](#), [706](#), [706](#), [896](#),
[921](#)
quadscheme, [29](#), [49](#), [153](#), [164](#), [176](#), [285](#), [306](#),
[375](#), [515](#), [598](#), [648](#), [651](#), [678](#), [687](#),
[689](#), [691](#), [698](#), [708](#), [734](#), [735](#), [819](#),
[823](#), [874](#), [894](#)
quantile, [459](#), [710](#)
quantile.default, [710](#)
quantile.im, [20](#), [328](#), [490](#), [710](#)

range, [490](#)
range.default, [490](#)
range.im (mean.im), [489](#)
raster.x, [19](#), [556](#)–[558](#), [711](#)
raster.xy (raster.x), [711](#)
raster.y, [19](#), [556](#)–[558](#)
raster.y (raster.x), [711](#)
rat, [643](#), [644](#), [712](#)
rbind, [122](#)
rbind.hyperframe, [23](#), [319](#)
rbind.hyperframe (cbind.hyperframe), [122](#)
rCauchy, [17](#), [27](#), [31](#), [119](#), [121](#), [713](#), [796](#)
rcell, [17](#), [31](#), [714](#), [806](#)
rDGS, [16](#), [197](#), [715](#), [821](#)
rDiggleGratton, [16](#), [717](#), [717](#), [740](#), [821](#)
reach, [49](#), [176](#), [177](#), [654](#), [718](#)
reach.ppm, [654](#)
read.table, [658](#), [840](#)
reduced.sample, [266](#), [296](#), [369](#), [374](#), [392](#),
[399](#), [720](#)
redwood, [17](#), [658](#), [721](#), [722](#), [723](#)
redwoodfull, [18](#), [722](#), [722](#)
reflect, [18](#), [36](#), [159](#), [160](#), [274](#), [723](#)
relrisk, [24](#), [111](#), [183](#), [184](#), [724](#), [839](#)
Replace.im, [726](#)
rescale, [505](#), [727](#), [728](#)–[731](#), [911](#)
rescale.im, [728](#)
rescale.own, [728](#), [729](#), [730](#), [731](#)
rescale.hpp, [728](#), [730](#)
rescale.psp, [731](#)
rescale.units (methods.units), [504](#)
rescue.rectangle, [37](#), [732](#), [798](#)

reset.spatstat.options
 (spatstat.options), 865
 residuals.ppm, 28, 190–192, 217, 276, 458,
 466, 514, 515, 653, 694, 696, 733,
 859
 residualspaper, 18, 32, 736
 rGaussPoisson, 17, 31, 737, 750, 753, 796,
 806, 811, 826
 rgb, 738, 739
 rgb2hex (colourtools), 145
 rgbim, 20, 738
 rHardcore, 16, 717, 718, 739, 821
 rho2hat, 501, 740, 744
 rhohat, 23, 502, 503, 741, 742
 ripras, 19, 105, 159, 744, 840, 841, 893
 rjitter, 16, 17, 31, 32, 746
 rknn, 24, 747
 rlabel, 17, 748
 rLGCP, 27, 31, 749, 853
 rlinegrid, 21, 31, 751
 rMatClust, 17, 27, 31, 486, 488, 489, 658,
 738, 750, 752, 754, 755, 796, 806,
 811, 826, 853
 rMaternI, 16, 31, 658, 753, 755, 806, 818
 rMaternII, 16, 31, 658, 754, 806, 818
 rmh, 17, 31, 81, 176, 177, 225, 646, 648, 654,
 694, 696, 716–718, 740, 755, 757,
 760, 766, 767, 770, 772–776, 779,
 780, 782, 784–786, 820–822, 865,
 866, 888
 rmh.default, 694, 756, 757, 766, 767, 806,
 854
 rmh.ppm, 28, 29, 566, 654, 756, 760, 765, 854
 rmhcontrol, 176–178, 695, 696, 757–759,
 765–767, 769, 774, 775, 780, 782,
 784–786, 854, 866, 917
 rmhcontrol.default, 178, 867
 rmhexpand, 177, 244, 245, 769, 770, 772, 772,
 933
 rmhmodel, 654, 719, 757, 767, 770, 772, 774,
 774, 776, 782, 784–786
 rmhmodel.default, 775, 775, 782, 785
 rmhmodel.list, 775, 781, 785
 rmhmodel.ppm, 654, 775, 782, 783
 rmhstart, 758, 765–767, 770–772, 775, 780,
 782, 785, 785, 854
 rMosaicField, 31, 787, 788
 rMosaicSet, 31, 787, 788
 rmpoint, 16, 31, 766, 789, 794, 806
 rmpoispp, 16, 31, 756, 766, 790, 792, 805, 806
 rNeymanScott, 16, 31, 658, 714, 738, 750,
 753, 795, 826, 833
 rotate, 18, 19, 36, 37, 39, 40, 244, 245, 274,
 557, 728, 730–732, 797, 846,
 848–850
 rotate.owin, 558, 797, 798, 799, 800
 rotate.ppp, 797, 799, 800
 rotate.psp, 21, 800
 rpoint, 16, 31, 766, 790, 801, 806, 827, 829
 rpoisline, 21, 31, 751, 802, 804
 rpoislinetess, 22, 31, 787, 788, 803
 rpoislpp, 23, 26, 462, 804, 828
 rpoispp, 16, 31, 658, 714, 738, 750, 753–756,
 766, 794–796, 805, 808, 810, 811,
 818, 826, 829, 833, 855
 rpoispp3, 22, 235, 806, 830
 rpoisppOnLines, 17, 31, 804, 807
 rpoisppx, 22, 809, 832
 rPoissonCluster, 16, 810
 rshift, 17, 31, 32, 811, 814, 816, 817
 rshift.ppp, 812, 812, 815–817
 rshift.psp, 812, 814, 815
 rshift.splitppp, 812, 816
 rSSI, 16, 31, 658, 806, 817
 rstrat, 16, 29, 31, 715, 806, 818, 823
 rStrauss, 16, 31, 717, 718, 739, 740, 760,
 777, 806, 819, 822
 rStraussHard, 821, 821
 rsyst, 16, 31, 715, 819, 823
 rthin, 17, 31, 33, 824
 rThomas, 17, 27, 31, 658, 738, 753, 796, 806,
 811, 825, 853, 898, 900
 rug, 501, 502
 runifdisc, 16, 31, 826
 runiflpp, 23, 26, 462, 805, 827
 runifpoint, 16, 31, 658, 715, 801, 802, 806,
 819, 823, 827, 828, 831, 866
 runifpoint3, 22, 807, 829
 runifpointOnLines, 17, 31, 638, 808, 828,
 830
 runifpointx, 22, 809, 831
 rVarGamma, 17, 27, 31, 796, 832, 922–925
 samecolour (colourtools), 145
 sample, 801, 829
 SatPiece, 29, 91, 297, 551, 568, 569, 647,
 651, 833, 834, 835
 Saturated, 29, 551, 568, 569, 571, 647, 651,
 719, 835
 scalardilate, 18, 836
 scale, 837
 scaletointerval, 20, 837
 scan.test, 24, 32, 838
 scanpp, 658, 840
 segments, 134, 631, 632, 636

