Package 'spatstat.geom'

September 18, 2026

Version 3.8-3

Date 2026-09-18

Title Geometrical Functionality of the 'spatstat' Family

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

Depends R (>= 3.5.0), spatstat.data (>= 3.1-9), spatstat.univar (>= 3.2-0), stats, graphics, grDevices, utils, methods

Imports spatstat.utils (>= 3.2-5), deldir (>= 2.0-2), polyclip (>= 1.10)

Suggests spatstat.random (>= 3.5), spatstat.explore (>= 3.8), spatstat.model (>= 3.7), spatstat.linnet (>= 3.5), spatial, fftwtools (>= 0.9-8), spatstat (>= 3.6)

Description Defines spatial data types and supports geometrical operations on them. Data types include point patterns, windows (domains), pixel images, line segment patterns, tessellations and hyperframes. Capabilities include creation and manipulation of data (using command line or graphical interaction), plotting, geometrical operations (rotation, shift, rescale, affine transformation), convex hull, discretisation and pixellation, Dirichlet tessellation, Delaunay triangulation, pairwise distances, nearest-neighbour distances, distance transform, morphological operations (erosion, dilation, closing, opening), quadrat counting, geometrical measurement, geometrical covariance, colour maps, calculus on spatial domains, Gaussian blur, level sets of images, transects of images, intersections between objects, minimum distance matching. (Excludes spatial data on a network, which are supported by the package 'spatstat.linnet'.)

License GPL (>= 2)

URL http://spatstat.org/

NeedsCompilation yes

ByteCompile true

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

Author Adrian Baddeley [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0001-9499-8382>),
Rolf Turner [aut, cph] (ORCID: <https://orcid.org/0000-0001-5521-5218>),
Ege Rubak [aut, cph] (ORCID: <https://orcid.org/0000-0002-6675-533X>),
Warick Brown [ctb],
Tilman Davies [ctb],
Ute Hahn [ctb],
Martin Hazelton [ctb],
Abdollah Jalilian [ctb],
Greg McSwiggan [ctb, cph],
Sebastian Meyer [ctb, cph],
Jens Oehlschlaegel [ctb, cph],
Suman Rakshit [ctb],
Dominic Schuhmacher [ctb],
Rasmus Waagepetersen [ctb]

Repository CRAN

Date/Publication 2026-09-18 09:20:02 UTC

Contents

spatstat.geom-package 12
add.texture 23
affine 24
affine.im 25
affine.owin 26
affine.ppp 27
affine.psp 28
affine.tess 29
angles.psp 31
anylist 32
anyNA.im 33
append.psp 34
applynbd 35
area.owin 38
areaGain 39
areaLoss 40
as.box3 42
as.boxx 43
as.colourmap 44
as.data.frame.hyperframe 45
as.data.frame.im 46
as.data.frame.owin 47
as.data.frame.ppp 48
as.data.frame.psp 49
as.data.frame.tess 50
as.function.im 51

as.function.owin . . . 52
as.function.tess . . . 53
as.hyperframe . . . 54
as.hyperframe.ppx . . . 55
as.im . . . 57
as.layered . . . 63
as.mask . . . 64
as.matrix.im . . . 67
as.matrix.owin . . . 68
as.owin . . . 69
as.polygonal . . . 72
as.ppp . . . 74
as.psp . . . 76
as.rectangle . . . 79
as.solist . . . 80
as.tess . . . 81
bdist.pixels . . . 82
bdist.points . . . 84
bdist.tiles . . . 85
beachcolours . . . 86
border . . . 87
bounding.box.xy . . . 88
boundingbox . . . 89
boundingcircle . . . 91
box3 . . . 92
boxx . . . 93
bufftess . . . 94
by.im . . . 96
by.ppp . . . 97
cbind.hyperframe . . . 99
centroid.owin . . . 100
chop.tess . . . 101
clickbox . . . 102
clickdist . . . 103
clickpoly . . . 104
clickppp . . . 105
clip.infline . . . 106
closepairs . . . 107
closepairs.pp3 . . . 109
closetriples . . . 111
closing . . . 112
colourmap . . . 114
colouroutputs . . . 116
colourtools . . . 118
commonGrid . . . 120
compatible . . . 121
compatible.im . . . 122
complement.owin . . . 123

concatxy . . . 124
connected . . . 125
connected.ppp . . . 128
connected.tess . . . 129
contour.im . . . 130
contour.imlist . . . 132
convexhull . . . 133
convexhull.xy . . . 134
convexify . . . 135
convexmetric . . . 136
convolve.im . . . 138
coords . . . 139
corners . . . 141
covering . . . 142
crossdist . . . 143
crossdist.default . . . 144
crossdist.pp3 . . . 145
crossdist.ppp . . . 146
crossdist.ppx . . . 147
crossdist.psp . . . 148
crossing.psp . . . 150
cut.im . . . 151
cut.ppp . . . 152
default.colourmap . . . 154
default.dummy . . . 156
default.image.colours . . . 157
default.symbolmap . . . 158
default.symbolmap.ppp . . . 159
delaunay . . . 163
delaunayDistance . . . 164
deltametric . . . 165
diameter . . . 166
diameter.box3 . . . 167
diameter.boxx . . . 168
diameter.owin . . . 170
dilated.areas . . . 171
dilation . . . 172
dirichlet . . . 173
dirichletAreas . . . 174
dirichletVertices . . . 175
dirichletWeights . . . 176
disc . . . 178
discpartarea . . . 180
discretise . . . 181
discs . . . 182
distbdry . . . 184
distfun . . . 185
distmap . . . 187

distmap.owin . . . 188
distmap.ppp . . . 190
distmap.psp . . . 192
domain . . . 193
duplicated.ppp . . . 195
edges . . . 196
edges2triangles . . . 197
edges2vees . . . 199
edit.hyperframe . . . 200
edit.ppp . . . 201
ellipse . . . 202
endpoints.psp . . . 203
eroded.areas . . . 204
erosion . . . 205
erosionAny . . . 207
eval.im . . . 208
Extract.anylist . . . 209
Extract.hyperframe . . . 210
Extract.im . . . 213
Extract.layered . . . 216
Extract.listof . . . 217
Extract.owin . . . 219
Extract.ppp . . . 220
Extract.ppx . . . 223
Extract.psp . . . 225
Extract.quad . . . 226
Extract.solist . . . 227
Extract.splitppp . . . 229
Extract.tess . . . 230
extrapolate.psp . . . 231
fardist . . . 232
FeretBox . . . 233
FeretDiamFun . . . 234
fillholes.owin . . . 235
flipxy . . . 236
fourierbasis . . . 237
Frame . . . 239
framedist.pixels . . . 240
funxy . . . 241
gridcentres . . . 242
gridweights . . . 244
grow.boxx . . . 245
grow.rectangle . . . 246
harmonise . . . 247
harmonise.im . . . 248
harmonise.owin . . . 249
harmoniseLevels . . . 250
has.close . . . 251

headtail . . . 252
hextess . . . 254
hist.funxy . . . 255
hist.im . . . 256
hyperframe . . . 258
identify.ppp . . . 260
identify.psp . . . 261
identify.tess . . . 262
im . . . 263
im.apply . . . 265
im.object . . . 267
imcov . . . 268
incircle . . . 269
infline . . . 271
inside.boxx . . . 272
inside.owin . . . 273
integral.im . . . 275
integral.tessfun . . . 276
intensity . . . 277
intensity.ppp . . . 278
intensity.ppx . . . 280
intensity.psp . . . 281
intensity.quadratcount . . . 282
interp.colourmap . . . 283
interp.im . . . 284
intersect.boxx . . . 285
intersect.owin . . . 286
intersect.tess . . . 288
invoke.metric . . . 290
invoke.symbolmap . . . 291
is.boxx . . . 292
is.connected . . . 293
is.connected.ppp . . . 294
is.convex . . . 295
is.empty . . . 296
is.im . . . 297
is.linim . . . 298
is.linnet . . . 298
is.lpp . . . 299
is.marked . . . 300
is.marked.ppp . . . 301
is.multitype . . . 302
is.multitype.ppp . . . 303
is.na.hyperframe . . . 304
is.na.solist . . . 305
is.NAobject . . . 306
is.owin . . . 307
is.ppp . . . 307

is.rectangle . . . 308
is.subset.owin . . . 309
layered . . . 310
layerplotargs . . . 311
layout.boxes . . . 312
lengths_psp . . . 313
levelset . . . 314
levelset.distfun . . . 316
lut . . . 317
marks . . . 319
marks.psp . . . 321
marks.tess . . . 322
markstat . . . 323
matchingdist . . . 325
Math.im . . . 327
Math.imlist . . . 329
maxnndist . . . 331
mean.im . . . 332
mergeLevels . . . 333
mergeTiles . . . 335
methods.box3 . . . 336
methods.boxx . . . 337
methods.distfun . . . 338
methods.funxy . . . 340
methods.layered . . . 341
methods.pp3 . . . 342
methods.ppx . . . 344
methods.unitname . . . 345
metric.object . . . 346
midpoints.psp . . . 347
MinkowskiSum . . . 348
multiplicity.ppp . . . 350
NAobject . . . 351
nearest.raster.point . . . 352
nearestsegment . . . 353
nearestValue . . . 354
nestsplit . . . 355
nncross . . . 356
nncross.pp3 . . . 359
nncross.ppx . . . 362
nndist . . . 364
nndist.pp3 . . . 367
nndist.ppx . . . 369
nndist.psp . . . 371
nnfun . . . 373
nnmap . . . 374
nnmark . . . 376
nnwhich . . . 378

nnwhich.pp3 . . . 381
nnwhich.ppx . . . 382
nobjects . . . 384
npoints . . . 385
nsegments . . . 386
nvertices . . . 387
opening . . . 388
overlap.owin . . . 389
owin . . . 390
owin.object . . . 393
owin2mask . . . 394
padimage . . . 397
pairdist . . . 398
pairdist.default . . . 399
pairdist.pp3 . . . 400
pairdist.ppp . . . 402
pairdist.ppx . . . 403
pairdist.psp . . . 404
perimeter . . . 406
periodify . . . 407
persp.im . . . 408
persp.ppp . . . 411
perspPoints . . . 414
pHcolourmap . . . 416
pixelcentres . . . 417
pixellate . . . 418
pixellate.owin . . . 419
pixellate.ppp . . . 420
pixellate.psp . . . 423
pixelquad . . . 424
plot.anylist . . . 426
plot.colourmap . . . 429
plot.hyperframe . . . 431
plot.im . . . 433
plot.imlist . . . 440
plot.layered . . . 442
plot.listof . . . 443
plot.onearrow . . . 447
plot.owin . . . 448
plot.pp3 . . . 452
plot.ppp . . . 453
plot.pppmatching . . . 461
plot.psp . . . 462
plot.quad . . . 465
plot.quadratcount . . . 467
plot.solist . . . 468
plot.splitppp . . . 471
plot.symbolmap . . . 473

plot.tess . . . 475
plot.textstring . . . 477
plot.texturemap . . . 478
plot.yardstick . . . 480
pointsOnLines . . . 482
polarbody . . . 483
polartess . . . 484
pp3 . . . 485
ppp . . . 486
ppp.object . . . 490
pppdist . . . 492
pppmatching . . . 495
pppmatching.object . . . 496
ppx . . . 498
print.im . . . 499
print.owin . . . 500
print.ppp . . . 501
print.psp . . . 502
print.quad . . . 503
progressreport . . . 504
project2segment . . . 506
project2set . . . 507
psp . . . 509
psp.object . . . 510
psp2mask . . . 512
quad.object . . . 513
quadratcount . . . 514
quadrats . . . 517
quadscheme . . . 518
quadscheme.logi . . . 521
quantess . . . 523
quantile.im . . . 525
quantilefun.im . . . 526
quasirandom . . . 527
raster.x . . . 529
rectdistmap . . . 530
reflect . . . 531
regularpolygon . . . 532
relevel.im . . . 533
Replace.im . . . 534
requireversion . . . 536
rescale . . . 537
rescale.im . . . 539
rescale.owin . . . 540
rescale.ppp . . . 541
rescale.psp . . . 542
rescue.rectangle . . . 544
restrict.colourmap . . . 545

rev.colourmap . . . 546
rexplode . . . 547
rgbim . . . 548
ripras . . . 549
rjitter . . . 551
rjitter.pp3 . . . 553
rlinegrid . . . 555
rotate . . . 556
rotate.im . . . 556
rotate.infline . . . 557
rotate.owin . . . 559
rotate.ppp . . . 560
rotate.psp . . . 561
round.ppp . . . 562
rounding.ppp . . . 563
rQuasi . . . 564
rsyst . . . 565
run.simplepanel . . . 566
runifrect . . . 569
scalardilate . . . 570
scaletointerval . . . 571
scanpp . . . 572
selfcrossing.psp . . . 574
selfcut.psp . . . 575
sessionLibs . . . 576
setcov . . . 577
shift . . . 578
shift.im . . . 579
shift.owin . . . 580
shift.ppp . . . 581
shift.ppx . . . 582
shift.psp . . . 584
sidelengths.owin . . . 585
simplepanel . . . 586
simplify.owin . . . 589
solapply . . . 591
solist . . . 592
solutionset . . . 593
spatdim . . . 594
spatstat.options . . . 595
split.hyperframe . . . 600
split.im . . . 601
split.ppp . . . 602
split.ppx . . . 605
spokes . . . 607
square . . . 608
stratrand . . . 609
subset.hyperframe . . . 611

subset.ppp . . . 612
subset.psp . . . 614
summary.anylist . . . 616
summary.distfun . . . 617
summary.im . . . 618
summary.listof . . . 619
summary.owin . . . 620
summary.ppp . . . 621
summary.psp . . . 622
summary.quad . . . 623
summary.solist . . . 624
summary.splitppp . . . 625
superimpose . . . 626
SupportFun . . . 628
symbolmap . . . 630
tess . . . 633
test.crossing.psp . . . 635
text.ppp . . . 636
texturemap . . . 638
textureplot . . . 639
tile.areas . . . 640
tile.centroids . . . 641
tileindex . . . 642
tilenames . . . 643
tiles . . . 645
tiles.empty . . . 646
timed . . . 647
timeTaken . . . 648
transmat . . . 649
triangulate.owin . . . 650
trim.rectangle . . . 651
tweak.colourmap . . . 652
union.quad . . . 653
unique.ppp . . . 654
uniquemap.ppp . . . 655
unitname . . . 656
unmark . . . 658
unstack.ppp . . . 659
unstack.solist . . . 660
update.symbolmap . . . 662
venn.tess . . . 663
vertices . . . 664
volume . . . 665
where.max . . . 666
whichhalfplane . . . 667
Window . . . 668
Window.tess . . . 670
with.hyperframe . . . 671

yardstick 672
zapsmall.im 674

Index **675**

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

Description

The **spatstat.geom** package belongs to the **spatstat** family of packages. It defines classes of geometrical objects such as windows and point patterns, and provides functionality for geometrical operations on them.

Details

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

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

This sub-package **spatstat.geom** defines the main classes of geometrical objects (such as windows, point patterns, line segment patterns, pixel images) and supports geometrical operations (such as shifting and rotating, measuring areas and distances, finding nearest neighbours in a point pattern).

Functions for performing statistical analysis and modelling are in the separate sub-packages **spatstat.explore** and **spatstat.model**.

Functions for linear networks are in the separate sub-package **spatstat.linnet**.

For an overview of all the functions available in the **spatstat** family, see the help file for **spatstat** in the **spatstat** package.

Structure of the spatstat family

The original **spatstat** package grew to be very large, and CRAN requested that the package be divided into several **sub-packages**. Currently the sub-packages are:

- **spatstat.utils** containing basic utilities
- **spatstat.data** containing datasets
- **spatstat.sparse** containing linear algebra utilities
- **spatstat.univar** containing functions for estimating probability distributions of random variables
- **spatstat.geom** containing geometrical objects and geometrical operations
- **spatstat.random** containing code for generating random spatial patterns
- **spatstat.explore** containing the main functionality for exploratory and non-parametric analysis of spatial data
- **spatstat.model** containing the main functionality for statistical modelling and inference for spatial data

- **spatstat.linnet** containing functions for spatial data on a linear network
- **spatstat**, which simply loads the other sub-packages listed above, and provides documentation.

When you install **spatstat**, these sub-packages are also installed. Then if you load the **spatstat** package by typing `library(spatstat)`, the other sub-packages listed above will automatically be loaded or imported. For an overview of all the functions available in these sub-packages, see the help file for **spatstat** in the **spatstat** package,

Additionally there are several **extension packages:**

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

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

OVERVIEW OF CAPABILITIES

Following is an overview of the capabilities of the **spatstat.geom** sub-package.

Types of spatial data:

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

`ppp`	point pattern
`owin`	window (spatial region)
`im`	pixel image
`psp`	line segment pattern
`tess`	tessellation
`pp3`	three-dimensional point pattern
`ppx`	point pattern in any number of dimensions

Additional data types are supported in **spatstat.linnet**.

To create a point pattern:

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

To simulate a random point pattern:

Most of the methods for generating random data are provided in **spatstat.random**. The following basic methods are supplied in **spatstat.geom**:

runifrect	generate n independent uniform random points in a rectangle
rsyst	systematic random sample of points
rjitter	apply random displacements to points in a pattern

Standard point pattern datasets:

Datasets installed in the **spatstat** family are provided in the sub-package `spatstat.data`.

To manipulate a point pattern:

plot.ppp	plot a point pattern (e.g. `plot(X)`)
spatstat.gui::iplot	plot a point pattern interactively
persp.ppp	perspective plot of marked point pattern
edit.ppp	interactive text editor
[.ppp	extract or replace a subset of a point pattern `pp[subset]` or `pp[subwindow]`
subset.ppp	extract subset of point pattern satisfying a condition
superimpose	combine several point patterns
by.ppp	apply a function to sub-patterns of a point pattern
cut.ppp	classify the points in a point pattern
split.ppp	divide pattern into sub-patterns
unmark	remove marks
npoints	count the number of points
coords	extract coordinates, change coordinates
marks	extract marks, change marks or attach marks
rotate	rotate pattern
shift	translate pattern
flipxy	swap x and y coordinates
reflect	reflect in the origin
periodify	make several translated copies
affine	apply affine transformation
scalardilate	apply scalar dilation
nnmark	mark value of nearest data point
identify.ppp	interactively identify points
unique.ppp	remove duplicate points
duplicated.ppp	determine which points are duplicates
uniquemap.ppp	map duplicated points to unique points
connected.ppp	find clumps of points
dirichlet	compute Dirichlet-Voronoi tessellation
delaunay	compute Delaunay triangulation
delaunayDistance	graph distance in Delaunay triangulation
convexhull	compute convex hull
discretise	discretise coordinates
pixellate.ppp	approximate point pattern by pixel image
as.im.ppp	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).

owin	Create a window object owin(xlim, ylim) for rectangular window owin(poly) for polygonal window owin(mask) for binary image window
Window	Extract window of another object
Frame	Extract the containing rectangle ('frame') of another object
as.owin	Convert other data to a window object
square	make a square window
disc	make a circular window
ellipse	make an elliptical window
ripras	Ripley-Rasson estimator of window, given only the points
convexhull	compute convex hull of something
letterR	polygonal window in the shape of the R logo
clickpoly	interactively draw a polygonal window
clickbox	interactively draw a rectangle

To manipulate a window:

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

Digital approximations:

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

raster.y	raster y coordinates
raster.xy	raster x and y coordinates
as.polygonal	convert pixel mask to polygonal window

See spatstat.options to control the approximation

Geometrical computations with windows:

edges	extract boundary edges
intersect.owin	intersection of two windows
union.owin	union of two windows
setminus.owin	set subtraction of two windows
inside.owin	determine whether a point is inside a window
area.owin	compute area
perimeter	compute perimeter length
diameter.owin	compute diameter
incircle	find largest circle inside a window
inradius	radius of incircle
connected.owin	find connected components of window
eroded.areas	compute areas of eroded windows
dilated.areas	compute areas of dilated windows
bdist.points	compute distances from data points to window boundary
bdist.pixels	compute distances from all pixels to window boundary
bdist.tiles	boundary distance for each tile in tessellation
distmap.owin	distance transform image
distfun.owin	distance transform
centroid.owin	compute centroid (centre of mass) of window
is.subset.owin	determine whether one window contains another
is.convex	determine whether a window is convex
convexhull	compute convex hull
triangulate.owin	decompose into triangles
as.mask	pixel approximation of window
as.polygonal	polygonal approximation of window
is.rectangle	test whether window is a rectangle
is.polygonal	test whether window is polygonal
is.mask	test whether window is a mask
setcov	spatial covariance function of window
pixelcentres	extract centres of pixels in mask
clickdist	measure distance between two points clicked by user

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.

im	create a pixel image
as.im	convert other data to a pixel image
pixellate	convert other data to a pixel image
as.matrix.im	convert pixel image to matrix
as.data.frame.im	convert pixel image to data frame

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

Line segment patterns

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

psp	create a line segment pattern
as.psp	convert other data into a line segment pattern
edges	extract edges of a window
is.psp	determine whether a dataset has class "psp"
plot.psp	plot a line segment pattern
print.psp	print basic information
summary.psp	print summary information
[.psp	extract a subset of a line segment pattern
subset.psp	extract subset of line segment pattern

as.data.frame.psp	convert line segment pattern to data frame
marks.psp	extract marks of line segments
marks<-.psp	assign new marks to line segments
unmark.psp	delete marks from line segments
midpoints.psp	compute the midpoints of line segments
endpoints.psp	extract the endpoints of line segments
lengths_psp	compute the lengths of line segments
angles.psp	compute the orientation angles of line segments
superimpose	combine several line segment patterns
flipxy	swap x and y coordinates
rotate.psp	rotate a line segment pattern
shift.psp	shift a line segment pattern
periodify	make several shifted copies
affine.psp	apply an affine transformation
pixellate.psp	approximate line segment pattern by pixel image
as.mask.psp	approximate line segment pattern by binary mask
distmap.psp	compute the distance map of a line segment pattern
distfun.psp	compute the distance map of a line segment pattern
selfcrossing.psp	find crossing points between line segments
selfcut.psp	cut segments where they cross
crossing.psp	find crossing points between two line segment patterns
extrapolate.psp	extrapolate line segments to infinite lines
nncross	find distance to nearest line segment from a given point
nearestsegment	find line segment closest to a given point
project2segment	find location along a line segment closest to a given point
pointsOnLines	generate points evenly spaced along line segment
rlinegrid	generate a random array of parallel lines through a window

Tessellations

An object of class "tess" represents a tessellation.

tess	create a tessellation
quadrats	create a tessellation of rectangles
hextess	create a tessellation of hexagons
polartess	tessellation using polar coordinates
quantess	quantile tessellation
venn.tess	Venn diagram tessellation
dirichlet	compute Dirichlet-Voronoi tessellation of points
delaunay	compute Delaunay triangulation of points
as.tess	convert other data to a tessellation
plot.tess	plot a tessellation
tiles	extract all the tiles of a tessellation
[.tess	extract some tiles of a tessellation
[<-.tess	change some tiles of a tessellation
intersect.tess	intersect two tessellations or restrict a tessellation to a window
chop.tess	subdivide a tessellation by a line
tile.areas	area of each tile in tessellation

bdist.tiles	boundary distance for each tile in tessellation
connected.tess	find connected components of tiles
shift.tess	shift a tessellation
rotate.tess	rotate a tessellation
reflect.tess	reflect about the origin
flipxy.tess	reflect about the diagonal
affine.tess	apply affine transformation

Functions which are constant on each tile of a tessellation:

as.function.tess	convert tessellation to function
plot.tessfun	plot the function
integral.tessfun	integrate the function
as.tess.tessfun	retrieve the original 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".

pp3	create a 3-D point pattern
plot.pp3	plot a 3-D point pattern
coords	extract coordinates
as.hyperframe	extract coordinates
subset.pp3	extract subset of 3-D point pattern
unitname.pp3	name of unit of length
npoints	count the number of points
box3	create a 3-D rectangular box
as.box3	convert data to 3-D rectangular box
unitname.box3	name of unit of length
diameter.box3	diameter of box
volume.box3	volume of box
shortside.box3	shortest side of box
eroded.volumes	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.

ppx	create a multidimensional space-time point pattern
coords	extract coordinates
as.hyperframe	extract coordinates
subset.ppx	extract subset
unitname.ppx	name of unit of length
npoints	count the number of points
boxx	define multidimensional box
diameter.boxx	diameter of box
volume.boxx	volume of box
shortside.boxx	shortest side of box

eroded.volumes.boxx	volumes of erosions of box

Linear networks

An object of class "linnet" represents a linear network (for example, a road network). This is supported in the sub-package **spatstat.linnet**.

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

Hyperframes

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

hyperframe	create a hyperframe
as.hyperframe	convert data to hyperframe
plot.hyperframe	plot hyperframe
with.hyperframe	evaluate expression using each row of hyperframe
cbind.hyperframe	combine hyperframes by columns
rbind.hyperframe	combine hyperframes by rows
as.data.frame.hyperframe	convert hyperframe to data frame
subset.hyperframe	method for subset
head.hyperframe	first few rows of hyperframe
tail.hyperframe	last few rows of hyperframe

Layered objects

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

layered	create layered object
plot.layered	plot layered object
[.layered	extract subset of layered object

Colour maps

A colour map is a mechanism for associating colours with data. It can be regarded as a function, mapping data to colours. Using a colourmap object in a plot command ensures that the mapping from numbers to colours is the same in different plots.

colourmap	create a colour map
plot.colourmap	plot the colour map only
tweak.colourmap	alter individual colour values
interp.colourmap	make a smooth transition between colours
beachcolourmap	one special colour map

Inspection of data:

summary(X)	print useful summary of point pattern X
X	print basic description of point pattern X
any(duplicated(X))	check for duplicated points in pattern X

intensity	Mean intensity
quadratcount	Quadrat counts

Distances in a point pattern:

nndist	nearest neighbour distances
nnwhich	find nearest neighbours
pairdist	distances between all pairs of points
crossdist	distances between points in two patterns
nncross	nearest neighbours between two point patterns
exactdt	distance from any location to nearest data point
distmap	distance map image
distfun	distance map function
nnmap	nearest point image
nnfun	nearest point function

Programming tools:

applynbd	apply function to every neighbourhood in a point pattern
markstat	apply function to the marks of neighbours in a point pattern
pppdist	find the optimal match between two point patterns

Distances in a three-dimensional point pattern:

pairdist.pp3	distances between all pairs of points
crossdist.pp3	distances between points in two patterns
nndist.pp3	nearest neighbour distances
nnwhich.pp3	find nearest neighbours
nncross.pp3	find nearest neighbours in another pattern

Distances in multi-dimensional point pattern:

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

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

Licence

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

Acknowledgements

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

Additional contributions and suggestions from Monsuru Adepeju, Corey Anderson, Ang Qi Wei, Ryan Arellano, Jens Åström, Robert Aue, Marcel Austenfeld, Sandro Azaele, Guy Bayegnak, Colin Beale, Melanie Bell, Thomas Bendtsen, Ricardo Bernhardt, Andrew Bevan, Brad Biggerstaff, Anders Bilgrau, Leanne Bischof, Christophe Biscio, Roger Bivand, Jose M. Blanco Moreno, Florent Bonneu, Jordan Brown, Ian Buller, Julian Burgos, Simon Byers, Ya-Mei Chang, Jianbao Chen, Igor Chernayavsky, Y.C. Chin, Bjarke Christensen, Lucía Cobo Sanchez, Jean-Francois Coeurjolly, Kim Colyvas, Hadrien Commenges, Rochelle Constantine, Robin Corria Ainslie, Richard Cotton, Marcelino de la Cruz, Peter Dalgaard, Mario D'Antuono, Sourav Das, Peter Diggle, Patrick Donnelly, Ian Dryden, Stephen Eglen, Ahmed El-Gabbas, Belarmain Fandohan, Olivier Flores, David Ford, Peter Forbes, Shane Frank, Janet Franklin, Funwi-Gabga Neba, Oscar Garcia, Agnes Gault, Jonas Geldmann, Marc Genton, Shaaban Ghalandarayeshi, Julian Gilbey, Jason Goldstick, Pavel Grabarnik, C. Graf, Ute Hahn, Andrew Hardegen, Martin Bøgsted Hansen, Martin Hazelton, Juha Heikkinen, Mandy Hering, Markus Herrmann, Maximilian Hesselbarth, Paul Hewson, Hamidreza Heydarian, Kassel Hingee, Kurt Hornik, Philipp Hunziker, Jack Hywood, Ross Ihaka, Čenk Içös, Aruna Jammalamadaka, Robert John-Chandran, Devin Johnson, Mahdieh Khanmohammadi, Bob Klaver, Lily Kozmian-Ledward, Peter Kovesi, Mike Kuhn, Jeff Laake, Robert Lamb, Frédéric Lavancier, Tom Lawrence, Tomas Lazauskas, Jonathan Lee, George Leser, Angela Li, Li Haitao, George Limitsios, Andrew Lister, Nestor Luambua, Ben Madin, Martin Maechler, Kiran Marchikanti, Jeff Marcus, Robert Mark, Peter McCullagh, Monia Mahling, Jorge Mateu Mahiques, Ulf Mehlig, Frederico Mestre, Sebastian Wastl Meyer, Mi Xiangcheng, Lore De Middeleer, Robin Milne, Enrique Miranda, Jesper Møller, Annie Mollié, Ines Moncada, Mehdi Moradi, Virginia Morera Pujol, Erika Mudrak, Gopalan Nair, Nader Najari, Nicoletta Nava, Linda Stougaard Nielsen, Felipe Nunes, Jens Randel Nyengaard, Jens Oehlschlägel, Thierry Onkelinx, Sean O'Riordan, Evgeni Parilov, Jeff Picka, Nicolas Picard, Tim Pollington, Mike Porter, Sergiy Protsiv, Adrian Raftery, Suman Rakshit, Ben Ramage, Pablo Ramon, Xavier Raynaud, Nicholas Read, Matt Reiter, Ian Renner, Tom Richardson, Brian Ripley, Ted Rosenbaum, Barry Rowlingson, Jason Rudokas, Tyler Rudolph, John Rudge, Christopher Ryan, Farzaneh Safavimanesh, Aila Särkkä, Cody Schank, Katja Schladitz, Sebastian Schutte, Bryan Scott, Olivia Semboli, François Sémécurbe, Vadim Shcherbakov, Shen Guochun, Shi Peijian, Harold-Jeffrey Ship, Tammy L Silva, Ida-Maria Sintorn, Yong Song, Malte Spiess, Mark Stevenson, Kaspar Stucki, Jan Sulavik, Michael Sumner, P. Surovy, Ben Taylor, Thordis Linda Thorarinsdottir, Leigh Torres, Berwin Turlach, Torben Tvedebrink, Kevin Ummer, Medha Uppala, Malissa Usher, Andrew van Burgel, Tobias Verbeke, Mikko Vihtakari, Alexendre Villers, Fabrice Vinatier, Maximilian Vogtland, Sasha Voss, Sven Wagner, Hao Wang, H. Wendrock, Jan Wild, Carl G. Witthoft, Selene Wong, Maxime Woringer, Luke Yates, Mike Zamboni and Achim Zeileis.

Author(s)

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

`add.texture` *Fill Plot With Texture*

Description

Draws a simple texture inside a region on the plot.

Usage

```
add.texture(W, texture = 4, spacing = NULL, ...)
```

Arguments

`W`	Window (object of class `"owin"`) inside which the texture should be drawn.
`texture`	Integer from 1 to 8 identifying the type of texture. See Details.
`spacing`	Spacing between elements of the texture, in units of the current plot.
`...`	Further arguments controlling the plot colour, line width etc.

Details

The chosen texture, confined to the window `W`, will be added to the current plot. The available textures are:

texture=1: Small crosses arranged in a square grid.

texture=2: Parallel vertical lines.

texture=3: Parallel horizontal lines.

texture=4: Parallel diagonal lines at 45 degrees from the horizontal.

texture=5: Parallel diagonal lines at 135 degrees from the horizontal.

texture=6: Grid of horizontal and vertical lines.

texture=7: Grid of diagonal lines at 45 and 135 degrees from the horizontal.

texture=8: Grid of hexagons.

Author(s)

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

See Also

`owin`, `plot.owin`, `textureplot`, `texturemap`.

Examples

```
W <- Window(chorley)
plot(W, main="")
add.texture(W, 7)
```

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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

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

and Rolf Turner <rolfturner@posteo.net>

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)
if(interactive()) plot(X)
affine(letterR, shear, c(0, 0.5))
affine(as.mask(letterR), shear, c(0, 0.5))
```

`affine.ppp` *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 'ppp'
affine(X, mat=diag(c(1,1)), vec=c(0,0), ...)
```

Arguments

`X`	Point pattern (object of class `"ppp"`).
`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 `"ppp"`) representing the result of applying the affine transformation.

Author(s)

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

See Also

`affine`, `affine.owin`, `affine.psp`, `affine.im`, `flipxy`, `rotate`, `shift`

Examples

```
# shear transformation
X <- affine(cells, matrix(c(1,0,0.6,1),ncol=2))
if(interactive()) {
  plot(X)
  # rescale y coordinates by factor 1.3
  plot(affine(cells, diag(c(1,1.3))))
}
```

`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@curtin.edu.au>`

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

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)))
```

---

affine.tess *Apply Geometrical Transformation To Tessellation*

Description

Apply various geometrical transformations of the plane to each tile in a tessellation.

Usage

```
  ## S3 method for class 'tess'
reflect(X)

  ## S3 method for class 'tess'
flipxy(X)

  ## S3 method for class 'tess'
shift(X, ...)

  ## S3 method for class 'tess'
rotate(X, angle=pi/2, ..., centre=NULL)

  ## S3 method for class 'tess'
scalardilate(X, f, ...)

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

Arguments

X	Tessellation (object of class "tess").
angle	Rotation angle in radians (positive values represent anticlockwise rotations).

mat	Matrix representing a linear transformation.
vec	Vector of length 2 representing a translation.
f	Positive number giving scale factor.
...	Arguments passed to other methods.
centre	Centre of rotation. Either a vector of length 2, or a character string (partially matched to "centroid", "midpoint" or "bottomleft"). The default is the coordinate origin c(0,0).

Details

These are method for the generic functions reflect, flipxy, shift, rotate, scalardilate, affine for tessellations (objects of class "tess").

The individual tiles of the tessellation, and the window containing the tessellation, are all subjected to the same geometrical transformation.

The transformations are performed by the corresponding method for windows (class "owin") or images (class "im") depending on the type of tessellation.

If the argument origin is used in shift.tess it is interpreted as applying to the window containing the tessellation. Then all tiles are shifted by the same vector.

Value

Another tessellation (of class "tess") representing the result of applying the geometrical transformation.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

Generic functions reflect, shift, rotate, scalardilate, affine.

Methods for windows: reflect.default, shift.owin, rotate.owin, scalardilate.owin, affine.owin.

Methods for images: reflect.im, shift.im, rotate.im, scalardilate.im, affine.im.

Examples

```
live <- interactive()
if(live) {
  H <- hextess(letterR, 0.2)
  plot(H)
  plot(reflect(H))
  plot(rotate(H, pi/3))
} else H <- hextess(letterR, 0.6)

# shear transformation
shear <- matrix(c(1,0,0.6,1),2,2)
```

```
sH <- affine(H, shear)
if(live) plot(sH)
```

`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 `(x0,y0)` to `(x1,y1)`. The values returned are angles in the full range from $-\pi$ to π. The angle is computed as `atan2(y1-y0,x1-x0)`. 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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

`psp`, `marks.psp`, `summary.psp`, `midpoints.psp`, `lengths_psp`, `endpoints.psp`, `extrapolate.psp`.

Examples

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

anylist *List of Objects*

Description

Make a list of objects of any type.

Usage

```
anylist(...)
as.anylist(x)
```

Arguments

...	Any number of arguments of any type.
x	A list.

Details

An object of class "anylist" is a list of objects that the user intends to treat in a similar fashion.

For example it may be desired to plot each of the objects side-by-side: this can be done using the function plot.anylist.

The objects can belong to any class; they may or may not all belong to the same class.

In the **spatstat** package, various functions produce an object of class "anylist".

Value

A list, belonging to the class "anylist", containing the original objects.

Author(s)

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

Rolf Turner <rolfturner@posteo.net>

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

See Also

solist, as.solist, anylapply.

Examples

```
if(require(spatstat.explore)) {
  anylist(cells, intensity(cells), Kest(cells))
} else {
  anylist(cells, intensity(cells))
}
anylist()
```

anyNA.im *Check Whether Image Contains NA Values*

Description

Checks whether any pixel values in a pixel image are NA (meaning that the pixel lies outside the domain of definition of the image).

Usage

```
  ## S3 method for class 'im'
anyNA(x, recursive = FALSE)
```

Arguments

x	A pixel image (object of class "im").
recursive	Ignored.

Details

The function anyNA is generic: anyNA(x) is a faster alternative to any(is.na(x)).

This function anyNA.im is a method for the generic anyNA defined for pixel images. It returns the value TRUE if any of the pixel values in x are NA, and and otherwise returns FALSE.

Value

A single logical value.

Author(s)

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

See Also

im.object

Examples

```
anyNA(as.im(letterR))
```

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. It 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).

If one of the arguments is NULL, it will be ignored and the other argument will be returned.

Value

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

Author(s)

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

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=NULL, R=NULL, criterion=NULL, 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.
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.
criterion	Function. If this argument is present, the neighbourhood of a point of X is determined by evaluating this function. See under **Details**.
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. 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 exclude=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 dist[i] will be the distance of a point from the current point, and 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.

- If more than one of the arguments `N`, `R` and `criterion` is given, the neighbourhood is defined as the *intersection* of the neighbourhoods specified by these arguments. For example if `N=3` and `R=5` then the neighbourhood is formed by finding the 3 nearest neighbours of current point, and retaining only those neighbours which lie closer than 5 units from the current point.

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:

  `FUN(Y=Y, current=current, dists=dists, dranks=dranks, ...)`

  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 `...` 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 list of length 2 with entries `current$x` and `current$y` containing the coordinates of the current point. If `X` is marked, then `current` is a point pattern containing exactly one point, so that `current$x` is its x-coordinate and `current$marks` is its mark value. In either case, the coordinates of the current point can be referred to as `current$x` and `current$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 `npoints(X)`, 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 `npoints(X)`.

Author(s)

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

See Also

ppp.object, apply, markstat, marktable

Examples

```
 redwood
 # count the number of points within radius 0.2 of each point of X
 nneighbours <- applynbd(redwood, R=0.2, function(Y, ...){npoints(Y)-1})
 # equivalent to:
 nneighbours <- applynbd(redwood, R=0.2, function(Y, ...){npoints(Y)}, 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
 trees <- longleaf

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


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

 if(interactive()) {
 showoffK <- function(Y, current, dists, dranks, fullpicture,rad) {
plot(fullpicture, main="")
points(Y, cex=2)
       ux <- current[["x"]]
       uy <- current[["y"]]
points(ux, uy, pch="+",cex=3)
theta <- seq(0,2*pi,length=100)
polygon(ux + rad * cos(theta), uy+rad*sin(theta))
text(ux + rad/3, uy + rad/2,npoints(Y),cex=3)
if(interactive()) Sys.sleep(if(runif(1) < 0.1) 1.5 else 0.3)
return(npoints(Y))
 }
 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)
if(interactive()) Sys.sleep(if(runif(1) < 0.1) 1.5 else 0.3)
return(nnd)
 }

 applynbd(cells, N=1, showoffG, exclude=TRUE, fullpicture=cells)
 }
```

`area.owin` *Area of a Window*

Description

Computes the area of a window

Usage

```
 area(w)

 ## S3 method for class 'owin'
area(w)

 ## Default S3 method:
area(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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

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

Examples

```
w <- unit.square()
area(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(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(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.

...	Arguments passed to distmap to determine the pixel resolution, when exact=FALSE.
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@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

AreaInter, areaLoss

Examples

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

---

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[subset]) and one column for each value in r.

Author(s)

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

See Also

AreaInter, areaGain, dilated.areas

Examples

```
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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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.boxx *Convert Data to Multi-Dimensional Box*

Description

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

Usage

```
as.boxx(..., warn.owin = TRUE)
```

Arguments

...	Data that can be interpreted as giving the dimensions of a multi-dimensional box. See Details.
warn.owin	Logical value indicating whether to print a warning if a non-rectangular window (object of class "owin") is supplied.

Details

Either a single argument should be provided which is one of the following:

- an object of class "boxx"
- an object of class "box3"
- an object of class "owin"
- a numeric vector of even length, specifying the corners of the box. See Examples

or a list of arguments acceptable to boxx.

Value

A "boxx" object.

Author(s)

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

Rolf Turner <rolfturner@posteo.net>

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

Examples

```
# Convert unit square to two dimensional box.
W <- owin()
as.boxx(W)
# Make three dimensional box [0,1]x[0,1]x[0,1] from numeric vector
as.boxx(c(0,1,0,1,0,1))
```

`as.colourmap` *Convert to Colour Map*

Description

Convert some other kind of data to a colour map.

Usage

```
  as.colourmap(x, ...)

  ## S3 method for class 'colourmap'
as.colourmap(x, ...)

  ## S3 method for class 'symbolmap'
as.colourmap(x, ..., warn=TRUE)
```

Arguments

`x`	Data to be converted to a colour map. An object of class `"symbolmap"`, `"colourmap"` or some other kind of suitable data.
`...`	Other arguments passed to methods.
`warn`	Logical value specifying whether to issue a warning if `x` does not contain any colour map information.

Details

If `x` contains colour map information, it will be extracted and returned as a colour map object. Otherwise, `NULL` will be returned (and a warning will be issued if `warn=TRUE`, the default).

Value

A colour map (object of class `"colourmap"`) or `NULL`.

Author(s)

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

See Also

`colourmap`

Examples

```
m <- pHcolourmap(c(3,8))
g <- symbolmap(pch=21, bg=m, size=function(x){ 1.1 * x }, range=c(3,8))
opa <- par(mfrow=c(1,2))
plot(g, vertical=TRUE)
plot(as.colourmap(g), vertical=TRUE)
par(opa)
```

---

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

x	Hyperframe (object of class "hyperframe").
row.names	Optional character vector of row names.
optional	Argument passed to as.data.frame controlling what happens to row names.
...	Ignored.
discard	Logical. Whether to discard columns of the hyperframe that do not contain atomic data. See Details.
warn	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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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.

The data frame entries are automatically sorted in increasing order of the x coordinate (and in increasing order of y within x).

Value

A data frame.

Author(s)

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

Examples

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

Y <- as.data.frame(Z)

head(Y)
```

`as.data.frame.owin` *Convert Window to Data Frame*

Description

Converts a window object to a data frame.

Usage

```
## S3 method for class 'owin'
as.data.frame(x, ..., drop=TRUE)
```

Arguments

`x`	Window (object of class `"owin"`).
`...`	Further arguments passed to `as.data.frame.default` to determine the row names and other features.
`drop`	Logical value indicating whether to discard pixels that are outside the window, when `x` is a binary mask.

Details

This function returns a data frame specifying the coordinates of the window.

If `x` is a binary mask window, the result is a data frame with columns `x` and `y` containing the spatial coordinates of each *pixel*. If `drop=TRUE` (the default), only pixels inside the window are retained. If `drop=FALSE`, all pixels are retained, and the data frame has an extra column `inside` containing the logical value of each pixel (`TRUE` for pixels inside the window, `FALSE` for outside).

If `x` is a rectangle or a polygonal window, the result is a data frame with columns `x` and `y` containing the spatial coordinates of the *vertices* of the window. If the boundary consists of several polygons, the data frame has additional columns `id`, identifying which polygon is being traced, and `sign`, indicating whether the polygon is an outer or inner boundary (`sign=1` and `sign=-1` respectively).

Value

A data frame with columns named `x` and `y`, and possibly other columns.

Author(s)

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

See Also

as.data.frame.im, as.owin.data.frame

Examples

```
as.data.frame(square(1))

holey <- owin(poly=list(
                    list(x=c(0,10,0), y=c(0,0,10)),
                    list(x=c(2,2,4,4), y=c(2,4,4,2))))
as.data.frame(holey)

M <- as.mask(holey, eps=0.5)
Mdf <- as.data.frame(M)
```

---

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

Examples

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

Examples

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

`as.data.frame.tess` *Convert Tessellation to Data Frame*

Description

Converts a spatial tessellation object to a data frame.

Usage

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

Arguments

`x`	Tessellation (object of class `"tess"`).
`...`	Further arguments passed to `as.data.frame.owin` or `as.data.frame.im` and ultimately to `as.data.frame.default` to determine the row names and other features.

Details

This function converts the tessellation `x` to a data frame.

If `x` is a pixel image tessellation (a pixel image with factor values specifying the tile membership of each pixel) then this pixel image is converted to a data frame by `as.data.frame.im`. The result is a data frame with columns `x` and `y` giving the pixel coordinates, and `Tile` identifying the tile containing the pixel.