selfcrossing.psp, 21, 171, 841
 set.seed, 760, 852, 855
 setcov, 20, 328, 329, 842
 setmarks (marks), 476
 setminus.owin, 19
 setminus.owin (intersect.owin), 340
 shapley, 18, 843
 sharpen, 844
 sharpen.ppp, 18, 23, 24, 183, 184
 shift, 18, 19, 36, 37, 39, 40, 244, 245, 274,
 557, 591, 728, 730–732, 837, 845,
 846–850
 shift.im, 20, 846
 shift.owin, 558, 846, 847, 849, 850
 shift.ppp, 846, 848, 848, 850
 shift.psp, 21, 849
 shortside (diameter.box3), 193
 shortside.box3, 22
 shortside.boxx, 22
 shortside.boxx (diameter.boxx), 194
 simdat, 18, 658, 850
 simplenet, 22, 450, 851
 simplify.owin, 19, 82, 83, 589, 851
 simulate, 853–855
 simulate.kppm, 26, 31, 119, 121, 225, 410,
 494, 852, 854, 855, 914, 922, 924
 simulate.ppm, 17, 28, 29, 31, 176, 276, 458,
 653, 767, 853, 853, 855
 simulate.slrm, 30, 504, 855
 slrm, 30, 50, 51, 141, 273, 359, 458, 459, 503,
 504, 634, 672, 673, 855, 856
 smooth.fv, 24, 858
 smooth.msr, 516, 621, 859
 smooth.ppp, 18, 24, 102, 113, 114, 183, 184,
 323, 596, 620, 725, 845, 860
 smooth.spline, 574–576, 858, 859
 Softcore, 29, 551, 571, 647, 651, 679, 719,
 760, 766, 767, 775, 780, 782, 785,
 862
 solutionset, 20, 328, 433, 864
 source, 916
 spatstat (spatstat-package), 14
 spatstat-package, 14
 spatstat.options, 18, 19, 28, 73, 76, 156,
 261, 393, 394, 401, 493, 513, 606,
 611, 622–624, 627, 629, 670, 671,
 682, 690, 770, 772, 865
 split, 868, 872, 891
 split.im, 116, 868
 split.ppp, 18, 117, 186, 253, 258, 523, 529,
 635, 636, 704, 812, 817, 869, 890,
 891, 896