If `x` is a tessellation consisting of a rectangular grid of tiles or a list of polygonal tiles, then each tile is converted to a data frame by `as.data.frame.owin`, and these data frames are joined together, yielding a single large data frame containing columns `x`, `y` giving the coordinates of vertices of the polygons, and `Tile` identifying the tile.

Value

A data frame with columns named `x`, `y`, `Tile`, and possibly other columns.

Author(s)

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

See Also

`as.data.frame.owin`, `as.data.frame.im`

Examples

```
Z <- as.data.frame(dirichlet(cells))
head(Z, 10)
```

`as.function.im` *Convert Pixel Image to Function of Coordinates*

Description

Converts a pixel image to a function of the x and y coordinates.

Usage

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

Arguments

`x`	Pixel image (object of class `"im"`).
`...`	Ignored.

Details

This command converts a pixel image (object of class `"im"`) to a `function(x,y)` where the arguments `x` and `y` are (vectors of) spatial coordinates. This function returns the pixel values at the specified locations.

Value

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

Author(s)

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

Rolf Turner <rolfturner@posteo.net>

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

See Also

`[.im`

Examples

```
d <- setcov(square(1))
f <- as.function(d)
f(0.1, 0.3)
```

`as.function.owin` *Convert Window to Indicator Function*

Description

Converts a spatial window to a function of the x and y coordinates returning the value 1 inside the window and 0 outside.

Usage

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

Arguments

`x`	Pixel image (object of class `"owin"`).
`...`	Ignored.

Details

This command converts a spatial window (object of class `"owin"`) to a `function(x,y)` where the arguments `x` and `y` are (vectors of) spatial coordinates. This is the indicator function of the window: it returns the value 1 for locations inside the window, and returns 0 for values outside the window.

Value

A function in the R language with arguments `x,y`. It also belongs to the class `"indicfun"` which has methods for `plot` and `print`.

Author(s)

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

See Also

`as.im.owin`

Examples

```
W <- Window(humberside)
f <- as.function(W)
f
f(5000, 4500)
f(123456, 78910)
X <- runifrect(5, Frame(humberside))
f(X)
plot(f)
```

as.function.tess *Convert a Tessellation to a Function*

Description

Convert a tessellation into a function of the x and y coordinates. The default function values are factor levels specifying which tile of the tessellation contains the point (x, y).

Usage

```
  ## S3 method for class 'tess'
as.function(x,...,values=NULL)
```

Arguments

x	A tessellation (object of class "tess").
values	Optional. A vector giving the values of the function for each tile of x.
...	Ignored.

Details

This command converts a tessellation (object of class "tess") to a function(x,y) where the arguments x and y are (vectors of) spatial coordinates. The corresponding function values are factor levels identifying which tile of the tessellation contains each point. Values are NA if the corresponding point lies outside the tessellation.

If the argument values is given, then it determines the value of the function in each tile of x.

Value

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

The class "tessfun" has methods for plot, print, as.tess and integral.

Author(s)

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

See Also

integral.tessfun for integration of the function.

tileindex for the low-level calculation of tile index.

cut.ppp and split.ppp to divide up the points of a point pattern according to a tessellation.

Examples

```
X <- runifrect(7)
V <- dirichlet(X)
f <- as.function(V)
f(0.1, 0.4)
plot(f)
```

`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, ...)

## S3 method for class 'anylist'
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`, `listof`, `anylist` and a default method, all of which convert data that is like a hyperframe into a hyperframe object. (The method for the class `listof` and `anylist` 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@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>.

See Also

`hyperframe`, `as.hyperframe.ppx`

Examples

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

sims <- replicate(3, runifrect(10), simplify=FALSE)
as.hyperframe(as.listof(sims))
as.hyperframe(as.solist(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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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)
```

```
# ppx with marks which are point patterns
val <- runif(4, max=10)
num <- sapply(val, rpois, n=1)
E <- lapply(num, runifrect)
hf <- hyperframe(t=val, e=as.listof(E))
Z <- ppx(data=hf, domain=c(0,10))

# convert ppx to a hyperframe
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,
        rule.eps=c("adjust.eps", "grow.frame", "shrink.frame"),
        rule.pix=NULL,
        na.replace=NULL)

  ## S3 method for class 'owin'
as.im(X, W=NULL, ...,
        eps=NULL, dimyx=NULL, xy=NULL,
        rule.eps=c("adjust.eps", "grow.frame", "shrink.frame"),
        rule.pix=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,
        rule.eps=c("adjust.eps", "grow.frame", "shrink.frame"),
        rule.tile = c("sample", "majority"),
        na.replace=NULL, values=NULL)

  ## S3 method for class 'function'
as.im(X, W=NULL, ...,
```

```
        eps=NULL, dimyx=NULL, xy=NULL,
        rule.eps=c("adjust.eps", "grow.frame", "shrink.frame"),
        rule.pix=NULL,
        na.replace=NULL,
        stringsAsFactors=NULL,
        strict=FALSE, drop=TRUE)

  ## S3 method for class 'funxy'
as.im(X, W=Window(X), ...)

  ## S3 method for class 'expression'
as.im(X, W=NULL, ...)

  ## S3 method for class 'distfun'
as.im(X, W=NULL, ...,
        eps=NULL, dimyx=NULL, xy=NULL,
        rule.eps=c("adjust.eps", "grow.frame", "shrink.frame"),
        rule.pix=NULL,
        na.replace=NULL, approx=TRUE)

  ## S3 method for class 'nnfun'
as.im(X, W=NULL, ...,
        eps=NULL, dimyx=NULL, xy=NULL,
        rule.eps=c("adjust.eps", "grow.frame", "shrink.frame"),
        rule.pix=NULL,
        na.replace=NULL, approx=TRUE)

  ## S3 method for class 'data.frame'
as.im(X, ..., step, fatal=TRUE, drop=TRUE)

  ## Default S3 method:
as.im(X, W=NULL, ...,
        eps=NULL, dimyx=NULL, xy=NULL,
        rule.eps=c("adjust.eps", "grow.frame", "shrink.frame"),
        rule.pix=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, rule.eps	Optional parameters passed to as.mask which determine the pixel array geometry. See as.mask.

rule.pix	Optional parameter passed to owin2mask specifying how boundary pixels should be treated when converting a polygonal window to a pixel mask.
rule.tile	String (partially matched) specifying how to determine, for each pixel, which tile of the tessellation X covers the pixel. If rule.tile="sample" (the default), the tile covering the centre of the pixel is chosen. If rule.tile="majority", the tile occupying the largest fraction of area in the pixel is chosen.
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. A single atomic value (numeric, integer, logical etc).
values	Optional. Vector of values to be assigned to each tile of the tessellation, when X is a tessellation. An atomic vector (numeric, integer, logical etc.)
strict	Logical value indicating whether to match formal arguments of X when X is a function. If strict=FALSE (the default), all the ... arguments are passed to X. If strict=TRUE, only named arguments are passed, and only if they match the names of formal arguments of X.
step	Optional. A single number, or numeric vector of length 2, giving the grid step lengths in the x and y directions.
fatal	Logical value indicating what to do if the resulting image would be too large for available memory. If fatal=TRUE (the default), an error occurs. If fatal=FALSE, a warning is issued and NULL is returned.
drop	Logical value indicating what to do if the result would normally be a list of pixel images but the list contains only one image. If drop=TRUE (the default), the pixel image is extracted and the result is a pixel image. If drop=FALSE, this list is returned as the result.
stringsAsFactors	Logical value (passed to data.frame) specifying how to handle pixel values which are character strings. If TRUE, character values are interpreted as factor levels. If FALSE, they remain as character strings. The default depends on the version of R. See section *Handling Character Strings*.
approx	Logical value indicating whether to compute an approximate result at faster speed.

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"). By default, 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. If argument values is given, the result is a pixel image in which every pixel inside the i-th tile of the tessellation has pixel value equal to values[i].

- 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 "funxy" representing a function(x,y,...) defined in a spatial region. The function will be evaluated as described above. The window W defaults to the domain of definition of the function.
- an object of class "funxy" which also belongs to one of the following special classes. If approx=TRUE (the default), the function will be evaluated approximately using a very fast algorithm. If approx=FALSE, the function will be evaluated exactly at each grid location as described above.
  - an object of class "distfun" representing a distance function (created by the command distfun). The fast approximation is the distance transform distmap.
  - an object of class "nnfun" representing a nearest neighbour function (created by the command nnfun). The fast approximation is nnmap.
  - an object of class "densityfun" representing a kernel estimate of intensity (created by the command densityfun). The fast approximation is the Fast Fourier Transform algorithm in density.ppp.
  - an object of class "Smoothfun" representing kernel-smoothed values (created by the command Smoothfun). The fast approximation is the Fast Fourier Transform algorithm in Smooth.ppp.
- An expression involving the variables x and y representing the spatial coordinates, and possibly also involving other variables. The additional argument W must be present; it will be converted to a binary image mask. The expression X will be evaluated in an environment where x and y are **vectors** containing the spatial coordinates of all the pixels in the image mask. Evaluation of the expression X must yield a vector or factor, of the same length as x and y, giving the pixel values.
- 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.
- A data frame with at least three columns. Columns named x, y and z, if present, will be assumed to contain the spatial coordinates and the pixel values, respectively. Otherwise the x and y coordinates will be taken from the first two columns of the data frame, and any remaining columns will be interpreted as pixel values.

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 everwhere 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 <- as.im(X) will repair the internal format of X so that it conforms to the current version of **spatstat**.

If X is a data frame with m columns, then m-2 columns of data are interpreted as pixel values, yielding m-2 pixel images. The result of as.im.data.frame is a list of pixel images, belonging to the class "imlist". If m = 3 and drop=TRUE (the default), then the result is a pixel image rather than a list containing this image.

If X is a function(x,y) which returns a matrix of values, then as.im(X, W) will be a list of pixel images.

Value

A pixel image (object of class "im"), or a list of pixel images, or NULL if the conversion failed.

Boundary pixels

When W is a rectangle or a polygonal window, the default behaviour is to represent the window as the set of pixels whose *centres* fall inside the polygon. A pixel which contains part of the interior of the polygon, but whose centre falls outside the polygon, is excluded by default, for the sake of efficiency.

To ensure that all pixels which intersect the interior of the polygon are included in the pixel image, use the argument rule.pix="cover" which will be passed to owin2mask.

Character-valued images

By default, if the pixel value data are character strings, they will be treated as levels of a factor, and the resulting image will be factor-valued. To prevent the conversion of character strings to factors, use the argument stringsAsFactors=FALSE, which is recognised by most of the methods for as.im, or alternatively set options(stringsAsFactors=FALSE).

Handling Character Strings

The argument stringsAsFactors is a logical value (passed to data.frame) specifying how to handle pixel values which are character strings. If TRUE, character values are interpreted as factor levels. If FALSE, they remain as character strings. The default values of stringsAsFactors depends on the version of R.

- In R versions < 4.1.0 the factory-fresh default is stringsAsFactors=FALSE and the default can be changed by setting options(stringsAsFactors=FALSE).
- In R versions >= 4.1.0 the default is stringsAsFactors=FALSE and there is no option to change the default.

Author(s)

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

See Also

Separate documentation for as.im.ppp

Examples

```
# window object
W <- Window(demopat)
plot(W)
Z <- as.im(W)
image(Z)
# function
Z <- as.im(function(x,y) {x^2 + y^2}, unit.square())
image(Z)
# or as an expression
Z <- as.im(expression(x^2+y^2), square(1))

# 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
Z <- as.im(f, letterR, x0=2.5, y0=2)
image(Z)
# usual convention in R
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)

#' distance functions
d <- distfun(redwood)
Zapprox <- as.im(d)
Zexact <- as.im(d, approx=FALSE)
plot(solist(approx=Zapprox, exact=Zexact), main="")

# pixellate the Dirichlet tessellation
Di <- dirichlet(redwood)
plot(as.im(Di))
```

```
plot(Di, add=TRUE, border="white")

# as.im.data.frame is the reverse of as.data.frame.im
grad <- bei.extra$grad
slopedata <- as.data.frame(grad)
slope <- as.im(slopedata)
unitname(grad) <- unitname(slope) <- unitname(grad) # for compatibility
all.equal(slope, grad) # TRUE

## handling of character values
as.im("a", W=letterR, na.replace="b")
as.im("a", W=letterR, na.replace="b", stringsAsFactors=FALSE)
```

`as.layered` *Convert Data To Layered Object*

Description

Converts spatial data into a layered object.

Usage

```
as.layered(X)

## Default S3 method:
as.layered(X)

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

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

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

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

Arguments

`X`	Some kind of spatial data.

Details

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

The argument `X` should contain some kind of spatial data such as a point pattern, window, or pixel image.

If X is a simple object then it will be converted into a layered object containing only one layer which is equivalent to X.

If X can be interpreted as consisting of multiple layers of data, then the result will be a layered object consisting of these separate layers of data.

- if X is a list of class "listof" or "solist", then as.layered(X) consists of several layers, one for each entry in the list X;
- if X is a multitype point pattern, then as.layered(X) consists of several layers, each containing the sub-pattern consisting of points of one type;
- if X is a vector-valued measure, then as.layered(X) consists of several layers, each containing a scalar-valued measure.

Value

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

Author(s)

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

See Also

as.layered.msr, layered, split.ppp

Examples

```
as.layered(cells)
as.layered(amacrine)
```

---

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,
        rule.eps=c("adjust.eps", "grow.frame", "shrink.frame"))
```

Arguments

w	A window (object of class "owin") or data acceptable to as.owin.
eps	(optional) width and height of pixels. A single number, or a numeric vector of length 2.
dimyx	(optional) pixel array dimensions. A single integer, or an integer vector of length 2 giving dimensions in the *y* and *x* directions.
xy	(optional) data containing pixel coordinates, such as a pixel image (object of class "im"), or a window of type "mask". See Details.
rule.eps	Character string (partially matched) specifying what to do when eps is not a divisor of the frame size. Ignored if eps is missing or null. See Details.
...	Other arguments are ignored.

Details

A 'mask' is a spatial window that is represented by a pixel image with binary values. It is an object of class "owin" with type "mask".

This function as.mask creates a representation of any spatial window w as a mask. It generates a rectangular grid of locations in the plane, tests whether each of these locations lies inside w, and stores the results as a mask.

The most common use of this function is to approximate the shape of a rectangular or polygonal window w by a mask, for computational purposes. 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 argument w should be a window (object of class "owin"). If it is another kind of spatial data, then the window information will be extracted using as.owin.

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

If eps is given, then it specifies the *desired* grid spacing, that is, the desired size of the pixels. If eps is a single number, it specifies that the desired grid spacing is eps in both the x and y directions, that is, the desired pixels are squares with side length eps. If eps is a vector of length 2, it specifies that the desired grid spacing is eps[1] in the x direction and eps[2] in the y direction. That is, the desired pixels are rectangles of width eps[1] and height eps[2].

When eps is given, the argument rule.eps specifies what to do if pixels of the desired size would not fit exactly into the rectangular frame of w.

- if rule.eps="adjust.eps" (the default), the rectangular frame will remain unchanged, and the grid spacing (pixel size) eps will be reduced slightly so that an integer number of pixels fits exactly into the frame.
- if rule.eps="grow.frame", the grid spacing (pixel size) eps will remain unchanged, and the rectangular frame will be expanded slightly so that it consists of an integer number of pixels in each direction.
- if rule.eps="shrink.frame", the grid spacing (pixel size) eps will remain unchanged, and the rectangular frame will be contracted slightly so that it consists of an integer number of pixels in each 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. The grid spacing (pixel size) is determined by the frame size and the number of pixels.

If xy is given, then this should be some kind of data specifing the coordinates of a pixel grid. It may be

- a list or structure containing elements x and y which are numeric vectors of equal length. These will be taken as x and y coordinates of the margins of the grid. The pixel coordinates will be generated from these two vectors.
- a pixel image (object of class "im").
- a window (object of class "owin") which is of type "mask" so that it contains pixel coordinates.

If xy is given and is either a pixel image or a mask, then w may be omitted, and the window information will be extracted from xy.

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.

Discretisation rule

The rule used in as.mask is that a pixel is part of the discretised window if and only if the centre of the pixel falls in the original window. This is usually sufficient for most purposes, and is fast to compute.

Other discretisation rules are possible; they are available using the function owin2mask.

Converting a spatial pattern to a mask

If the intention is to discretise or pixellate a spatial pattern, such as a point pattern, line segment pattern or a linear network, then as.mask is not the appropriate function to use, because as.mask extracts only the window information and converts this window to a mask.

To discretise a point pattern, use pixellate.ppp. To discretise a line segment pattern, use pixellate.psp or psp2mask. To discretise a linear network, use pixellate.linnet.

Author(s)

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

See Also

owin2mask.

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)))
m <- as.mask(w)
if(interactive()) {
   plot(w)
   plot(m)
}
x <- 1:9
y <- seq(0.25, 9.75, by=0.5)
m <- as.mask(w, xy=list(x=x, y=y))

B <- square(1)
as.mask(B, eps=0.3)
as.mask(B, eps=0.3, rule.eps="g")
as.mask(B, eps=0.3, rule.eps="s")
```

---

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

as.matrix.owin

Examples

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

M <- as.matrix(Z)

median(M)

# plot the cumulative distribution function of pixel values
# plot(ecdf(as.matrix(Z)))
```

---

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 matrux.

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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)

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

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

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

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

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

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

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

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

Arguments

W	Data specifying an observation window, in any of several formats described under *Details* below.
fatal	Logical value determining what to do if the data cannot be converted to an observation window. See Details.
...	Ignored.
step	Optional. A single number, or numeric vector of length 2, giving the grid step lengths in the x and y directions.

Details

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

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

The argument W may be

- an object of class "owin"
- a structure with entries xrange, yrange specifying the x and y dimensions of a rectangle
- a structure with entries named xmin, xmax, ymin, ymax (in any order) specifying the x and y dimensions of a rectangle. This will accept objects of class bbox in the sf package.

- a numeric vector of length 4 (interpreted as `(xmin, xmax, ymin, ymax)` in that order) specifying the x and y dimensions of a rectangle
- a structure with entries named `xl`, `xu`, `yl`, `yu` (in any order) specifying the x and y dimensions of a rectangle as `(xmin, xmax) = (xl, xu)` and `(ymin, ymax) = (yl, yu)`. This will accept objects of class `spp` used in the Venables and Ripley **spatial** package.
- an object of class `"ppp"` representing a point pattern. In this case, the object's `window` structure will be extracted.
- an object of class `"psp"` representing a line segment pattern. In this case, the object's `window` structure will be extracted.
- an object of class `"tess"` representing a tessellation. In this case, the object's `window` structure will be extracted.
- an object of class `"quad"` representing a quadrature scheme. In this case, the window of the `data` component will be extracted.
- an object of class `"im"` representing a pixel image. In this case, a window of type `"mask"` will be returned, with the same pixel raster coordinates as the image. An image pixel value of `NA`, signifying that the pixel lies outside the window, is transformed into the logical value `FALSE`, which is the corresponding convention for window masks.
- an object of class `"ppm"`, `"kppm"`, `"slrm"` or `"dppm"` representing a fitted point process model. In this case, if `from="data"` (the default), `as.owin` extracts the original point pattern data to which the model was fitted, and returns the observation window of this point pattern. If `from="covariates"` then `as.owin` extracts the covariate images to which the model was fitted, and returns a binary mask window that specifies the pixel locations.
- an object of class `"lpp"` representing a point pattern on a linear network. In this case, `as.owin` extracts the linear network and returns a window containing this network.
- an object of class `"lppm"` representing a fitted point process model on a linear network. In this case, `as.owin` extracts the linear network and returns a window containing this network.
- A `data.frame` with exactly three columns. Each row of the data frame corresponds to one pixel. Each row contains the x and y coordinates of a pixel, and a logical value indicating whether the pixel lies inside the window.
- A `data.frame` with exactly two columns. Each row of the data frame contains the x and y coordinates of a pixel that lies inside the window.
- an object of class `"distfun"`, `"nnfun"` or `"funxy"` representing a function of spatial location, defined on a spatial domain. The spatial domain of the function will be extracted.
- an object of class `"rmhmodel"` representing a point process model that can be simulated using `rmh`. The window (spatial domain) of the model will be extracted. The window may be `NULL` in some circumstances (indicating that the simulation window has not yet been determined). This is not treated as an error, because the argument `fatal` defaults to `FALSE` for this method.
- an object of class `"layered"` representing a list of spatial objects. See `layered`. In this case, `as.owin` will be applied to each of the objects in the list, and the union of these windows will be returned.
- an object of another suitable class from another package. For full details, see `vignette('shapefiles')`.

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

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

Value

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

Author(s)

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

See Also

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

owin.object, owin.

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

Examples

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

# Venables & Ripley 'spatial' package
spatialpath <- system.file(package="spatial")
if(nchar(spatialpath) > 0) {
  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, repair=FALSE)
```

Arguments

W	A window (object of class "owin").
repair	Logical value indicating whether to check the validity of the polygon data and repair it, if W is already a polygonal window.

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

If W is already polygonal, it is returned unchanged, by default. However if repair=TRUE then the validity of the polygonal coordinates will be checked (for example to check the boundary is not self-intersecting) and repaired if necessary, so that the result could be different from W.

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

See Also

owin, as.owin, as.mask, simplify.owin

Examples

```
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 spatial 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)

  ## 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 if X contains window information.)
...	Optional arguments passed to ppp.
fatal	Logical value specifying what to do if the data cannot be converted. 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 function as.ppp is generic, with methods for the classes "ppp", "psp", "quad", "matrix", "data.frame" and a default method.

The dataset X may be:

- an object of class "ppp"
- an object of class "psp"
- a point pattern object created by 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.

In the **spatial** library, a point pattern is represented in either of the following formats:

- (in **spatial** versions 1 to 6) a structure with entries x, y xl, xu, yl, yu
- (in **spatial** version 7) a structure with entries x, y and area, where area is a structure with entries xl, xu, yl, yu

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.

Point pattern datasets can also be created by the function ppp.

Methods for as.ppp exist for some other classes of data; they are listed by methods(as.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.

Checking Validity of Data

By default, the validity of the data will be checked, unless X is already a point pattern object.

The additional arguments check and checkdup may be passed to ppp to specify whether the validity of the input data should be checked:

- If checkdup=TRUE, the algorithm issues a warning if there are any duplicated points.
- If check=TRUE, then any points which lie outside the observation window will be removed, as described in the help file for ppp.

If X is already an object of class "ppp", no checking is performed by default (the default settings are check=checkdup=FALSE).

For all other kinds of object X, checking is performed by default (default settings are check=checkdup=TRUE).

Author(s)

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

See Also

ppp, ppp.object, as.owin, owin.object.

Methods for as.ppp exist for some other classes of data; they are listed by methods(as.ppp).

Examples

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

# Venables-Ripley format
# check for 'spatial' package
spatialpath <- system.file(package="spatial")
if(nchar(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)

  ## Default S3 method:
as.psp(x, ..., window=NULL, marks=NULL,
       check=spatstat.options("checksegments"), fatal=TRUE)
```

Arguments

x	Data which will be converted into a line segment pattern
window	Data which define a window for the pattern.
...	Ignored.
marks	(Optional) vector or data frame of marks for the pattern
check	Logical value indicating whether to check the validity of the data, e.g. to check that the line segments lie inside the window.
fatal	Logical value. See Details.
from, to	Point patterns (object of class "ppp") containing the first and second endpoints (respectively) of each segment. Incompatible with x.

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

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

See Also

psp, psp.object, as.owin, owin.object.

See edges for extracting the edges of a polygonal window as a "psp" 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=runifrect(10), to=runifrect(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 `boundingbox` 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@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

owin, as.owin, boundingbox

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

as.rectangle(lansing)

as.rectangle(copper$SouthLines)
```

---

as.solist *Convert List of Two-Dimensional Spatial Objects*

Description

Given a list of two-dimensional spatial objects, convert it to the class "solist".

Usage

```
as.solist(x, ...)
```

Arguments

x	A list of objects, each representing a two-dimensional spatial dataset.
...	Additional arguments passed to solist.

Details

This command makes the list x into an object of class "solist" (spatial object list). See solist for details.

The entries in the list x should be two-dimensional spatial datasets (not necessarily of the same class).

Value

A list, usually of class "solist".

Author(s)

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

See Also

solist, as.anylist, solapply.

Examples

```
x <- list(cells, Window(cells), setcov(Window(cells)))
y <- as.solist(x)
```

`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 '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@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

tess

quadratcount

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
qNZ <- quadratcount(nztrees, nx=4, ny=3)
as.tess(qNZ)
```

---

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=c("image", "matrix", "coords"), method=c("C", "interpreted"))
```

Arguments

w	A window (object of class "owin").
...	Arguments passed to as.mask to determine the pixel resolution.
style	Character string (partially matched) determining the format of the output: either "matrix", "coords" or "image".
method	Choice of algorithm to use when w is polygonal.

Details

This function computes, for each pixel u in the Frame containing the window `w`, the shortest distance $d(u, w^c)$ from u to the complement of w. This value is zero for pixels lying outside `w`, and is positive for pixels inside `w`.

If the window is a binary mask then the distance from each pixel to the boundary is computed using the distance transform algorithm `distmap.owin`. The result is equivalent to `distmap(W, invert=TRUE)`.

If the window is a rectangle or a polygonal region, the grid of pixels is determined by the arguments `"\dots"` passed to `as.mask`. The distance from each pixel to the boundary is calculated exactly, using analytic geometry. This is slower but more accurate than in the case of a binary mask.

For software testing purposes, there are two implementations available when `w` is a polygon: the default is `method="C"` which is much faster than `method="interpreted"`.

To compute the distance from each pixel to the bounding rectangular frame `Frame(W)`, use `framedist.pixels`.

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

See Also

`framedist.pixels`

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

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
opa <- par(mfrow=c(1,2))
plot(bdist.pixels(letterR))
plot(framedist.pixels(letterR))
par(opa)
```

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 Window(X) 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@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

bdist.pixels, bdist.tiles, ppp.object, erosion

Examples

```
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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

tess, bdist.points, bdist.pixels

Examples

```
P <- runifrect(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)
```

beachcolours *Create Colour Scheme for a Range of Numbers*

Description

Given a range of numerical values, this command creates a colour scheme that would be appropriate if the numbers were altitudes (elevation above or below sea level).

Usage

```
beachcolours(range, sealevel = 0, monochrome = FALSE,
             ncolours = if (monochrome) 16 else 64,
             nbeach = 1)
beachcolourmap(range, ...)
```

Arguments

range	Range of numerical values to be mapped. A numeric vector of length 2.
sealevel	Value that should be treated as zero. A single number, lying between range[1] and range[2].
monochrome	Logical. If TRUE then a greyscale colour map is constructed.
ncolours	Number of distinct colours to use.
nbeach	Number of colours that will be yellow.
...	Arguments passed to beachcolours.

Details

Given a range of numerical values, these commands create a colour scheme that would be appropriate if the numbers were altitudes (elevation above or below sea level).

Numerical values close to zero are portrayed in green (representing the waterline). Negative values are blue (representing water) and positive values are yellow to red (representing land). At least, these are the colours of land and sea in Western Australia. This colour scheme was proposed by Baddeley et al (2005).

The function beachcolours returns these colours as a character vector, while beachcolourmap returns a colourmap object.

The argument range should be a numeric vector of length 2 giving a range of numerical values.

The argument sealevel specifies the height value that will be treated as zero, and mapped to the colour green. A vector of ncolours colours will be created, of which nbeach colours will be green.

The argument monochrome is included for convenience when preparing publications. If monochrome=TRUE the colour map will be a simple grey scale containing ncolours shades from black to white.

Value

For beachcolours, a character vector of length ncolours specifying colour values. For beachcolourmap, a colour map (object of class "colourmap").

Author(s)

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

References

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

See Also

colourmap, colourtools.

Examples

```
plot(beachcolourmap(c(-2,2)))
```

---

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

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

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

erosion, dilation

Examples

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

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

---

bounding.box.xy *Convex Hull of Points*

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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

---

boundingbox *Bounding Box of a Window, Image, or Point Pattern*

Description

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

Usage

```
boundingbox(...)

## Default S3 method:
boundingbox(...)

## S3 method for class 'im'
boundingbox(...)

## S3 method for class 'owin'
boundingbox(...)

## S3 method for class 'ppp'
boundingbox(...)
```

```
## S3 method for class 'psp'
boundingbox(...)

## S3 method for class 'lpp'
boundingbox(...)

## S3 method for class 'linnet'
boundingbox(...)

## S3 method for class 'solist'
boundingbox(...)
```

Arguments

...	One or more windows (objects of class "owin"), pixel images (objects of class "im") or point patterns (objects of class "ppp" or "lpp") or line segment patterns (objects of class "psp") or linear networks (objects of class "linnet") or any combination of such objects. Alternatively, the argument may be a list of such objects, of class "solist".

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" or "lpp"), the bounding box is the smallest rectangle that contains all the points of the pattern. This is usually smaller than the bounding box of the window of the point pattern.

For a line segment pattern (object of class "psp") or a linear network (object of class "linnet"), the bounding box is the smallest rectangle that contains all endpoints of line segments.

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

owin, as.owin, as.rectangle

Author(s)

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

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 <- boundingbox(w)
# returns rectangle [1,3] x [2,7]

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

---

boundingcircle *Smallest Enclosing Circle*

Description

Find the smallest circle enclosing a spatial window or other object. Return its radius, or the location of its centre, or the circle itself.

Usage

```
boundingradius(x, ...)

boundingcentre(x, ...)

boundingcircle(x, ...)

## S3 method for class 'owin'
boundingradius(x, ...)

## S3 method for class 'owin'
boundingcentre(x, ...)

## S3 method for class 'owin'
boundingcircle(x, ...)

## S3 method for class 'ppp'
boundingradius(x, ...)

## S3 method for class 'ppp'
boundingcentre(x, ...)

## S3 method for class 'ppp'
boundingcircle(x, ...)
```

Arguments

x	A window (object of class "owin"), or another spatial object.
...	Arguments passed to as.mask to determine the pixel resolution for the calculation.

Details

The boundingcircle of a spatial region W is the smallest circle that contains W. The boundingradius is the radius of this circle, and the boundingcentre is the centre of the circle.

The functions boundingcircle, boundingcentre and boundingradius are generic. There are methods for objects of class "owin", "ppp" and "linnet".

Value

The result of boundingradius is a single numeric value.

The result of boundingcentre is a point pattern containing a single point.

The result of boundingcircle is a window representing the boundingcircle.

Author(s)

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

See Also

diameter

Examples

```
boundingradius(letterR)

plot(grow.rectangle(Frame(letterR), 0.2), main="", type="n")
plot(letterR, add=TRUE, col="grey")
plot(boundingcircle(letterR), add=TRUE, border="green", lwd=2)
plot(boundingcentre(letterR), pch="+", cex=2, col="blue", add=TRUE)

X <- runifrect(5)
plot(X)
plot(boundingcircle(X), add=TRUE)
plot(boundingcentre(X), pch="+", cex=2, col="blue", add=TRUE)
```

---

box3 *Three-Dimensional Box*

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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))
```

---

boxx *Multi-Dimensional Box*

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

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"))
```

---

bufftess *Buffer Distance Tessellation*

Description

Constructs a spatial tessellation, composed of rings or buffers at specified distances away from the given spatial object.

Usage

```
bufftess(X, breaks, W = Window(X), ..., polygonal = TRUE)
```

Arguments

X	A spatial object in two dimensions, such as a point pattern (class "ppp") or line segment pattern (class "psp").
breaks	Either a numeric vector specifying the cut points for the distance values, or a single integer specifying the number of cut points.
W	Optional. Window (object of class "owin") inside which the tessellation will be constructed.
...	Optional arguments passed to as.mask controlling the pixel resolution when polygonal=FALSE, and optional arguments passed to cut.default controlling the labelling of the distance bands.
polygonal	Logical value specifying whether the tessellation should consist of polygonal tiles (polygonal=TRUE, the default) or should be constructed using a pixel image (polygonal=FALSE).

Details

This function divides space into tiles defined by distance from the object X. The result is a tessellation (object of class "tess") that consists of concentric rings around X.

The distance values which determine the tiles are specified by the argument breaks.

- If breaks is a vector of numerical values, then these values are taken to be the distances defining the tiles. The first tile is the region of space that lies at distances between breaks[1] and breaks[2] away from X; the second tile is the region lying at distances between breaks[2] and breaks[3] away from X; and so on. The number of tiles will be length(breaks)-1.
- If breaks is a single integer, it is interpreted as specifying the number of intervals between breakpoints. There will be breaks+1 equally spaced break points, ranging from zero to the maximum achievable distance. The number of tiles will equal breaks.

The tessellation can be computed using either raster calculations or vector calculations.

- If polygonal=TRUE (the default), the tiles are computed as polygonal windows using vector geometry, and the result is a tessellation consisting of polygonal tiles. This calculation could be slow and could require substantial memory, but produces a geometrically accurate result.
- If polygonal=FALSE, the distance map of X is computed as a pixel image (distmap), then the distance values are divided into discrete bands using cut.im. The result is a tessellation specified by a pixel image. This computation is faster but less accurate.

Value

A tessellation (object of class "tess").

The result also has an attribute breaks which is the vector of distance breakpoints.

Author(s)

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

See Also

Polygonal calculations are performed using dilation and setminus.owin. Pixel calculations are performed using distmap and cut.im. See as.mask for details of arguments that control pixel resolution.

For other kinds of tessellations, see tess, hextess, venn.tess, polartess, dirichlet, delaunay, quantess, quadrats and rpoislinetess.

Examples

```
X <- cells[c(FALSE,FALSE,FALSE,TRUE)]
if(interactive()) {
  b <- c(0, 0.05, 0.1, 0.15, 0.2, Inf)
  n <- 5
} else {
  ## simpler data for testing
  b <- c(0, 0.1, 0.2, Inf)
  n <- 3
}
plot(bufftess(X, b), do.col=TRUE, col=1:n)
```

---

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

data	A pixel image (object of class "im").
INDICES	Grouping variable. Either a tessellation (object of class "tess") or a factor-valued pixel image.
FUN	Function to be applied to each sub-image of data.
...	Extra arguments passed to FUN.

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

split.im, tess, im

Examples

```
W <- square(1)
X <- as.im(function(x,y){sqrt(x^2+y^2)}, W)
Y <- dirichlet(runifrect(12, W))
# mean pixel value in each subset
unlist(by(X, Y, mean))
# trimmed mean
unlist(by(X, Y, mean, trim=0.05))
```

---

by.ppp *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

data	Point pattern (object of class "ppp").
INDICES	Grouping variable. Either a factor, a pixel image with factor values, or a tessellation.
FUN	Function to be applied to subsets of data.
...	Additional arguments to FUN.

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 ith 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 "anylist" or "solist" as appropriate) containing the results returned from FUN for each of the subpatterns.

Author(s)

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

See Also

ppp, split.ppp, cut.ppp, tess, im.

Examples

```
# multitype point pattern, broken down by type
by(amacrine, FUN=minnndist)
by(amacrine, FUN=function(x) { intensity(unmark(x)) })

if(require(spatstat.explore)) {
# 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, 4,4)
## compute minimum nearest neighbour distance for points in each tile
B <- by(swedishpines, tes, minnndist)

if(require(spatstat.explore)) {
```

```
B <- by(swedishpines, tes, clarkevans, correction="Donnelly")
simplify2array(B)
}
```

---

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

See Also

hyperframe, as.hyperframe

Examples

```
if(require(spatstat.random)) {
  lambda <- runif(5, min=10, max=30)
  X <- solapply(as.list(lambda), rpoispp)
  h <- hyperframe(lambda=lambda, X=X)
  g <- hyperframe(id=letters[1:5], Y=rev(X))
  gh <- cbind(h, g)
  hh <- rbind(h[1:2, ], h[3:5,])
}
```

`centroid.owin` *Centroid of a window*

Description

Computes the centroid (centre of mass) of a window

Usage

```
centroid.owin(w, as.ppp = FALSE)
```

Arguments

`w`	A window
`as.ppp`	Logical flag indicating whether to return the centroid as a point pattern (ppp object)

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, unless the window is convex. If `as.ppp=TRUE` and the centroid of `w` lies outside `w`, then the window of the returned point pattern will be a rectangle containing the original window (using `as.rectangle`.

Value

Either a list with components `x, y`, or a point pattern (of class `ppp`) consisting of a single point, giving the coordinates of the centroid of the window `w`.

Author(s)

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

See Also

owin, as.owin

Examples

```
w <- owin(c(0,1),c(0,1))
centroid.owin(w)
# returns 0.5, 0.5

w <- Window(demopat)
# an irregular window
cent <- centroid.owin(w, as.ppp = TRUE)

wapprox <- as.mask(w)
# pixel approximation of window

if(interactive()) {
  plot(cent)
  # plot the window and its centroid
  points(centroid.owin(wapprox))
  # should be indistinguishable
}
```

---

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 "infline")

Details

The argument L should be a set of infinite straight lines in the plane (stored in an object L of class "infline" created by the function infline).

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

infline, clip.infline

Examples

```
L <- infline(p=1:3, theta=pi/4)
W <- square(4)
chop.tess(W, L)
```

---

clickbox *Interactively Define a Rectangle*

Description

Allows the user to specify a rectangle by point-and-click in the display.

Usage

```
clickbox(add=TRUE, ...)
```

Arguments

add	Logical value indicating whether to create a new plot (add=FALSE) or draw over the existing plot (add=TRUE).
...	Graphics arguments passed to polygon to plot the box.

Details

This function allows the user to create a rectangular window by interactively clicking on the screen display.

The user is prompted to point the mouse at any desired locations for two corners of the rectangle, and click the left mouse button to add each point.

The return value is a window (object of class `"owin"`) representing the rectangle.

This function uses the R command `locator` to input the mouse clicks. It only works on screen devices such as 'X11', 'windows' and 'quartz'.

Value

A window (object of class `"owin"`) representing the selected rectangle.

Author(s)

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

See Also

`clickpoly`, `clickppp`, `clickdist`, `locator`

---

clickdist *Interactively Measure Distance*

Description

Measures the distance between two points which the user has clicked on.

Usage

```
clickdist()
```

Details

This function allows the user to measure the distance between two spatial locations, interactively, by clicking on the screen display.

When `clickdist()` is called, the user is expected to click two points in the current graphics device. The distance between these points will be returned.

This function uses the R command `locator` to input the mouse clicks. It only works on screen devices such as 'X11', 'windows' and 'quartz'.

Value

A single nonnegative number.

Author(s)

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

See Also

locator, clickppp, clickpoly, clickbox

---

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, ..., snap.step=NULL, snap.origin=c(0,0))
```

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, and to polygon to plot the polygons.
snap.step	Optional. Spatial coordinates will be rounded to the nearest multiple of snap.step. A positive number specifying the step length, or a vector of 2 positive numbers specifying step lengths for the x and y coordinates.
snap.origin	Optional. Numeric vector of length 2. Coordinates of the origin that will be used when rounding coordinates.

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.

If `snap.step` is given, a grid of lines will be drawn with the specified spacing, and the clicked locations will be snapped (rounded) to the nearest grid point.

Value

A window (object of class `"owin"`) representing the polygon.

Author(s)

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

See Also

`identify.ppp`, `clickbox`, `clickppp`, `clickdist`, `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

`n`	Number of points to be added (if this is predetermined).
`win`	Window in which to create the point pattern. An object of class `"owin"`.
`types`	Vector of types, when creating a multitype point pattern.
`...`	Optional extra arguments to be passed to `locator` to control the display.
`add`	Logical value indicating whether to create a new plot (`add=FALSE`) or draw over the existing plot (`add=TRUE`).
`main`	Main heading for plot.
`hook`	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@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>.

See Also

`identify.ppp`, `locator`, `clickpoly`, `clickbox`, `clickdist`

---

clip.infline *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.infline(L, win)
```

Arguments

`L`	Object of class `"infline"` specifying a set of infinite straight lines in the plane.
`win`	Window (object of class `"owin"`).

Details

This function computes the intersection between a set of infinite straight lines in the plane (stored in an object `L` of class `"infline"` created by the function `infline`) and a window `win`. The result is a pattern of line segments. Each line segment carries a mark indicating which line it belongs to.

Value

A line segment pattern (object of class `"psp"`) with a single column of marks.

Author(s)

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

See Also

`infline`,`psp`.

To divide a window into pieces using infinite lines, use `chop.tess`.

Examples

```
L <- infline(p=1:3, theta=pi/4)
W <- square(4)
clip.infline(L, W)
```

---

closepairs *Close Pairs of Points*

Description

Low-level functions to find all close pairs of points.

Usage

```
closepairs(X, rmax, ...)

## S3 method for class 'ppp'
closepairs(X, rmax, twice=TRUE,
                          what=c("all","indices","ijd"),
                          distinct=TRUE, neat=TRUE,
                          periodic=FALSE, ...)

crosspairs(X, Y, rmax, ...)

## S3 method for class 'ppp'
crosspairs(X, Y, rmax,
                         what=c("all", "indices", "ijd"),
                         periodic=FALSE, ...,
                         iX=NULL, iY=NULL)
```

Arguments

X, Y	Point patterns (objects of class "ppp").
rmax	Maximum distance between pairs of points to be counted as close pairs.
twice	Logical value indicating whether all ordered pairs of close points should be returned. If twice=TRUE (the default), each pair will appear twice in the output, as (i,j) and again as (j,i). If twice=FALSE, then each pair will appear only once, as the pair (i,j) with i < j.
what	String specifying the data to be returned for each close pair of points. If what="all" (the default) then the returned information includes the indices i,j of each pair, their x,y coordinates, and the distance between them. If what="indices" then only the indices i,j are returned. If what="ijd" then the indices i,j and the distance d are returned.
distinct	Logical value indicating whether to return only the pairs of points with different indices i and j (distinct=TRUE, the default) or to also include the pairs where i=j (distinct=FALSE).
neat	Logical value indicating whether to ensure that i < j in each output pair, when twice=FALSE.
periodic	Logical value indicating whether to use the periodic edge correction. The window of X should be a rectangle. Opposite pairs of edges of the window will be treated as identical.
...	Extra arguments, ignored by methods.
iX, iY	Optional vectors used to determine whether a point in X is identical to a point in Y. See Details.

Details

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

closepairs(X,rmax) finds all pairs of distinct points in the pattern X which lie at a distance less than or equal to rmax apart, and returns them. The result is a list with the following components:

i Integer vector of indices of the first point in each pair.

j Integer vector of indices of the second point in each pair.

xi,yi Coordinates of the first point in each pair.

xj,yj Coordinates of the second point in each pair.

dx Equal to xj-xi

dy Equal to yj-yi

d Euclidean distance between each pair of points.

If what="indices" then only the components i and j are returned. This is slightly faster and more efficient with use of memory.

crosspairs(X,rmax) identifies all pairs of neighbours (X[i], Y[j]) between the patterns X and Y, and returns them. The result is a list with the same format as for closepairs.

The arguments iX and iY are used when the two point patterns X and Y may have some points in common. In this situation crosspairs(X, Y) would return some pairs of points in which the two points are identical. To avoid this, attach a unique integer identifier to each point, such that two points are identical if their identifier values 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.

Value

A list with components i and j, and possibly other components as described under Details.

Warning about accuracy

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

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

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

Author(s)

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

See Also

closepairs.pp3 for the corresponding functions for 3D point patterns.

Kest, Kcross, nndist, nncross, applynbd, markstat for functions which use these capabilities.

Examples

```
d <- closepairs(cells, 0.1)
head(as.data.frame(d))

Y <- split(amacrine)
e <- crosspairs(Y$on, Y$off, 0.1)
```

---

closepairs.pp3 *Close Pairs of Points in 3 Dimensions*

Description

Low-level functions to find all close pairs of points in three-dimensional point patterns.

Usage

```
## S3 method for class 'pp3'
closepairs(X, rmax, twice=TRUE,
                        what=c("all", "indices", "ijd"),
                        distinct=TRUE, neat=TRUE, ...)

## S3 method for class 'pp3'
crosspairs(X, Y, rmax, what=c("all", "indices", "ijd"), ...)
```

Arguments

X, Y	Point patterns in three dimensions (objects of class "pp3").
rmax	Maximum distance between pairs of points to be counted as close pairs.
twice	Logical value indicating whether all ordered pairs of close points should be returned. If twice=TRUE, each pair will appear twice in the output, as (i,j) and again as (j,i). If twice=FALSE, then each pair will appear only once, as the pair (i,j) such that i < j.
what	String specifying the data to be returned for each close pair of points. If what="all" (the default) then the returned information includes the indices i,j of each pair, their x,y,z coordinates, and the distance between them. If what="indices" then only the indices i,j are returned. If what="ijd" then the indices i,j and the distance d are returned.
distinct	Logical value indicating whether to return only the pairs of points with different indices i and j (distinct=TRUE, the default) or to also include the pairs where i=j (distinct=FALSE).
neat	Logical value indicating whether to ensure that i < j in each output pair, when twice=FALSE.
...	Ignored.