split.ppx, 871
 split<-.ppp (split.ppp), 869
 spokes, 29, 175, 176, 708, 709, 873
 spruces, 18, 875
 square, 19, 556, 876
 step, 28, 457, 653, 857
 stepfun, 778, 779
 stieltjes, 877
 stratrand, 175, 176, 306, 708, 709, 874
 Strauss, 29, 74, 296, 297, 310, 517, 520, 521,
 551, 566, 571, 638, 647, 651, 678,
 679, 682, 719, 760, 766, 767, 775,
 780, 782, 784, 785, 821, 878, 919
 StraussHard, 29, 268, 310, 551, 571, 647,
 651, 679, 719, 760, 766, 767, 775,
 780, 782, 785, 822, 879
 suffstat, 881
 sum, 490
 sum.im (mean.im), 489
 summary, 20, 23, 29, 495, 497, 499, 505,
 883–886, 888–891
 summary.im, 317, 328, 490, 674, 883
 summary.linnet (methods.linnet), 495
 summary.listof, 250, 884
 summary.lpp (methods.lpp), 496
 summary.owin, 558, 674, 885, 888, 889
 summary.pp3 (methods.pp3), 499
 summary.ppm, 28, 276, 359, 458, 886
 summary.ppp, 676, 758, 885, 887
 summary.psp, 21, 46, 428, 506, 677, 888
 summary.quad, 678, 889
 summary.splitppp, 258, 890
 summary.units (methods.units), 504
 sumouter, 891
 superimpose, 18, 21, 54, 153, 658, 870, 871,
 892
 swedishpines, 18, 658, 894
 symbols, 628–630, 632

table, 481
 terms, 276, 494, 498, 503, 509
 terms.kppm (methods.kppm), 494
 terms.lppm (methods.lppm), 498
 terms.ppm, 458, 653
 terms.ppm (formula.ppm), 276
 terms.slrm (methods.slrm), 503
 tess, 16, 21, 90, 94, 116, 117, 155, 172, 173,
 179, 201, 259, 260, 342, 636, 637,
 707, 869, 871, 895, 901, 902, 921
 thomas.estK, 27, 119, 409, 410, 437, 486,
 489, 506, 508, 825, 826, 897, 900,
 923

thomas.estpcf, 27, 121, 409, 410, 440, 489, 899, 925
tile.areas, 22, 896, 901, 902
tiles, 21, 896, 901, 902
transect.im, 20, 903
trim.rectangle, 904
triplet.family, 905, 906
Triplets, 29, 760, 775, 780, 785, 905, 905
Tstat, 24, 907
txtProgressBar, 681

union.owin, 19
union.owin(intersect.owin), 340
union.quad, 698, 908
unique.ppp, 18, 215, 518, 909
uniroot, 418
unit.square(square), 876
unitname, 106, 107, 491, 492, 495, 497, 499–501, 505, 654, 728–731, 910
unitname.box3, 22
unitname.box3(methods.box3), 490
unitname.boxx(methods.boxx), 491
unitname.linnet(methods.linnet), 495
unitname.lpp(methods.lpp), 496
unitname.pp3, 22
unitname.pp3(methods.pp3), 499
unitname.ppm, 654
unitname.ppx, 22
unitname.ppx(methods.ppx), 500
unitname<-(unitname), 910
unitname<- .box3(methods.box3), 490
unitname<- .boxx(methods.boxx), 491
unitname<- .linnet(methods.linnet), 495
unitname<- .lpp(methods.lpp), 496
unitname<- .pp3(methods.pp3), 499
unitname<- .ppx(methods.ppx), 500
units, 732
unmark, 18, 250, 253, 255, 477, 497, 912
unmark.lpp(methods.lpp), 496
unmark.psp, 21
unnormdensity, 913
update, 498, 915–917
update.kppm, 26, 410, 494, 914
update.lppm(methods.lppm), 498
update.ppm, 28, 276, 458, 653, 694, 915
update.rmhcontrol, 178, 766, 767, 917
update.slrm, 504
urkiola, 18, 918

valid.ppm, 28, 650, 651, 682, 919
varblock, 24, 31, 227, 391, 460, 706, 920
vargamma.estK, 27, 119, 409, 410, 921, 925
vargamma.estpcf, 27, 121, 409, 410, 923, 923
vcov, 926, 927
vcov.kppm, 26, 410, 494, 666, 914, 925
vcov.ppm, 28, 276, 458, 653, 926, 926
vcov.slrm, 504
vertices, 347, 929
Vmark, 25
Vmark(Emark), 219
volume, 930
volume.box3, 22, 106, 930
volume.box3(diameter.box3), 193
volume.boxx, 22, 107, 930
volume.boxx(diameter.boxx), 194
volume.owin(area.owin), 57

which.max.im, 725, 931
whist, 932
will.expand, 774, 933
with, 934–936
with.fv, 24, 100, 859, 934
with.hyperframe, 23, 319, 608, 935

X11, 622
X11.options, 622
xfig, 622
xy.coords, 796, 903

zapsmall, 937
zapsmall.im, 20, 936