Details

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

closepairs(X,rmax) identifies all pairs of neighbours in the pattern X and returns them. The result is a list with the following components:

i Integer vector of indices of the first point in each pair.

j Integer vector of indices of the second point in each pair.

xi,yi,zi Coordinates of the first point in each pair.

xj,yj,zj Coordinates of the second point in each pair.

dx Equal to xj-xi

dy Equal to yj-yi

dz Equal to zj-zi

d Euclidean distance between each pair of points.

If what="indices" then only the components i and j are returned. This is slightly faster.

crosspairs(X,rmax) identifies all pairs of neighbours (X[i], Y[j]) between the patterns X and Y, and returns them. The result is a list with the same format as for closepairs.

Value

A list with components i and j, and possibly other components as described under Details.

Warning about accuracy

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

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

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

Author(s)

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

, Rolf Turner <rolfturner@posteo.net>

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

See Also

closepairs

Examples

```
X <- pp3(runif(10), runif(10), runif(10), box3(c(0,1)))
Y <- pp3(runif(10), runif(10), runif(10), box3(c(0,1)))
a <- closepairs(X, 0.1)
b <- crosspairs(X, Y, 0.1)
```

---

closetriples *Close Triples of Points*

Description

Low-level function to find all close triples of points.

Usage

```
closetriples(X, rmax)
```

Arguments

X	Point pattern (object of class "ppp" or "pp3").
rmax	Maximum distance between each pair of points in a triple.

Details

This low-level function finds all triples of points in a point pattern in which each pair lies closer than rmax.

Value

A data frame with columns i,j,k giving the indices of the points in each triple, and a column diam giving the diameter (maximum pairwise distance) in the triple.

Author(s)

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

See Also

closepairs, Tstat.

Examples

```
closetriples(redwoodfull, 0.02)
closetriples(redwoodfull, 0.005)
```

---

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

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 closing.
...	extra arguments passed to as.mask 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 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@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

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

```
v <- closing(letterR, 0.25)
plot(v, main="closing")
plot(letterR, add=TRUE)

plot(closing(cells, 0.1))
points(cells)
```

`colourmap` *Colour Lookup Tables*

Description

Create a colour map (colour lookup table).

Usage

```
colourmap(col, ..., range=NULL, breaks=NULL, inputs=NULL, gamma=1,
          compress=NULL, decompress=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`.
`gamma`	Exponent for the gamma correction, when `range` is given. A single positive number. See Details.
`compress`	Optional. Experimental. An R function determining a nonlinear transformation of the domain of the colour map. See section on *Nonlinear colour maps*.
`decompress`	Experimental. An R function giving the inverse function of `compress`, if `compress` is specified.

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]. The argument col should have the same length as inputs.
- 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. The interval will be divided evenly into bands, each of which is assigned one of the colours in col. (If gamma is given, then the bands are equally spaced on a scale where the original values are raised to the power gamma.) (See the section on *Nonlinear colour maps* for the case where compress and decompress are given.)
- 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 argument col should have length equal to length(breaks) - 1.

It is also permissible for col to be a single colour value, representing a trivial colour map in which all data values are mapped to the same colour.

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" and "lut".

Nonlinear colour maps

If the arguments compress and decompress are given, they define a transformation of the range of numbers. A typical example would be a logarithmic colour map defined by compress = log10 and decompress = function(x) { 10^x }.

These functions have no effect on the interpretation of the arguments range, breaks and inputs. However, if range is given, then the range of values will be divided into intervals which have equal length on the scale defined by compress. That is, the range of numbers determined by compress(range) will be evenly divided into intervals, and these intervals will be mapped back to the original scale by decompress to determine the breaks. For a logarithmic colour map with compress=log10, the specified range will be divided into intervals which are equal on a logarithmic scale.

The arguments compress and decompress affect the way in which the colour map is plotted by plot.colourmap. For a continuous colour map, the range of input values is plotted on the compressed scale, but annotated on the original scale. See the Examples.

The arguments compress and decompress should be functions which are vectorised (i.e. if x is a vector then compress(x) and decompress(x) are also vectors of the same length as x) and increasing (if x < y then compress(x) < compress(y) and decompress(x) < decompress(y).

The argument decompress is not needed in the following cases:

- If compress is the function log10, then decompress is taken to be its inverse function(x) { 10^x }.
- If compress is a cumulative distribution function (of class "ecdf", "ewcdf" or "interpolatedCDF") then decompress is taken to be its inverse function decompress = quantilefun(compress).

Author(s)

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

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.

To make a smooth transition between colours, see interp.colourmap. To alter individual colour values, see tweak.colourmap. To extract or replace all colour values, see colouroutputs.

See also restrict.colourmap and rev.colourmap.

See colourtools for more tools to manipulate colour values.

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)
## logarithmic colour map
cl <- colourmap(rainbow(25), range=c(0.1, 1000), compress=log10)
plot(cl)
```

---

colouroutputs *Extract or Assign Colour Values in a Colour Map*

Description

Extract the colour values in a colour map, or assign new colour values.

Usage

```
colouroutputs(x)

colouroutputs(x) <- value
```

Arguments

x	A colour map (object of class "colourmap").
value	A vector of values that can be interpreted as colours.

Details

An object of class "colourmap" is effectively a function that maps its inputs (numbers or factor levels) to colour values.

The command colouroutputs(x) extracts the colour values in the colour map x.

The assignment colouroutputs(x) <- value replaces the colour values in the colour map x by the entries in value. The replacement vector value should have the same length as colouroutputs(x), and its entries should be interpretable as colours.

To change only some of the colour values in a colour map, it may be easier to use tweak.colourmap.

Value

The result of colouroutputs is a character vector of colour values. The result of the assignment colouroutputs(x) <- value is another colour map (object of class "colourmap").

Author(s)

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

See Also

colourmap, interp.colourmap, tweak.colourmap, colourtools.

Examples

```
m <- colourmap(rainbow(5), range=c(0,1))
m
# reverse order of colours
colouroutputs(m) <- rev(colouroutputs(m))
m
```

colourtools *Convert and Compare Colours in Different Formats*

Description

These functions convert between different formats for specifying a colour in R, determine whether colours are equivalent, and convert colour to greyscale.

Usage

```
col2hex(x)
rgb2hex(v, maxColorValue=255)
rgb2hsva(red, green=NULL, blue=NULL, alpha=NULL, maxColorValue=255)
paletteindex(x)
samecolour(x,y)
complementarycolour(x)
interp.colours(x, length.out=512)
is.colour(x)
to.grey(x, weights=c(0.299, 0.587, 0.114), transparent=FALSE)
is.grey(x)
to.opaque(x)
to.transparent(x, fraction)
to.saturated(x, s=1)
```

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 of a single colour, or a 3-column matrix giving the RGB values of several colours. Alternatively a vector of length 4 or a matrix with 4 columns, giving the RGB and alpha (transparency) values.
red, green, blue, alpha	Arguments acceptable to rgb determining the red, green, blue channels and optionally the alpha (transparency) channel. Note that red can also be a matrix with 3 **rows** giving the RGB values, or a matrix with 4 rows giving RGB and alpha values.
maxColorValue	Number giving the maximum possible value for the entries in v or red,green,blue,alpha.
weights	Numeric vector of length 3 giving relative weights for the red, green, and blue channels respectively.
transparent	Logical value indicating whether transparent colours should be converted to transparent grey values (transparent=TRUE) or converted to opaque grey values (transparent=FALSE, the default).
fraction	Transparency fraction. Numerical value or vector of values between 0 and 1, giving the opaqueness of a colour. A fully opaque colour has fraction=1.
length.out	Integer. Length of desired sequence.
s	Saturation value (between 0 and 1).

Details

`is.colour(x)` can be applied to any kind of data `x` and returns `TRUE` if `x` can be interpreted as a colour or colours. The remaining functions expect data that can be interpreted as colours.

`col2hex` converts colours specified in any format into their hexadecimal character codes.

`rgb2hex` converts RGB colour values into their hexadecimal character codes. It is a very minor extension to `rgb`. Arguments to `rgb2hex` should be similar to arguments to `rgb`.

`rgb2hsva` converts RGB colour values into HSV colour values including the alpha (transparency) channel. It is an extension of `rgb2hsv`. Arguments to `rgb2hsva` should be similar to arguments to `rgb2hsv`.

`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.

`is.grey` determines whether each entry of `x` is a greyscale colour, and returns a logical vector.

`to.grey` converts the colour data in `x` to greyscale colours. Alternatively `x` can be an object of class `"colourmap"` and `to.grey(x)` is the modified colour map.

`to.opaque` converts the colours in `x` to opaque (non-transparent) colours, and `to.transparent` converts them to transparent colours with a specified transparency value. Note that `to.transparent(x,1)` is equivalent to `to.opaque(x)`.

For `to.grey`, `to.opaque` and `to.transparent`, if all the data in `x` specifies colours from the standard palette, and if the result would be equivalent to `x`, then the result is identical to `x`.

`to.saturated` converts each colour in `x` to its fully-saturated equivalent. For example, pink is mapped to red. Shades of grey are converted to black; white is unchanged.

`complementarycolour` replaces each colour by its complementary colour in RGB space (the colour obtained by replacing RGB values `(r, g, b)` by `(255-r, 255-g, 255-b)`). The transparency value is not changed. Alternatively `x` can be an object of class `"colourmap"` and `complementarycolour(x)` is the modified colour map.

`interp.colours` interpolates between each successive pair of colours in a sequence of colours, to generate a more finely-spaced sequence. It uses linear interpolation in HSV space (with hue represented as a two-dimensional unit vector).

Value

For `col2hex` and `rgb2hex` a character vector containing hexadecimal colour codes.

For `to.grey`, `to.opaque` and `to.transparent`, either a character vector containing hexadecimal colour codes, or a value identical to the input `x`.

For `rgb2hsva`, a matrix with 3 or 4 rows containing HSV colour values.

For `paletteindex`, an integer vector, possibly containing `NA` values.

For `samecolour` and `is.grey`, 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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

col2rgb, rgb2hsv, palette.

See also the class of colour map objects in the **spatstat** package: colourmap, interp.colourmap, tweak.colourmap.

Examples

```
samecolour("grey", "gray")
paletteindex("grey")
col2hex("orange")
to.grey("orange")
to.saturated("orange")
complementarycolour("orange")
is.grey("lightgrey")
is.grey(8)
to.transparent("orange", 0.5)
to.opaque("red")
interp.colours(c("orange", "red", "violet"), 5)
```

---

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

See Also

harmonise.im, compatible.im, as.im

Examples

```
if(require(spatstat.random)) {
   A <- setcov(square(1), dimyx=32)
   G <- as.im(function(x,y) { x^2 - y }, W=owin(), dimyx=8)
   H <- commonGrid(A, letterR, G)
   newR <- as.mask(letterR, xy=H)
   newG <- as.im(G, W=H)
   if(interactive()) plot(solist(G=newG, R=newR), main="")
}
```

---

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 "unitname". See the documentation for these methods for further information.

Value

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

Author(s)

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

See Also

compatible.fv, compatible.fasp, compatible.im, compatible.unitname

---

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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

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
w <- Window(demopat)
outside <- complement.owin(w)
# mask
w <- as.mask(Window(demopat))
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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

superimpose, quadscheme

Examples

```
dat <- runifrect(30)
xy <- list(x=runif(10),y=runif(10))
new <- concatxy(dat, xy)
```

---

connected *Connected components*

Description

Finds the topologically-connected components of a spatial object, such as the connected clumps of pixels in a binary image.

Usage

```
connected(X, ...)

## S3 method for class 'owin'
connected(X, ..., polygonal=FALSE, method="C", connect=8)

## S3 method for class 'im'
connected(X, ..., background, method="C", connect=8)
```

Arguments

X	A spatial object such as a pixel image (object of class "im") or a window (object of class "owin").
polygonal	Logical value specifying whether to use polygonal geometry. See Details.
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.
...	Arguments passed to as.mask to determine the pixel resolution.
connect	The connectivity of the pixel grid: either 8 or 4.

Details

The function connected is generic, with methods for pixel images (class "im") and windows (class "owin") described here. There are also methods for tessellations (connected.tess), point patterns (connected.ppp and connected.lpp), and linear networks (connected.linnet).

The functions described here compute the connected component transform (Rosenfeld and Pfalz, 1966) of a binary image or binary mask. The pixel values in X are first mapped to logical values. Then the algorithm identifies the connected components (topologically-connected clumps of pixels) in the foreground.

When X is converted to a logical valued image, the pixel value NA is always mapped to FALSE. If background is specified, then the value background is also mapped to FALSE. All other values are mapped to TRUE. In the special case where X is already a logical-valued image, the default is background=FALSE, so that the values NA and FALSE are mapped to FALSE.

Two pixels belong to the same connected component if they have the value TRUE and if they are neighbours. 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.

Pixels are defined to be neighbours if they are physically adjacent to each other. If connect=4, each pixel has 4 neighbours, lying one step above or below, or one step to the left or right. If connect=8 (the default), each pixel has 8 neighbours, lying one step above or below, or one step to the left or right, or one diagonal step away. (Pixels at the edge of the image have fewer neighbours.) The 8-connected algorithm is the default because it gives better results when the pixel grid is coarse. The 4-connected algorithm is faster and is recommended when the pixel grid is fine.

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).

By default, 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.

If X is a window and polygonal=TRUE, the result is a tessellation (object of class "tess") whose tiles are the connected components.

Value

A pixel image (object of class "im") with factor values. The levels of the factor correspond to the connected components.

For connected.owin, if polygonal=TRUE, the result is a tessellation (object of class "tess") whose tiles are 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@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>.

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

connected.ppp, connected.tess, im.object, tess

Examples

```
d <- distmap(cells, dimyx=256)
X <- levelset(d, 0.07)
plot(X)
Z <- connected(X)
plot(Z)
# or equivalently
Z <- connected(d <= 0.07)

# 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))

## polygonal algorithm
A <- regularpolygon(7)
B <- regularpolygon(13, 0.3)
D <- owin(c(-1, -0.6), c(1, 1.4))
E <- regularpolygon(5, 0.3)
Funky <- union.owin(D, E, setminus.owin(A, B))
plot(Funky, col="blue")
plot(connected(Funky, polygonal=TRUE), do.col=TRUE)
```

connected.ppp *Connected Components of a Point Pattern*

Description

Finds the topologically-connected components of a point pattern, when all pairs of points closer than a threshold distance are joined.

Usage

```
## S3 method for class 'ppp'
connected(X, R, ...)

## S3 method for class 'pp3'
connected(X, R, ...)
```

Arguments

X	A point pattern (object of class "ppp" or "pp3").
R	Threshold distance. Pairs of points closer than R units apart will be joined together.
...	Other arguments, not recognised by these methods.

Details

This function can be used to identify clumps of points in a point pattern.

The function connected is generic. This file documents the methods for point patterns in dimension two or three (objects of class "ppp" or "pp3").

The point pattern X is first converted into an abstract graph by joining every pair of points that lie closer than R units apart. Then the connected components of this graph are identified.

Two points in X belong to the same connected component if they can be reached by a series of steps between points of X, each step being shorter than R units in length.

The result is a vector of labels for the points of X where all the points in a connected component have the same label.

Value

A point pattern, equivalent to X except that the points have factor-valued marks, with levels corresponding to the connected components.

Author(s)

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

See Also

connected.im, im.object, tess

Examples

```
Y <- connected(redwoodfull, 0.1)
if(interactive()) {
 plot(Y, cols=1:length(levels(marks(Y))),
      main="connected(redwoodfull, 0.1)")
}
X <- osteo$pts[[1]]
Z <- connected(X, 32)
if(interactive()) {
 plot(Z, col=marks(Z), main="")
}
```

---

connected.tess *Connected Components of Tiles of a Tessellation*

Description

Given a tessellation, find the topologically-connected pieces of each tile, and make a new tessellation using these pieces.

Usage

```
## S3 method for class 'tess'
connected(X, ...)
```

Arguments

X	A tessellation (object of class "tess").
...	Arguments passed to as.mask to determine the pixel resolution.

Details

The function connected is generic. This function connected.tess is the method for tessellations.

Given the tessellation X, the algorithm considers each tile of the tessellation, and identifies its connected components (topologically-connected pieces) using connected.owin. Each of these pieces is treated as a distinct tile and a new tessellation is made from these pieces.

The result is another tessellation obtained by subdividing each tile of X into one or more new tiles.

Value

Another tessellation (object of class "tess").

Author(s)

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

See Also

connected.owin

Examples

```
BB <- grow.rectangle(Frame(letterR), 0.2)
H <- tess(tiles=list(IN=letterR, OUT=complement.owin(letterR, BB)))
opa <- par(mfrow=c(1,2))
plot(H, do.col=TRUE)
plot(connected(H), do.col=TRUE, col=2:4)
par(opa)
```

---

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=FALSE, add=FALSE,
           nlevels=10, levels=NULL, labels=NULL, log=FALSE,
           col=par("fg"),
           clipwin=NULL, show.all=!add, do.plot=TRUE)
```

Arguments

x	Pixel image to be plotted. An object of class "im".
main	Character string to be displayed as the main title.
nlevels, levels	Arguments passed to contour.default controlling the choice of contour levels.
labels	Arguments passed to contour.default controlling the text labels plotted next to the contour lines.
log	Logical value. If TRUE, the contour levels will be evenly-spaced on a logarithmic scale.
axes	Logical. If TRUE, coordinate axes are plotted (with tick marks) around a region slightly larger than the image window. If FALSE (the default), 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.

col	Colour in which to draw the contour lines. Either a single value that can be interpreted as a colour value, or a colourmap object.
clipwin	Optional. A window (object of class "owin"). Only this subset of the data will be displayed.
...	Other arguments passed to contour.default controlling the contour plot; see Details.
show.all	Logical value indicating whether to display all plot elements including the main title, bounding box, and (if axis=TRUE) coordinate axis markings. Default is TRUE for new plots and FALSE for added plots.
do.plot	Logical value indicating whether to actually perform the plot.

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("par.contour").

If log=TRUE, the contour lines will be evenly-spaced on a logarithmic scale, provided the range of pixel values is at least 1.5 orders of magnitude (a ratio of at least 32). Otherwise the levels will be evenly-spaced on the original scale.

If col is a colour map (object of class "colourmap", see colourmap) then the contours will be plotted in different colours as determined by the colour map. The contour at level z will be plotted in the colour col(z) associated with this level in the colour map.

Value

Invisibly, a rectangle (object of class "owin" specifying a rectangle) containing the plotted region.

Author(s)

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

See Also

im.object, plot.im, persp.im

Examples

```
# an image
Z <- setcov(owin())
contour(Z, axes=TRUE)
contour(Z)

V <- 100 * Z^2 + 1
contour(V, log=TRUE, labcex=1)

co <- colourmap(rainbow(100), range=c(0,1))
contour(Z, col=co, lwd=2)
```

`contour.imlist` *Array of Contour Plots*

Description

Generates an array of contour plots.

Usage

```
  ## S3 method for class 'imlist'
contour(x, ...)

  ## S3 method for class 'listof'
contour(x, ...)
```

Arguments

`x`	An object of the class `"imlist"` representing a list of pixel images. Alternatively `x` may belong to the outdated class `"listof"`.
`...`	Arguments passed to `plot.solist` to control the spatial arrangement of panels, and arguments passed to `contour.im` to control the display of each panel.

Details

This is a method for the generic command `contour` for the class `"imlist"`. An object of class `"imlist"` represents a list of pixel images.

(The outdated class `"listof"` is also handled.)

Each entry in the list `x` will be displayed as a contour plot, in an array of panels laid out on the same graphics display, using `plot.solist`. Invididual panels are plotted by `contour.im`.

Value

Null.

Author(s)

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

See Also

plot.solist, contour.im

Examples

```
# bei.extra is a named list of covariate images
 contour(bei.extra,
         main="Barro Colorado: covariates")
```

---

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

See Also

owin, convexhull.xy, is.convex

Examples

```
W <- Window(demopat)
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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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 <- runifrect(30)
plot(X, main="convexhull.xy(X)")
plot(convexhull.xy(X), add=TRUE)
```

---

convexify *Weil's Convexifying Operation*

Description

Converts the window W into a convex set by rearranging the edges, preserving spatial orientation of each edge.

Usage

```
convexify(W, eps)
```

Arguments

W	A window (object of class "owin").
eps	Optional. Minimum edge length of polygonal approximation, if W is not a polygon.

Details

Weil (1995) defined a convexification operation for windows W that belong to the convex ring (that is, for any W which is a finite union of convex sets). Note that this is **not** the same as the convex hull.

The convexified set $f(W)$ has the same total boundary length as W and the same distribution of orientations of the boundary. If W is a polygonal set, then the convexification $f(W)$ is obtained by rearranging all the edges of W in order of their spatial orientation.

The argument W must be a window. If it is not already a polygonal window, it is first converted to one, using simplify.owin. The edges are sorted in increasing order of angular orientation and reassembled into a convex polygon.

Value

A window (object of class "owin").

Author(s)

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

Rolf Turner <rolfturner@posteo.net>

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

References

Weil, W. (1995) The estimation of mean particle shape and mean particle number in overlapping particle systems in the plane. *Advances in Applied Probability* **27**, 102–119.

See Also

convexhull for the convex hull of a window.

Examples

```
opa <- par(mfrow=c(1,2))
plot(letterR)
plot(convexify(letterR))
par(opa)
```

---

convexmetric *Distance Metric Defined by Convex Set*

Description

Create the distance metric associated with a given convex polygon.

Usage

```
convexmetric(K)
```

Arguments

K	Convex set defining the metric. A polygon that is symmetric about the origin. See Details.

Details

This function creates the distance metric associated with the convex set K so that the unit ball of the metric is equal to K. It returns an object of class "metric" representing the metric (see metric.object).

The argument K must be a window (class "owin"). It will be converted to a polygon. It must be convex, and symmetric about the origin.

To perform distance calculations (for example, nearest-neighbour distances) using this metric instead of the Euclidean metric, first check whether the standard function for this purpose (for example `nndist.ppp`) has an argument named `metric`. If so, use the standard function and add the argument `metric`; if not, use the low-level function `invoke.metric`.

To see which operations are currently supported by the metric, use `summary`, as shown in the examples.

Value

An object of class `"metric"`.

Author(s)

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

See Also

`metric.object`, `invoke.metric`

Examples

```
K <- owin(poly=list(x=c(2.5,2,0.5,-2.5,-2,-0.5),y=c(0,1,2,0,-1,-2)))
plot(K)
points(0,0)
m <- convexmetric(K)
m
summary(m)

## show redwood data and identify point number 43
plot(redwood, main="")
plot(redwood[43], pch=16, add=TRUE)

## compute nearest neighbour distances and identifiers
##   using the distance metric m
nd <- nndist(redwood, metric=m)
nw <- nnwhich(redwood, metric=m)

## Nearest neighbour distance for point number 43 is nd[43]; verify
B43 <- disc(radius=nd[43], centre=redwood[43], metric=m)
plot(B43, add=TRUE)

## nearest neighbour for point number 43 is point number nw[43]; verify
plot(redwood[nw[43]], pch=3, col="red", 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

$$C(v) = \int X(u)Y(v-u)\,\mathrm{d}u$$

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

See Also

imcov, reflect

Examples

```
if(interactive()) {
  X <- as.im(letterR)
  Y <- as.im(square(1))
} else {
  ## coarser image for testing
  X <- as.im(letterR, dimyx=32)
  Y <- as.im(square(1), dimyx=32)
}
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
  ## S3 method for class 'quad'
coords(x, ...)
```

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 multi-dimensional space-time point pattern (object of class "ppx") or a quadrature scheme (object of class "quad").
...	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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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)
```

corners *Corners of a rectangle*

Description

Returns the four corners of a rectangle

Usage

```
corners(window)
```

Arguments

window	A window. An object of class owin, or data in any format acceptable to as.owin().

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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))
```

covering *Cover Region with Discs*

Description

Given a spatial region, this function finds an efficient covering of the region using discs of a chosen radius.

Usage

```
covering(W, r, ..., giveup=1000)
```

Arguments

W	A window (object of class "owin").
r	positive number: the radius of the covering discs.
...	extra arguments passed to as.mask controlling the pixel resolution for the calculations.
giveup	Maximum number of attempts to place additional discs.

Details

This function finds an efficient covering of the window W using discs of the given radius r. The result is a point pattern giving the centres of the discs.

The algorithm tries to use as few discs as possible, but is not guaranteed to find the minimal number of discs. It begins by placing a hexagonal grid of points inside W, then adds further points until every location inside W lies no more than r units away from one of the points.

Value

A point pattern (object of class "ppp") giving the centres of the discs.

Author(s)

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

Examples

```
rr <- 0.5
X <- covering(letterR, rr)
plot(grow.rectangle(Frame(X), rr), type="n", main="")
plot(X, pch=16, add=TRUE, col="red")
plot(letterR, add=TRUE, lwd=3)
plot(X %mark% (2*rr), add=TRUE, markscale=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@curtin.edu.au>

See Also

`crossdist.ppp`, `crossdist.psp`, `crossdist.default`, `pairdist`, `nndist`

`crossdist.default` *Pairwise distances between two different sets of points*

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", squared=FALSE)
```

Arguments

`X`, `Y`	Numeric vectors of equal length specifying the coordinates of the first set of points.
`x2`, `y2`	Numeric vectors of equal length specifying the coordinates of the second set of points.
`...`	Ignored.
`period`	Optional. Dimensions for periodic edge correction.
`method`	String specifying which method of calculation to use. Values are `"C"` and `"interpreted"`.
`squared`	Logical. If `squared=TRUE`, the squared distances are returned instead (this computation is faster).

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@curtin.edu.au>

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, squared=FALSE)
```

Arguments

X, Y	Point patterns in three dimensions (objects of class "pp3").
...	Ignored.
periodic	Logical. Specifies whether to apply a periodic edge correction.
squared	Logical. If squared=TRUE, the squared distances are returned instead (this computation is faster).

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@curtin.edu.au> based on code for two dimensions by Pavel Grabarnik <pavel.grabar@issp.serpukhov.su>.

See Also

crossdist, pairdist, nndist, G3est

Examples

```
if(require(spatstat.random)) {
   X <- runifpoint3(20)
   Y <- runifpoint3(30)
} else {
   X <- osteo$pts[[1]]
   Y <- osteo$pts[[2]]
   Y <- Y[domain(X)]
}
   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", squared=FALSE,
                metric=NULL)
```

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".
squared	Logical. If squared=TRUE, the squared distances are returned instead (this computation is faster).
metric	Optional. A distance metric (object of class "metric", see metric.object) which will be used to compute the distances.

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@curtin.edu.au>.

See Also

crossdist, crossdist.default, crossdist.psp, pairdist, nndist, Gest

Examples

```
Y <- runifrect(6, Window(cells))
d <- crossdist(cells, Y)
d <- crossdist(cells, Y, 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@curtin.edu.au>

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="C", type="Hausdorff")
```

Arguments

X, Y	Line segment patterns (objects of class "psp").
...	Ignored.
method	String specifying which method of calculation to use. Values are "C" and "interpreted". Usually not specified.
type	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="C" (the default) then compiled C code is used. The C 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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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,fatal=TRUE,details=FALSE)
```

Arguments

`A, B`	Line segment patterns (objects of class `"psp"`).
`details`	Logical value indicating whether to return additional information. See below.
`fatal`	Logical value indicating what to do if the windows of `A` and `B` do not overlap. See Details.

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.

The result is a point pattern consisting of all the intersection points.

If `details=TRUE`, additional information is computed, specifying where each intersection point came from. The resulting point pattern has a data frame of marks, with columns named `iA, jB, tA, tB`. The marks `iA` and `jB` are the indices of the line segments in `A` and `B`, respectively, which produced each intersection point. The marks `tA` and `tB` are numbers between 0 and 1 specifying the position of the intersection point along the original segments.

If the windows `Window(A)` and `Window(B)` do not overlap, then an error will be reported if `fatal=TRUE`, while if `fatal=FALSE` an error will not occur and the result will be `NULL`.

Value

Point pattern (object of class `"ppp"`).

Author(s)

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

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")
as.data.frame(crossing.psp(a,b,details=TRUE))
```

---

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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.ppp *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 window, a tessellation, or a string giving the name of a column of marks or the name of a spatial coordinate.
...	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 window (object of class "owin"). Each point of x will be classified according to whether it falls inside or outside this window.
- a character string, giving the name of one of the columns of marks(x), if this is a data frame.
- a character string "x" or "y" identifying one of the spatial coordinates.

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 operators [.ppp or subset.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@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

cut, ppp.object, tess

Examples

```
# (1) cutting based on numeric marks of point pattern

trees <- longleaf
# Longleaf Pines data
# the marks are positive real numbers indicating tree diameters.
```

```
if(interactive()) {
plot(trees)
}

# cut the range of tree diameters into three intervals
long3 <- cut(trees, breaks=3)
if(interactive()) {
plot(long3)
}

# adult trees defined to have diameter at least 30 cm
long2 <- cut(trees, 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

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) inside/outside a given region
with(murchison, cut(gold, greenstone))

# (5) multivariate marks
finpines
cut(finpines, "height", breaks=4)
```

`default.colourmap` *Default Colour Map for Plotting a Spatial Pattern*

Description

Determines a colour map for plotting a spatial pattern (or other data) when one is not supplied by the user.

Usage

```
default.colourmap(x, ...,
```

```
col=spatstat.options("image.colfun"),
scramble.cols=FALSE,
monochrome=spatstat.options("monochrome"))
```

Arguments

x	A vector of atomic values, or a factor.
col	Optional. A specification of colours to be used. See Details.
scramble.cols	Logical value. If TRUE, the sequence of colour values will be randomly permuted.
monochrome	Logical value. If TRUE, the colours will be converted to greyscale colours.
...	Additional arguments passed to methods.

Details

In the **spatstat** package, an object of class "colourmap" defines a mapping between data and colours.

The function default.colourmap provides a suitable default colour map for the values in x.

If x is a factor, default.colourmap(x) is a mapping from the factor levels to colours. If x is a logical vector, default.colourmap(x) is a mapping from the values TRUE and FALSE to colours. If x is a numeric vector, default.colourmap(x) is a mapping from numbers in the interval between the minimum and maximum values of x to colours.

The argument col may provide colour information in any of the following formats:

- A colour map (object of class "colourmap")
- A palette function (a function(n) or function(n, ...) which returns a vector of colour values, such as rainbow
- A vector of integers between 1 and 8 indexing the standard colour palette
- A vector of character strings giving common names of colours (e.g. "red")
- A vector of character strings giving hexadecimal codes for colours (e.g. produced by rgb or rainbow)
- NULL.

Value

A colour map (object of class "colourmap").

Scrambling the colours

The argument scramble.cols=TRUE will cause the sequence of colours in the colour map to be randomly permuted. This is a useful trick when it is desired that adjacent colours in the sequence should be easily distinguishable. The return value will be the randomised colour map.

Randomisation implies that the colour map will be different each time the command is executed. For reproducible results, set the random generator seed using set.seed.

Author(s)

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

See Also

colourmap

Examples

```
default.colourmap(letters[1:4])
default.colourmap(factor(letters[1:3], levels=letters[1:4]))
default.colourmap(TRUE)
default.colourmap(NULL)
```

---

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,
              quasi=FALSE, ..., eps=NULL, verbose=FALSE)
```

Arguments

X	The observed data point pattern. An object of class "ppp" or in a format recognised by as.ppp()
nd	Optional. Integer, or integer vector of length 2, specifying an nd * nd or nd[1] * nd[2] rectangular array of dummy points.
random	Logical value. If TRUE, the dummy points are generated randomly.
quasi	Logical value. If TRUE, the dummy points are generated by a quasirandom sequence.
ntile	Optional. Integer or pair of integers specifying the number of rows and columns of tiles used in the counting rule.
npix	Optional. Integer or pair of integers specifying the number of rows and columns of pixels used in computing approximate areas.
...	Ignored.
eps	Optional. Grid spacing. A positive number, or a vector of two positive numbers, giving the horizontal and vertical spacing, respectively, of the grid of dummy points. Incompatible with nd.
verbose	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 and quasi are both false (the default), then the function creates dummy points in a regular nd[1] by nd[1] rectangular grid. If random is true and quasi is false, 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. If quasi is true, a quasirandom pattern of nd[1] * nd[2] points is generated. In all cases, 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 is computed by the undocumented internal function default.n.tiling, using information about the data pattern X, and other arguments and settings. The default value of nd is always greater than or equal to spatstat.options("ndummy.min") and greater than or equal to 10 * ceiling(2 * sqrt(npoints(X))/10), and satisfies some other constraints. The default is designed so that model-fitting is relatively fast and stable, rather than highly accurate.

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

See Also

quad.object, quadscheme, corners, gridcentres, stratrand, spokes

Examples

```
P <- simdat
D <- default.dummy(P, 100)
plot(D)
Q <- quadscheme(P, D, "grid")
if(interactive()) {plot(union.quad(Q))}
```

---

default.image.colours *Default Colours for Images in Spatstat*

Description

Extract or change the default colours for images in **spatstat**.

Usage

```
default.image.colours()
reset.default.image.colours(col = NULL)
```

Arguments

col	A vector of colour values.

Details

These functions extract and change the current default colours used for plotting colour images in the **spatstat** family of packages, in particular by the functions plot.im and plot.linim.

The default colour values are a vector of character strings which can be interpreted as colours. In any particular instance of plot.im or plot.linim, the default colours are interpolated to obtain a vector of colour values of the required length (usually 256, controlled by the argument ncolours to the plot command).

default.image.colours() returns the current default colours. reset.default.image.colours(col) sets the default colours to be the vector col. reset.default.image.colours() or reset.default.image.colours(NULL) resets the factory default, which is row 29 of the Kovesi uniform perceptual contrast table described in Kovesi.

Value

A character vector of values which can be interpreted as colours.

Author(s)

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

See Also

plot.im

Examples

```
a <- default.image.colours()
length(a)
```

---

default.symbolmap *Default Symbol Map for Plotting a Spatial Pattern*

Description

Determines the symbol map for plotting a spatial pattern, when one is not supplied by the user.

Usage

```
default.symbolmap(x, ...)
```

Arguments

x	A spatial object in the **spatstat** package, such as a point pattern (class "ppp").
...	Additional arguments passed to methods.

Details

In the **spatstat** package, an object of class "symbolmap" defines a mapping between data and graphical symbols.

If a plot command plot(x, ...) has been issued, and if the arguments were not sufficient to determine the symbol map that should be used, then default.symbolmap(x, ...) will be executed to determine the default symbol map.

The function default.symbolmap is generic, with a method for point patterns (class "ppp") and possibly for other classes.

Value

A symbol map (object of class "symbolmap").

Author(s)

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

See Also

default.symbolmap.ppp

---

default.symbolmap.ppp *Default Symbol Map for Point Pattern*

Description

Determines a symbol map for plotting the spatial point pattern x.

Usage

```
## S3 method for class 'ppp'
default.symbolmap(x, ...,
      chars = NULL, cols = NULL,
      fixsize = FALSE,
      maxsize = NULL, meansize = NULL, markscale = NULL,
      minsize = NULL, zerosize = NULL, transform = NULL,
      scramble.cols = FALSE)
```

Arguments

x	A spatial point pattern (object of class "ppp").
...	extra graphical parameters, passed to symbolmap (and ultimately to points and/or symbols).
chars	the plotting character(s) used to plot points. Either a single character, an integer, or a vector of single characters or integers. Ignored if symap is given.
cols	the colour(s) used to plot points. Either an integer index from 1 to 8 (indexing the standard colour palette), a character string giving the name of a colour, a string giving the hexadecimal representation of a colour, or a vector of such integers or strings. See the section on *Colour Specification* in the help for par. Alternatively cols may be a function, as described in Details below.
fixsize	Logical value specifying whether the symbols should all have the same physical size on the plot. Default is FALSE.
maxsize	*Maximum* physical size of the circles/squares plotted when x is a marked point pattern with numerical marks. Incompatible with meansize and markscale.
meansize	*Average* physical size of the circles/squares plotted when x is a marked point pattern with numerical marks. Incompatible with maxsize and markscale.
markscale	physical scale factor determining the sizes of the circles/squares plotted when x is a marked point pattern with numerical marks. Mark value will be multiplied by markscale to determine physical size. Incompatible with maxsize and meansize.
minsize	*Minimum* physical size of the circles/squares plotted when x is a marked point pattern with numerical marks. Incompatible with zerosize.
zerosize	Physical size of the circle/square representing a mark value of zero, when x is a marked point pattern with numerical marks. Incompatible with minsize. Defaults to zero.
transform	Experimental. A function that will be applied to the mark values before the symbol map scale is determined.
scramble.cols	Logical value specifying whether to randomly permute the sequence cols. Applies only when x is a multitype point pattern. This is a useful trick when it is desired that adjacent colours should be easily distinguishable.

Details

This algorithm determines a symbol map that can be used to represent the points of x graphically. It serves as the default symbol map for the plot method plot.ppp.

Users can modify the behaviour of plot.ppp by saving the symbol map produced by default.symbolmap.ppp, modifying the symbol map using update.symbolmap or other tools, and passing the modified symbol map to plot.ppp as the argument symap.

The default representation depends on the marks of the points, as follows.

unmarked point pattern: If the point pattern does not have marks, then every point will be represented by the same plot symbol.

multitype point pattern: If marks(x) is a factor, then each level of the factor is represented by a different plot character.

continuous marks: If `marks(x)` is a numeric vector, each point is represented by a circle with *diameter* proportional to the mark (if the value is positive) or a square with *side length* proportional to the absolute value of the mark (if the value is negative).

other kinds of marks: If `marks(x)` is neither numeric nor a factor, then each possible mark will be represented by a different plotting character. The default is to represent the ith smallest mark value by `points(..., pch=i)`.

The following arguments can be used to modify how the points are plotted:

- If `fixsize=TRUE`, or if the graphics parameter `size` is given and is a single value, then numerical marks will be rendered as symbols of the same physical size
- 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(marks(x))`, and then the ith 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. A colour can be specified either as an integer from 1 to 8 (indexing the standard colour palette), a character string giving the name of a colour, or a character string giving the hexadecimal representation of a colour.
  - For an unmarked point pattern, `cols` should be a single colour value.
  - For a multitype point pattern, `cols` should be a vector of colour values, of the same length as `levels(marks(x))`: that is, there is one colour for each possible mark value. The ith level or type will be plotted using colour `cols[i]`.
  - Alternatively for a multitype point pattern, `cols` can be a `function(n)` or `function(n, ...)` which returns a vector of `n` colour values. This function will be invoked to obtain a vector of colours, with one colour for each possible mark value.
  - For a point pattern with continuous marks, `cols` should be a vector of colour values, of any length: the range of mark values will be mapped to the specified colours.
  - Alternatively, for any kind of data, `cols` can be a colour map (object of class `"colourmap"`) created by `colourmap`. The colour map will be applied to the mark values to determine the colours plotted.
  - For a multitype point pattern only, if `scramble.cols=TRUE`, the colours associated with the different types of points will be randomly permuted. This is a useful trick if it is desired that adjacent colours should be easily distinguishable.
- If `cols` is absent, the colours used to plot the points may be determined by the extra arguments `fg` and `bg` for foreground (edge) and background (fill) colours. (These parameters are not recommended for plotting multitype point patterns, due to quirks of the graphics system.)
- The default colour for the points is a semi-transparent grey, if this is supported by the plot device. This behaviour can be suppressed (so that the default colour is non-transparent) by setting `spatstat.options(transparent=FALSE)`.
- The arguments `maxsize`, `meansize` and `markscale` are incompatible with each other (and incompatible with `symap`). The arguments `minsize` and `zerosize` are incompatible with

each other (and incompatible with symap). Together, these arguments control the physical size of the circles and squares which represent the marks in a point pattern with continuous marks. The size of a circle is defined as its *diameter*; the size of a square is its side length. If markscale is given, then a mark value of m is plotted as a circle of diameter m * markscale + zerosize (if m is positive) or a square of side abs(m) * markscale + zerosize (if m is negative). If maxsize is given, then the largest mark in absolute value, mmax=max(abs(marks(x))), will be scaled to have physical size maxsize. If meansize is given, then the average absolute mark value, mmean=mean(abs(marks(x))), will be scaled to have physical size meansize. If minsize is given, then the minimum mark value, mmean=mean(abs(marks(x))), will be scaled to have physical size minsize.

- The user can set the default values of these plotting parameters using spatstat.options("par.points").

Additionally the user can specify any of the graphics parameters recognised by symbolmap, including shape, size, pch, cex, cols, col, fg, bg, lwd, lty, etch, direction,headlength,headangle,arrowtype.

Value

A symbol map (object of class "symbolmap") or a list of symbol maps, one for each column of marks.

Scrambling the colours

The argument scramble.cols=TRUE will cause the sequence of colours in the colour map (if any) to be randomly permuted. This is a useful trick when it is desired that adjacent colours in the sequence should be easily distinguishable.

Randomisation implies that the return value (the symbol map) will be different each time the command is executed. For reproducible results, set the random generator seed using set.seed.

Author(s)

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

See Also

plot.ppp

default.symbolmap

symbolmap

Examples

```
default.symbolmap(longleaf)
default.symbolmap(lansing)
```

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 based on X is computed; see dirichlet. This tessellation is extended to cover the entire two-dimensional plane. 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 to make the Delaunay triangulation. 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@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

tess, dirichlet, convexhull.xy, ppp, delaunayDistance, delaunayNetwork.

Examples

```
X <- runifrect(42)
plot(delaunay(X))
plot(X, add=TRUE)
```

delaunayDistance *Distance on Delaunay Triangulation*

Description

Computes the graph distance in the Delaunay triangulation of a point pattern.

Usage

```
delaunayDistance(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 based on X is computed; see dirichlet. This tessellation is extended to cover the entire two-dimensional plane. 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 to make the Delaunay triangulation.

The *graph distance* in the Delaunay triangulation between two points X[i] and X[j] is the minimum number of edges of the Delaunay triangulation that must be traversed to go from X[i] to X[j]. Two points have graph distance 1 if they are immediate neighbours.

This command returns a matrix D such that D[i,j] is the graph distance between X[i] and X[j].

Value

A symmetric square matrix with non-negative integer entries.

Definition of neighbours

Note that dirichlet(X) restricts the Dirichlet tessellation to the window containing X, whereas dirichletDistance uses the Dirichlet tessellation over the entire two-dimensional plane. Some points may be Delaunay neighbours according to delaunayDistance(X) although the corresponding tiles of dirichlet(X) do not share a boundary inside Window(X).

Author(s)

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

See Also

delaunay, delaunayNetwork.

Examples

```
X <- runifrect(20)
M <- delaunayDistance(X)
plot(delaunay(X), lty=3)
text(X, labels=M[1, ], cex=2)
```

`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

$$\Delta(A,B) = \left[\frac{1}{|W|} \int_W |\min(c, d(x,A)) - \min(c, d(x,B))|^p \, \mathrm{d}x \right]^{1/p}$$

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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 <- runifrect(20)
Y <- runifrect(10)
deltametric(X, Y, p=1,c=0.1)
```

---

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), "boxx" (multi-dimensional boxes) and "linnet" (linear networks).

Value

The numerical value of the diameter of the object.

Author(s)

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

See Also

diameter.owin, diameter.box3, diameter.boxx,

---

diameter.box3 *Geometrical Calculations for Three-Dimensional Box*

Description

Calculates the volume, diameter, shortest side, side lengths, or eroded volume of a three-dimensional box.

Usage

```
## S3 method for class 'box3'
diameter(x)

## S3 method for class 'box3'
volume(x)

shortside(x)
sidelengths(x)
eroded.volumes(x, r)

## S3 method for class 'box3'
shortside(x)

## S3 method for class 'box3'
sidelengths(x)

## S3 method for class 'box3'
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. `sidelengths.box3` returns all 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 `sidelengths.box3`, a vector of three numbers. For `eroded.volumes`, a numeric vector of the same length as `r`.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

`as.box3`

Examples

```
X <- box3(c(0,10),c(0,10),c(0,5))
diameter(X)
volume(X)
sidelengths(X)
shortside(X)
hd <- shortside(X)/2
eroded.volumes(X, seq(0,hd, length=10))
```

`diameter.boxx` *Geometrical Calculations for Multi-Dimensional Box*

Description

Calculates the volume, diameter, shortest side, side lengths, 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'
sidelengths(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. sidelengths.boxx returns the lengths of each side 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 sidelengths.boxx, a numeric vector of length equal to the number of spatial dimensions. For eroded.volumes.boxx, a numeric vector of the same length as r.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

boxx

Examples

```
X <- boxx(c(0,10),c(0,10),c(0,5),c(0,2))
diameter(X)
volume(X)
shortside(X)
sidelengths(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 window.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

area.owin, perimeter, edges, owin, as.owin

Examples

```
w <- owin(c(0,1),c(0,1))
diameter(w)
# returns sqrt(2)
diameter(letterR)
```

dilated.areas *Areas of Morphological Dilations*

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.
...	Arguments passed to distmap to control the pixel resolution, if exact=FALSE.
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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

owin, as.owin, dilation, eroded.areas

Examples

```
X <- runifrect(10)
a <- dilated.areas(X, c(0.1,0.2), W=square(1), exact=TRUE)
```

`dilation` *Morphological Dilation*

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

`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 dilation.
`...`	extra arguments passed to `as.mask` controlling the pixel resolution, if the pixel approximation is used; or passed to `disc` if the polygonal approximation is used.
`polygonal`	Logical flag indicating whether to compute a polygonal approximation to the dilation (`polygonal=TRUE`) or a pixel grid approximation (`polygonal=FALSE`).
`tight`	Logical flag indicating whether the bounding frame of the window should be taken as the smallest rectangle enclosing the dilated region (`tight=TRUE`), or should be the dilation of the bounding frame of `w` (`tight=FALSE`).

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 `"\dots"` 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@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>

See Also

erosion for the opposite operation.

dilationAny for morphological dilation using any shape.

owin, as.owin

Examples

```
plot(dilation(redwood, 0.05))
points(redwood)

plot(dilation(letterR, 0.2))
plot(letterR, add=TRUE, lwd=2, border="red")

X <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
plot(dilation(X, 0.1))
plot(X, add=TRUE, col="red")
```

---

dirichlet *Dirichlet Tessellation of Point Pattern*

Description

Computes the Dirichlet tessellation of a spatial point pattern. Also known as the Voronoi or Thiessen tessellation.

Usage

```
dirichlet(X)
```

Arguments

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

Details

In a spatial point pattern X, the Dirichlet 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.

The Dirichlet tessellation is also known as the Voronoi or Thiessen tessellation.

This function computes the Dirichlet tessellation (within the original window of X) using the function deldir in the package **deldir**.

To ensure that there is a one-to-one correspondence between the points of X and the tiles of dirichlet(X), duplicated points in X should first be removed by X <- unique(X, rule="deldir").

The tiles of the tessellation will be computed as polygons if the original window is a rectangle or a polygon. Otherwise the tiles will be computed as binary masks.

Value

A tessellation (object of class "tess").

Author(s)

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

See Also

tess, delaunay, ppp, dirichletVertices.

For the Dirichlet tessellation on a linear network, see lineardirichlet.

Examples

```
X <- runifrect(42)
plot(dirichlet(X))
plot(X, add=TRUE)
```

---

dirichletAreas *Compute Areas of Tiles in Dirichlet Tessellation*

Description

Calculates the area of each tile in the Dirichlet-Voronoi tessellation of a point pattern.

Usage

```
dirichletAreas(X)
```

Arguments

X	Point pattern (object of class "ppp").

Details

This is an efficient algorithm to calculate the areas of the tiles in the Dirichlet-Voronoi tessellation.

If the window of X is a binary pixel mask, the tile areas are computed by counting pixels. Otherwise the areas are computed exactly using analytic geometry.

If any points of X are duplicated, the duplicates will have tile area zero.

Value

Numeric vector with one entry for each point of X.

Author(s)

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

Rolf Turner <rolfturner@posteo.net>

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

See Also

dirichlet, dirichletVertices

Examples

```
aa <- dirichletAreas(cells)
```

---

dirichletVertices *Vertices and Edges of Dirichlet Tessellation*

Description

Computes the Dirichlet-Voronoi tessellation of a point pattern and extracts the vertices or edges of the tiles.

Usage

```
dirichletVertices(X)

dirichletEdges(X, clip=TRUE)
```

Arguments

X	Point pattern (object of class "ppp").
clip	Logical value specifying whether to clip the tile edges to the window. See Details.

Details

These function compute the Dirichlet-Voronoi tessellation of X (see dirichlet) and extract the vertices or edges of the tiles of the tessellation.

The Dirichlet vertices are the spatial locations which are locally farthest away from X, that is, where the distance function of X reaches a local maximum.

The Dirichlet edges are the dividing lines equally distant between a pair of points of X.

The Dirichlet tessellation of X is computed using dirichlet. The vertices or edges of all tiles of the tessellation are extracted.

For dirichletVertices, any vertex which lies on the boundary of the window of X is deleted. The remaining vertices are returned, as a point pattern, without duplicated entries.

For dirichletEdges, the edges are initially computed inside the rectangle Frame(X). Then if clip=TRUE (the default), these edges are intersected with Window(X), which may cause an edge to be broken into several pieces.

Value

dirichletVertices returns a point pattern (object of class "ppp") in the same window as X.

dirichletEdges returns a line segment pattern (object of class "psp").

Author(s)

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

See Also

dirichlet, dirichletAreas

Examples

```
plot(dirichlet(cells))

plot(dirichletVertices(cells), add=TRUE)

ed <- dirichletEdges(cells)
```

---

dirichletWeights *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

```
dirichletWeights(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 deldir. 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 Window(X)).

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

quad.object, gridweights

Examples

```
Q <- quadscheme(runifrect(10))
X <- as.ppp(Q) # data and dummy points together
w <- dirichletWeights(X, exact=FALSE)
```

`disc` *Circular Window*

Description

Creates a circular window

Usage

```
disc(radius=1, centre=c(0,0), ...,
     mask=FALSE, npoly=128, delta=NULL,
     type=c("inscribed", "circumscribed", "approx"),
     metric=NULL)
```

Arguments

`radius`	Radius of the circle.
`centre`	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 `mask=FALSE`. Incompatible with `delta`.
`delta`	Tolerance of polygonal approximation: the length of arc that will be replaced by one edge of the polygon. Incompatible with `npoly`.
`type`	Optional. Character string (partially matched) indicating whether the polygonal approximation should be an inscribed polygon (`type="inscribed"`, the default), a circumscribed polygon (`type="circumscribed"`) or a polygon which is between the inscribed and circumscribed polygons (`type="approx"`).
`...`	Arguments passed to `as.mask` determining the pixel resolution, if `mask=TRUE`.
`metric`	Optional. A distance metric (object of class `"metric"`). The disc with respect to this metric will be computed.

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`.

The argument `radius` must be a single positive number. The argument `centre` specifies the disc centre: it can be either a numeric vector of length 2 giving the coordinates, or a `list(x,y)` giving the coordinates of exactly one point, or a point pattern (object of class `"ppp"`) containing exactly one point.

If `type="inscribed"` (the default), the polygon is inscribed in the perfect circle of radius `radius`, that is, all vertices of the polygon are points on the circle, and the circle contains the polygon.

If type="circumscribed", the polygon is circumscribed around the circle, that is, the polygon contains the circle, and all edges of the polygon are tangent to the circle. If type="approx", the polygon is constructed to lie between the inscribed and circumscribed polygons, giving a better approximation to the circle.

If the argument metric is given, it should be a distance metric (object of class "metric"). The disc with respect to this metric will be computed.

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

See Also

ellipse, discs, 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(disc())/pi
area(disc(mask=TRUE))/pi
# inscribed and circumscribed
plot(disc(npoly=256), main="Inscribed and circumscribed", border="green")
plot(disc(npoly=7, type="i"), add=TRUE, border="red")
plot(disc(npoly=7, type="c"), add=TRUE, border="blue")
plot(disc(npoly=7, type="a"), add=TRUE, border="grey")
```

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

See Also

`owin`, `disc`

Examples

```
X <- unmark(demopat)[1:3]
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. Optionally discretise the point locations as well, by moving them to the nearest pixel centres.

Usage

```
discretise(X, eps = NULL, dimyx = NULL, xy = NULL, move.points=FALSE,
           rule.eps=c("adjust.eps", "grow.frame", "shrink.frame"))
```

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
`move.points`	Logical value specifying whether the points should also be discretised by moving each point to the nearest pixel centre.
`rule.eps`	Argument passed to `as.mask` controlling the discretisation.

Details

This function modifies the point pattern `X` by converting its observation window `Window(X)` to a binary pixel image (a window of type `"mask"`). It ensures that no points of `X` are deleted by the discretisation. If `move.points=TRUE`, the point coordinates are also discretised.

The window is first discretised using `as.mask`. Next,

- If `move.points=TRUE`, each point of `X` is moved to the centre of the nearest pixel inside the discretised window.
- If `move.points=FALSE` (the default), the point coordinates are unchanged. 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`, `xy` and `rule.eps` 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`, `xy` and `rule.eps`, and the process of converting a window to one of type `mask`.

Value

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

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

See Also

as.mask

Examples

```
X <- demopat
plot(X, main="original pattern")
Y <- discretise(X, dimyx=50)
plot(Y, main="discretise(X)")
stopifnot(npoints(X) == npoints(Y))

# what happens if we just convert the window to a mask?
W <- Window(X)
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(npoints(X) - npoints(XM), "points of X lie outside M\n"))
```

---

discs *Union of Discs*

Description

Make a spatial region composed of discs with given centres and radii.

Usage

```
discs(centres, radii = marks(centres)/2, ...,
      separate = FALSE, mask = FALSE, trim = TRUE,
      delta = NULL, npoly=NULL)
```

Arguments

centres	Point pattern giving the locations of centres for the discs.
radii	Vector of radii for each disc, or a single number giving a common radius. (Notice that the default assumes that the marks of X are *diameters*.)
...	Optional arguments passed to as.mask to determine the pixel resolution, if mask=TRUE.
separate	Logical. If TRUE, the result is a list containing each disc as a separate entry. If FALSE (the default), the result is a window obtained by forming the union of the discs.
mask	Logical. If TRUE, the result is a binary mask window. If FALSE, the result is a polygonal window. Applies only when separate=FALSE.
trim	Logical value indicating whether to restrict the result to the original window of the centres. Applies only when separate=FALSE.
delta	Argument passed to disc to determine the tolerance for the polygonal approximation of each disc. Applies only when mask=FALSE. Incompatible with npoly.
npoly	Argument passed to disc to determine the number of edges in the polygonal approximation of each disc. Applies only when mask=FALSE. Incompatible with delta.

Details

This command is typically applied to a marked point pattern dataset X in which the marks represent the sizes of objects. The result is a spatial region representing the space occupied by the objects.

If the marks of X represent the diameters of circular objects, then the result of discs(X) is a spatial region constructed by taking discs, of the specified diameters, centred at the points of X, and forming the union of these discs. If the marks of X represent the areas of objects, one could take discs(X, sqrt(marks(X)/pi)) to produce discs of equivalent area.

A fast algorithm is used to compute the result as a binary mask, when mask=TRUE. This option is recommended unless polygons are really necessary.

If mask=FALSE, the discs will be constructed as polygons by the function disc. To avoid computational problems, by default, the discs will all be constructed using the same physical tolerance value delta passed to disc. The default is such that the smallest disc will be approximated by a 16-sided polygon. (The argument npoly should not normally be used, to avoid computational problems arising with small radii.)

Value

If separate=FALSE, a window (object of class "owin").

If separate=TRUE, a list of windows.

Author(s)

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

See Also

disc, union.owin

Examples

```
plot(discs(anemones, mask=TRUE, eps=0.5))
```

---

distbdry *Distance To Boundary*

Description

Compute the distance from each point in a point pattern to the boundary of the spatial domain.

Usage

```
distbdry(X)

## S3 method for class 'ppp'
distbdry(X)

## S3 method for class 'pp3'
distbdry(X)
```

Arguments

X	A point pattern (class "ppp" or "pp3" or other classes).

Details

For each point X[i] this function computes the shortest distance to the boundary of the observation window or domain of the point pattern.

The function distbdry is generic, with methods for class "ppp", "pp3" and other classes.

For a two-dimensional spatial point pattern X of class "ppp", distbdry(X) is equivalent to bdist.points(X).

Value

A numeric vector with one entry for each point of X.

Author(s)

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

See Also

bdist.points

Examples

```
distbdry(cells)[1:10]
```

---

distfun *Distance Map as a Function*

Description

Compute the distance function of an object, and return it as a function.

Usage

```
  distfun(X, ...)

  ## S3 method for class 'ppp'
distfun(X, ..., k=1, undef=Inf)

  ## S3 method for class 'psp'
distfun(X, ...)

  ## S3 method for class 'owin'
distfun(X, ..., invert=FALSE, signed=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.
k	An integer. The distance to the kth nearest point will be computed.
undef	The value that should be returned if the distance is undefined (that is, if X contains fewer than k points).
invert	Logical value. If TRUE, compute the distance transform of the complement of X.
signed	Logical value. If TRUE, compute the signed distance transform which is negative inside X and positive outside 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. Alternatively x can be a point pattern (object of

class "ppp" or "lpp") of locations at which the distance function should be computed (and then y should be missing).

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.

The distance values returned by f <- distfun(X); d <- f(x) are computed using coordinate geometry; they are more accurate, but slower to compute, than the distance values returned by Z <- distmap(X); d <- Z[x] which are computed using a fast recursive algorithm.

The result of f <- distfun(X) also belongs to the class "funxy" and to the special class "distfun". It can be printed and plotted immediately as shown in the Examples.

A distfun object can be converted to a pixel image using as.im.

Value

A function with arguments x,y. The function belongs to the class "distfun" which has methods for print and summary, and for geometric operations like shift. It also belongs to the class "funxy" which has methods for plot, contour and persp.

Distance values

The values returned by the distance function f <- distfun(X) are distances, expressed as multiples of the unit of length of the spatial coordinates in X. The unit of length is given by unitname(X).

Note that, if the unit of length in X is a composite expression such as '2 microns', then the values of f are expressed as multiples of 2 microns, rather than being expressed in microns.

Domain of the function

The domain of the distance function is unbounded, that is, the distance function can be evaluated at any spatial location.

However, when the distance function is plotted by plot, contour or persp, the function domain is assumed to be the rectangular frame surrounding the original object X. To generate a plot of the distance function on a different region, use the argument W in the plot command, as explained in the help for plot.funxy.

Author(s)

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

See Also

distmap, summary.distfun, methods.distfun, methods.funxy, plot.funxy

Examples

```
f <- distfun(letterR)
f
plot(f)
f(0.2, 0.3)
```

```
plot(distfun(letterR, invert=TRUE), eps=0.1)
plot(distfun(letterR, signed=TRUE), eps=0.1, col=beachcolourmap)

d <- distfun(cells)
d2 <- distfun(cells, k=2)
d(0.5, 0.5)
d2(0.5, 0.5)
domain(d)
summary(d)

z <- d(japanesepines)
```

---

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) as well as other classes.

Value

A pixel image (object of class "im") whose grey scale values are the values of the distance map.

Distance values

The pixel values in the image distmap(X) are distances, expressed as multiples of the unit of length of the spatial coordinates in X. The unit of length is given by unitname(X).

Note that, if the unit of length in X is a composite expression such as '2 microns', then the values in distmap(X) are expressed as multiples of 2 microns, rather than being expressed in microns.

Author(s)

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

See Also

distmap.ppp, distmap.psp, distmap.owin, distfun

Examples

```
U <- distmap(cells)
V <- distmap(letterR)
if(interactive()) {
plot(U)
plot(V)
}
```

---

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,
        signed=FALSE, connect=8, metric=NULL)
```

Arguments

X	A window (object of class "owin").
...	Arguments passed to as.mask to control pixel resolution.
discretise	Logical flag controlling the choice of algorithm when X is a polygonal window. See Details.
invert	Logical value. If TRUE, compute the distance transform of the complement of the window.
signed	Logical value. If TRUE, compute the signed distance transform which is negative inside X and positive outside X.
connect	Neighbourhood connectivity for the discrete distance transform algorithm. Either 8 or 24.
metric	Optional. A distance metric (object of class "metric", see metric.object) which will be used to compute the distances.

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.

Distance values

The pixel values in the image `distmap(X)` are distances, expressed as multiples of the unit of length of the spatial coordinates in `X`. The unit of length is given by `unitname(X)`.

Note that, if the unit of length in `X` is a composite expression such as '2 microns', then the values in `distmap(X)` are expressed as multiples of 2 microns, rather than being expressed in microns.

Author(s)

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

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

```
U <- distmap(letterR)
if(human <- interactive()) {
  plot(U, main="distmap(letterR)")
  plot(attr(U, "bdry"))
}
V <- distmap(letterR, signed=TRUE)
if(human)
  plot(V, main="signed distance", col=beachcolourmap)
```

`distmap.ppp` *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, ..., clip=FALSE,
                squared=FALSE, extras=TRUE, metric=NULL)
```

Arguments

`X`	A point pattern (object of class `"ppp"`).
`...`	Arguments passed to `as.mask` to control pixel resolution.
`clip`	Logical value specifying whether the resulting pixel image should be clipped to the window of `X`.
`squared`	Logical value specifying whether to return the squared distance values.
`extras`	Logical value specifying whether to return extra data (point indices and boundary distances). See Details.
`metric`	Optional. A distance metric (object of class `"metric"`, see `metric.object`) which will be used to compute the distances.

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`. If `squared=TRUE`, the squared distances are returned, which is a faster computation.

If `clip=FALSE` (the default), the resulting pixel values are defined at every pixel in the rectangle `Frame(X)`. If `clip=TRUE`, the pixel values are defined only inside `Window(X)`, and are `NA` outside this window. Computation is faster when `clip=FALSE`.

If extras=TRUE (the default), 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 boundary of the window containing X. 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.

Distance values

The pixel values in the image distmap(X) are distances, expressed as multiples of the unit of length of the spatial coordinates in X. The unit of length is given by unitname(X).

Note that, if the unit of length in X is a composite expression such as '2 microns', then the values in distmap(X) are expressed as multiples of 2 microns, rather than being expressed in microns.

Author(s)

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

See Also

Generic function distmap and other methods distmap.psp, distmap.owin.

Generic function distfun.

Nearest neighbour distance nncross.

unitname and rescale to control the unit of length.

Examples

```
U <- distmap(cells)
if(interactive()) {
  plot(U)
  plot(attr(U, "bdry"))
  plot(attr(U, "index"))
}
```

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, ..., extras=TRUE, clip=FALSE, metric=NULL)
```

Arguments

X	A line segment pattern (object of class "psp").
...	Arguments passed to as.mask to control pixel resolution.
extras	Logical value specifying whether to compute the additional attributes "index" and "bdry" described in Details.
clip	Logical value specifying whether the resulting pixel image should be clipped to the window of X.
metric	Optional. A distance metric (object of class "metric", see metric.object) which will be used to compute the distances.

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.

The result is a pixel image. If clip=FALSE (the default), the pixel values are defined at every pixel in the rectangle Frame(X). If clip=TRUE, the pixel values are defined only inside Window(X), and are NA outside this window. Computation is faster when clip=FALSE.

Additionally, if extras=TRUE, the return value has two attributes, "index" and "bdry", which are also pixel images. The pixels values of "bdry" give the distance from each pixel to the boundary of the window of X (and are zero outside this window). The pixel values of "index" are integers identifying which line segment of X is closest. If clip=FALSE (the default), these images are defined at every pixel in Frame(X); if clip=TRUE, they are clipped to the window of X. Computation is faster when extras=FALSE.

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.

Distance values

The pixel values in the image distmap(X) are distances, expressed as multiples of the unit of length of the spatial coordinates in X. The unit of length is given by unitname(X).

Note that, if the unit of length in X is a composite expression such as '2 microns', then the values in distmap(X) are expressed as multiples of 2 microns, rather than being expressed in microns.

Author(s)

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

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)
```

---

domain *Extract the Domain of any Spatial Object*

Description

Given a spatial object such as a point pattern, in any number of dimensions, this function extracts the spatial domain in which the object is defined.

Usage

```
domain(X, ...)

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

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

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

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

   ## S3 method for class 'pp3'
domain(X, ...)

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

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

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

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

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

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

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

Arguments

X	A spatial object such as a point pattern (in any number of dimensions), line segment pattern or pixel image.
...	Extra arguments. They are ignored by all the methods listed here.

Details

The function domain is generic.

For a spatial object X in any number of dimensions, domain(X) extracts the spatial domain in which X is defined.

For a two-dimensional object X, typically domain(X) is the same as Window(X).

Exceptions occur for methods related to linear networks.

Value

A spatial object representing the domain of X. Typically a window (object of class "owin"), a three-dimensional box ("box3"), a multidimensional box ("boxx") or a linear network ("linnet").

Author(s)

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

See Also

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

Examples

```
domain(redwood)
domain(bei.extra$elev)
domain(osteo$pts[[1]])
```

---

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, ..., rule=c("spatstat", "deldir", "unmark"))

 ## S3 method for class 'ppx'
duplicated(x, ...)

 ## S3 method for class 'ppp'
anyDuplicated(x, ..., rule=c("spatstat", "deldir", "unmark"))

 ## S3 method for class 'ppx'
anyDuplicated(x, ...)
```

Arguments

x	A spatial point pattern (object of class "ppp" or "ppx").
...	Ignored.
rule	Character string. The rule for determining duplicated points.

Details

These are methods for the generic functions duplicated and anyDuplicated for point pattern datasets (of class "ppp", see ppp.object, or class "ppx").

anyDuplicated(x) is a faster version of any(duplicated(x)).

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.

If rule="spatstat" (the default), two points are deemed identical if their coordinates are equal according to ==, *and* their marks are equal according to ==. This is the most stringent possible test. If rule="unmark", duplicated points are determined by testing equality of their coordinates only, using ==. If rule="deldir", duplicated points are determined by testing equality of their coordinates only, using the function duplicatedxy in the package **deldir**, which currently uses duplicated.data.frame. Setting rule="deldir" will ensure consistency with functions in the **deldir** package.

Value

duplicated(x) returns a logical vector of length equal to the number of points in x.

anyDuplicated(x) is a number equal to 0 if there are no duplicated points, and otherwise is equal to the index of the first duplicated point.

Author(s)

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

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)
duplicated(X, rule="deldir")
```

---

edges *Extract Boundary Edges of a Window.*

Description

Extracts the boundary edges of a window and returns them as a line segment pattern.

Usage

```
edges(x, ..., window = NULL, check = FALSE)
```

Arguments

x	A window (object of class "owin"), or data acceptable to as.owin, specifying the window whose boundary is to be extracted.
...	Ignored.
window	Window to contain the resulting line segments. Defaults to as.rectangle(x).
check	Logical. Whether to check the validity of the resulting segment pattern.

Details

The boundary edges of the window x will be extracted as a line segment pattern.

Value

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

Author(s)

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

See Also

perimeter for calculating the total length of the boundary.

Examples

```
edges(square(1))
edges(letterR)
```

---

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

iedge, jedge	Integer vectors, of equal length, specifying the edges.
nvert	Number of vertices in the network.
...	Ignored
check	Logical. Whether to check validity of input data.
friendly	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 kth 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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

edges2vees

Examples

```
i <- c(1, 2, 5, 5, 1, 4, 2)
j <- c(2, 3, 3, 1, 3, 2, 5)
edges2triangles(i, j)
```

edges2vees *List Dihedral Triples in a Graph*

Description

Given a list of edges between vertices, compile a list of all 'vees' or dihedral triples formed by these edges.

Usage

```
edges2vees(iedge, jedge, nvert=max(iedge, jedge), ...,
                check=TRUE)
```

Arguments

iedge, jedge	Integer vectors, of equal length, specifying the edges.
nvert	Number of vertices in the network.
...	Ignored
check	Logical. Whether to check validity of input data.

Details

Given a finite graph with nvert vertices and with edges specified by iedge, jedge, this low-level function finds all 'vees' or 'dihedral triples' in the graph, that is, all triples of vertices (i,j,k) where i and j are joined by an edge and i and k are joined by an edge.

The interpretation of iedge, jedge is that each successive pair of entries specifies an edge in the graph. The kth edge joins vertex iedge[k] to vertex jedge[k]. Entries of iedge and jedge must be integers from 1 to nvert.

Value

A 3-column matrix of integers, in which each row represents a triple of vertices, with the first vertex joined to the other two vertices.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

edges2triangles

Examples

```
i <- c(1, 2, 5, 5, 1, 4, 2)
j <- c(2, 3, 3, 1, 3, 2, 5)
edges2vees(i, j)
```

edit.hyperframe *Invoke Text Editor on Hyperframe*

Description

Invokes a text editor allowing the user to inspect and change entries in a hyperframe.

Usage

```
## S3 method for class 'hyperframe'
edit(name, ...)
```

Arguments

name	A hyperframe (object of class "hyperframe").
...	Other arguments passed to edit.data.frame.

Details

The function edit is generic. This function is the methods for objects of class "hyperframe".

The hyperframe name is converted to a data frame or array, and the text editor is invoked. The user can change entries in the columns of data, and create new columns of data.

Only the columns of atomic data (numbers, characters, factor values etc) can be edited.

Note that the original object name is not changed; the function returns the edited dataset.

Value

Another hyperframe.

Author(s)

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

Rolf Turner <rolfturner@posteo.net>

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

See Also

edit.data.frame, edit.ppp

Examples

```
if(interactive()) Z <- edit(flu)
```

edit.ppp *Invoke Text Editor on Spatial Data*

Description

Invokes a text editor allowing the user to inspect and change entries in a spatial dataset.

Usage

```
## S3 method for class 'ppp'
edit(name, ...)

## S3 method for class 'psp'
edit(name, ...)

## S3 method for class 'im'
edit(name, ...)
```

Arguments

name	A spatial dataset (object of class "ppp", "psp" or "im").
...	Other arguments passed to edit.data.frame.

Details

The function edit is generic. These functions are methods for spatial objects of class "ppp", "psp" and "im".

The spatial dataset name is converted to a data frame or array, and the text editor is invoked. The user can change the values of spatial coordinates or marks of the points in a point pattern, or the coordinates or marks of the segments in a segment pattern, or the pixel values in an image. The names of the columns of marks can also be edited.

If name is a pixel image, it is converted to a matrix and displayed in the same spatial orientation as if the image had been plotted.

Note that the original object name is not changed; the function returns the edited dataset.

Value

Object of the same kind as name containing the edited data.

Author(s)

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

Rolf Turner <rolfturner@posteo.net>

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

See Also

edit.data.frame, edit.hyperframe

Examples

```
if(interactive()) Z <- edit(cells)
```

---

ellipse *Elliptical Window.*

Description

Create an elliptical window.

Usage

```
ellipse(a, b, centre=c(0,0), phi=0, ..., mask=FALSE, npoly = 128)
```

Arguments

a, b	The half-lengths of the axes of the ellipse.
centre	The centre of the ellipse.
phi	The (anti-clockwise) angle through which the ellipse should be rotated (about its centre) starting from an orientation in which the axis of half-length a is horizontal.
mask	Logical value controlling the type of approximation to a perfect ellipse. See Details.
...	Arguments passed to as.mask to determine the pixel resolution, if mask is TRUE.
npoly	The number of edges in the polygonal approximation to the ellipse.

Details

This command creates a window object representing an ellipse with the given centre and axes.

By default, the ellipse is approximated by a polygon with npoly edges.

If mask=TRUE, then the ellipse is approximated by a binary pixel mask. The resolution of the mask is controlled by the arguments ... which are passed to as.mask.

The arguments a and b must be single positive numbers. The argument centre specifies the ellipse centre: it can be either a numeric vector of length 2 giving the coordinates, or a list(x,y) giving the coordinates of exactly one point, or a point pattern (object of class "ppp") containing exactly one point.

Value

An object of class owin (either of type "polygonal" or of type "mask") specifying an elliptical window.

Author(s)

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

See Also

disc, owin.object, owin, as.mask

Examples

```
W <- ellipse(a=5,b=2,centre=c(5,1),phi=pi/6)
plot(W,lwd=2,border="red")
WM <- ellipse(a=5,b=2,centre=c(5,1),phi=pi/6,mask=TRUE,dimyx=64)
plot(WM,add=TRUE,box=FALSE)
```

---

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

x	A line segment pattern (object of class "psp").
which	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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

`psp.object`, `ppp.object`, `marks.psp`, `summary.psp`, `midpoints.psp`, `lengths_psp`, `angles.psp`, `extrapolate.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)
```

---

eroded.areas *Areas of Morphological Erosions*

Description

Computes the areas of successive morphological erosions of a window.

Usage

```
eroded.areas(w, r, subset=NULL)
```

Arguments

`w`	A window.
`r`	Numeric vector of radii at which erosions will be performed.
`subset`	Optional window inside which the areas should be computed.

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 $x \in W$ 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.

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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 by a Disc*

Description

Perform morphological erosion of a window, a line segment pattern or a point pattern by a disc.

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

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 erosion.
shrink.frame	logical: if TRUE, erode the bounding rectangle as well.
...	extra arguments to as.mask controlling the pixel resolution, if pixel approximation is used.
strict	Logical flag determining the fate of boundary pixels, if pixel approximation is used. See details.
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 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 "\dots" 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@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>

See Also

dilation for the opposite operation.

erosionAny for morphological erosion using any shape.

owin, as.owin, eroded.areas

Examples

```
plot(letterR, main="erosion(letterR, 0.2)")
plot(erosion(letterR, 0.2), add=TRUE, col="red")
```

`erosionAny` *Morphological Erosion of Windows*

Description

Compute the morphological erosion of one spatial window by another.

Usage

```
erosionAny(A, B)

A %(-)% B
```

Arguments

`A`, `B`	Windows (objects of class `"owin"`).

Details

The operator `A %(-)% B` and function `erosionAny(A,B)` are synonymous: they both compute the morphological erosion of the window `A` by the window `B`.

The morphological erosion $A \ominus B$ of region A by region B is the spatial region consisting of all vectors z such that, when B is shifted by the vector z, the result is a subset of A.

Equivalently

$$A \ominus B = ((A^c \oplus (-B))^c$$

where $\oplus$ is the Minkowski sum, A^c denotes the set complement, and $(-B)$ is the reflection of B through the origin, consisting of all vectors $-b$ where b is a point in B.

If `B` is a disc of radius `r`, then `erosionAny(A, B)` is equivalent to `erosion(A, r)`. See `erosion`.

The algorithm currently computes the result as a polygonal window using the **polyclip** library. It will be quite slow if applied to binary mask windows.

Value

Another window (object of class `"owin"`).

Author(s)

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

See Also

`erosion`, `MinkowskiSum`

Examples

```
B <- square(c(-0.1, 0.1))
RminusB <- letterR %(-)% B
FR <- grow.rectangle(Frame(letterR), 0.3)
plot(FR, main="", type="n")
plot(letterR, add=TRUE, lwd=2, hatch=TRUE, box=FALSE)
plot(RminusB, add=TRUE, col="blue", box=FALSE)
plot(shift(B, vec=c(3.49, 2.98)),
     add=TRUE, border="red", lwd=2)
```

`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, warn=TRUE)
```

Arguments

`expr`	An expression.
`envir`	Optional. The environment in which to evaluate the expression, or a named list containing pixel images to be used in the expression.
`harmonize`	Logical. Whether to resolve inconsistencies between the pixel grids.
`warn`	Logical. Whether to issue a warning if the pixel grids were inconsistent.

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 harmonize=FALSE, images that are incompatible will cause an error. If harmonize=TRUE, images that have incompatible dimensions will be resampled so that they are compatible; if warn=TRUE, a warning will be issued.

Value

An image object of class "im".

Author(s)

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

See Also

im.apply for operations similar to apply, such as taking the sum of a list of images.

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)

## Use of 'envir': bei.extra is a list with components 'elev' and 'grad'
W <- eval.im(atan(grad) * 180/pi, bei.extra)
```

Extract.anylist *Extract or Replace Subset of a List of Things*

Description

Extract or replace a subset of a list of things.

Usage

```
  ## S3 method for class 'anylist'
x[i, ...]

  ## S3 replacement method for class 'anylist'
x[i] <- value
```

Arguments

x	An object of class "anylist" representing a list of things.
i	Subset index. Any valid subset index in the usual R sense.
value	Replacement value for the subset.
...	Ignored.

Details

These are the methods for extracting and replacing subsets for the class "anylist".

The argument x should be an object of class "anylist" representing a list of things. See anylist.

The method replaces a designated subset of x, and returns an object of class "anylist".

Value

Another object of class "anylist".

Author(s)

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

See Also

anylist, plot.anylist, summary.anylist

Examples

```
x <- anylist(A=runif(10), B=runif(10), C=runif(10))
x[1] <- list(A=rnorm(10))
```

Extract.hyperframe *Extract or Replace Subset of Hyperframe*

Description

Extract or replace a subset of a hyperframe.

Usage

```
  ## S3 method for class 'hyperframe'
x[i, j, drop, strip=drop, ...]
  ## S3 replacement method for class 'hyperframe'
x[i, j] <- value
  ## S3 method for class 'hyperframe'
x$name
  ## S3 replacement method for class 'hyperframe'
x$name <- value
  ## S3 method for class 'hyperframe'
x[[...]]
  ## S3 replacement method for class 'hyperframe'
x[[i, j]] <- value
```

Arguments

x	A hyperframe (object of class "hyperframe").
i, j	Row and column indices.
drop, strip	Logical values indicating what to do when the hyperframe has only one row or column. See Details.
...	Indices specifying elements to extract by [[.hyperframe. Ignored by [.hyperframe.
name	Name of a column of the hyperframe.
value	Replacement value for the subset. A hyperframe or (if the subset is a single column) a list or an atomic vector.

Details

These functions extract a designated subset of a hyperframe, or replace the designated subset with another hyperframe.

The function [.hyperframe is a method for the subset operator [for the class "hyperframe". It extracts the subset of x specified by the row index i and column index j.

The argument drop determines whether the array structure will be discarded if possible. The argument strip determines whether the list structure in a row or column or cell will be discarded if possible. If drop=FALSE (the default), the return value is always a hyperframe or data frame. If drop=TRUE, and if the selected subset has only one row, or only one column, or both, then

- if strip=FALSE, the result is a list, with one entry for each array cell that was selected.
- if strip=TRUE,
  - if the subset has one row containing several columns, the result is a list or (if possible) an atomic vector;
  - if the subset has one column containing several rows, the result is a list or (if possible) an atomic vector;
  - if the subset has exactly one row and exactly one column, the result is the object (or atomic value) contained in this row and column.

The function `[<-.hyperframe` is a method for the subset replacement operator `[<-` for the class `"hyperframe"`. It replaces the designated subset with the hyperframe `value`. The subset of `x` to be replaced is designated by the arguments `i` and `j` as above. The replacement `value` should be a hyperframe with the appropriate dimensions, or (if the specified subset is a single column) a list of the appropriate length.

The function `$.hyperframe` is a method for `$` for hyperframes. It extracts the relevant column of the hyperframe. The result is always a list (i.e. equivalent to using `[.hyperframe` with `strip=FALSE`).

The function `$<-.hyperframe` is a method for `$<-` for hyperframes. It replaces the relevant column of the hyperframe. The replacement value should be a list of the appropriate length.

The functions `[[.hyperframe` and `[[<-.hyperframe` are methods for `[[` and `[[<-.hyperframe` for hyperframes. They are analogous to `[[.data.frame` and `[[<-.data.frame` in that they can be used in different ways:

- when `[[.hyperframe` or `[[<-.hyperframe` are used with a single index, as in `x[[n]]` or `x[[n]] <- value`, they index the hyperframe as if it were a list, extracting or replacing a column of the hyperframe.
- when `[[.hyperframe` or `[[<-.hyperframe` are used with two indices, as in `x[[i,j]]` or `x[[i,j]] <- value`, they index the hyperframe as if it were a matrix, and can only be used to extract or replace one element.

Value

A hyperframe (of class `"hyperframe"`).

Author(s)

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

See Also

`hyperframe`

Examples

```
h <- hyperframe(X=list(square(1), square(2)), Y=list(sin, cos))
h
h[1, ]
h[1, ,drop=TRUE]
h[ , 1]
h[ , 1, drop=TRUE]
h[1,1]
h[1,1,drop=TRUE]
h[1,1,drop=TRUE,strip=FALSE]
h[1,1] <- list(square(3))
# extract column
h$X
# replace existing column
h$Y <- list(cells, cells)
# add new column
```

```
h$Z <- list(tan, exp)
#
h[["Y"]]
h[[2,1]]
h[[2,1]] <- square(3)
```

`Extract.im` *Extract Subset of Image*

Description

Extract a subset or subregion of a pixel image.

Usage

```
  ## S3 method for class 'im'
x[i, j, ..., drop=TRUE, tight=FALSE,
                                raster=NULL, rescue=is.owin(i),
                                rule.pix=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 linear network (object of class `"linnet"`) or a point pattern (an object of class `"ppp"`), or any type of index that applies to a matrix, or something that can be converted to a point pattern by `as.ppp` (using the window of `x`).
`j`	An integer or logical vector serving as the column index if matrix indexing is being used. Ignored if `i` is a spatial object.
`...`	Ignored.
`drop`	Logical value, specifying whether to return a vector containing the selected pixel values (`drop=TRUE`, the default) or to return a pixel image containing these values in their original spatial positions (`drop=FALSE`). The exception is that if `i` is a point pattern, then `drop` specifies whether to delete `NA` values. See Details.
`tight`	Logical value. If `tight=TRUE`, and if the result of the subset operation is an image, the image will be trimmed to the smallest possible rectangle.
`raster`	Optional. An object of class `"owin"` or `"im"` determining a pixel grid.
`rescue`	Logical value indicating whether rectangular blocks of data should always be returned as pixel images.
`rule.pix`	Argument passed to `owin2mask` when `i` is a polygonal window. This determines which pixels are deemed to lie inside the window.

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 arguments i,j according to the following rules (which are checked in this order):

1. i is a spatial object such as a window, a pixel image with logical values, a linear network, or a point pattern; or
2. i,j are indices for the matrix as.matrix(x); or
3. i can be converted to a point pattern by as.ppp(i, W=Window(x)), and i is not a matrix.

If i is a spatial window (an object of class "owin"), the pixels inside this window are selected.

- If drop=TRUE (the default) and either is.rectangle(i)=FALSE or rescue=FALSE, the pixel values are extracted; the result is a vector, with one entry for each pixel of x that lies inside the window i. Pixel values may be NA, indicating that the selected pixel lies outside the spatial domain of the image.
- if drop=FALSE, the result is another pixel image, obtained by setting the pixel values to NA outside the window i. The effect is that the pixel image x is clipped to the window i.
- if i is a rectangle and rescue=TRUE, the result is a pixel image as described above.
- To ensure that an image is produced in all circumstances, set drop=FALSE. To ensure that pixel values are extracted as a vector in all circumstances, set drop=TRUE, rescue=FALSE.

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 linear network (object of class "linnet"), the pixels which lie on this network are selected.

- If drop=TRUE (the default), the pixel values are extracted; the result is a vector, with one entry for each pixel of x that lies along the network i. Pixel values may be NA, indicating that the selected pixel lies outside the spatial domain of the image.
- if drop=FALSE, the result is a pixel image on a linear network (object of class "linim"), obtained by setting the pixel values of x to NA except for those which lie on the network i. The effect is that the pixel image x is restricted to the network i.

If i is a point pattern (an object of class "ppp") or something that can be converted to a point pattern, then the values of the pixel image at the points of this pattern are extracted. The result is a vector of pixel values. This is a simple way to read the pixel values at a given spatial location.

- if drop=FALSE the length of the result is equal to the number of points in the pattern. It may contain NA values which indicate that the corresponding point lies outside the spatial domain of the image.
- if drop=TRUE (the default), NA values are deleted. The result is a vector whose length may be shorter than the number of points of the pattern.

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 the algorithm attempts to interpret i and j as indices for the matrix as.matrix(x). Either i or j may be missing or blank. The result is usually a vector or matrix of pixel values. Exceptionally the result is a pixel image if i,j determines a rectangular subset of the pixel grid, and if the user specifies rescue=TRUE.

Finally, if none of the above conditions is met, the object i may also be a data frame or list of x,y coordinates which will be converted to a point pattern, taking the observation window to be Window(x). Then the pixel values at these points will be extracted as a vector.

Value

Either a pixel image or a vector of pixel values. See Details.

Warnings

If you have a 2-column matrix containing the x, y coordinates of point locations, then to prevent this being interpreted as an array index, you should convert it to a data.frame or to a point pattern.

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

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
R <- affine(letterR, diag(c(1,1)/2), c(-2,-0.7))
plot(X[R, drop=FALSE])
plot(X[R, drop=FALSE, tight=TRUE])

# a point pattern
```

```
Y <- X[cells]

# 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.layered` *Extract or Replace Subset of a Layered Object*

Description

Extract or replace some or all of the layers of a layered object, or extract a spatial subset of each layer.

Usage

```
  ## S3 method for class 'layered'
x[i, j, drop=FALSE, ...]

  ## S3 replacement method for class 'layered'
x[i] <- value

  ## S3 replacement method for class 'layered'
x[[i]] <- value
```

Arguments

`x`	A layered object (class `"layered"`).
`i`	Subset index for the list of layers. A logical vector, integer vector or character vector specifying which layers are to be extracted or replaced.
`j`	Subset index to be applied to the data in each layer. Typically a spatial window (class `"owin"`).
`drop`	Logical. If `i` specifies only a single layer and `drop=TRUE`, then the contents of this layer will be returned.
`...`	Additional arguments, passed to other subset methods if the subset index is a window.
`value`	List of objects which shall replace the designated subset, or an object which shall replace the designated element.

Details

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

The function `[.layered` extracts a designated subset of a layered object. It is a method for `[` for the class `"layered"`.

The functions `[<-.layered` and `[[<-.layered` replace a designated subset or designated entry of the object by new values. They are methods for `[<-` and `[[<-` for the `"layered"` class.

The index `i` specifies which layers will be retained. It should be a valid subset index for the list of layers.

The index `j` will be applied to each layer. It is typically a spatial window (class `"owin"`) so that each of the layers will be restricted to the same spatial region. Alternatively `j` may be any subset index which is permissible for the `"["` method for each of the layers.

Value

Usually an object of class `"layered"`.

Author(s)

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

See Also

`layered`

Examples

```
D <- distmap(cells)
L <- layered(D, cells,
             plotargs=list(list(ribbon=FALSE), list(pch=16)))

L[-2]
L[, square(0.5)]

L[[3]] <- japanesepines
L
```

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

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.owin *Extract Subset of Window*

Description

Extract a subset of a window.

Usage

```
  ## S3 method for class 'owin'
x[i, ...]
```

Arguments

x	A spatial window (object of class "owin").
i	Object defining the subregion. Either a spatial window, or a pixel image with logical values.
...	Ignored.

Details

This function computes the intersection between the window x and the domain specified by i, using intersect.owin.

This function is a method for the subset operator "[" for spatial windows (objects of class "owin"). It is provided mainly for completeness.

The index i may be either a window, or a pixel image with logical values (the TRUE values of the image specify the spatial domain).

Value

Another spatial window (object of class "owin").

Author(s)

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

See Also

intersect.owin

Examples

```
W <- owin(c(2.5, 3.2), c(1.4, 2.9))
plot(letterR)
plot(letterR[W], add=TRUE, col="red")
```

Extract.ppp *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=FALSE, ..., clip=FALSE]
  ## 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, or a pixel image with logical values defining a subset of the original observation window.
value	Replacement value for the subset. A point pattern.
j	Redundant. Included for backward compatibility.
drop	Logical value indicating whether to remove unused levels of the marks, if the marks are a factor.
clip	Logical value indicating how to form the window of the resulting point pattern, when i is a window. If clip=FALSE (the default), the result has window equal to i. If clip=TRUE, the resulting window is the intersection between the window of x and the window i.
...	Ignored. This argument is 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 i 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 argument drop determines whether to remove unused levels of a factor, if the point pattern is multitype (i.e. the marks are a factor) or if the marks are a data frame in which some of the columns are factors.

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 i is a subset of Window(x). Nor does it check whether value lies inside Window(x).

Author(s)

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

See Also

subset.ppp.

ppp.object, owin.object, unmark, split.ppp, cut.ppp

Examples

```
# Longleaf pines data
lon <- longleaf
if(human <- interactive()) {
plot(lon)
}


# adult trees defined to have diameter at least 30 cm
```

```
 longadult <- subset(lon, marks >= 30)
 if(human){
 plot(longadult)
 }
 # note that the marks are still retained.
 # Use unmark(longadult) to remove the marks

 # New Zealand trees data
 if(human){
 plot(nztrees)          # plot shows a line of trees at the far right
 abline(v=148, lty=2)   # cut along this line
 }
 nzw <- owin(c(0,148),c(0,95)) # the subwindow
 # trim dataset to this subwindow
 nzsub <- nztrees[nzw]
 if(human){
 plot(nzsub)
 }

 # Redwood data
 if(human){
 plot(redwood)
 }
 # Random thinning: delete 60% of data
 retain <- (runif(npoints(redwood)) < 0.4)
 thinred <- redwood[retain]
 if(human){
 plot(thinred)
 }

 # Scramble 60% of data
if(require(spatstat.random)) {
 X <- redwood
 modif <- (runif(npoints(X)) < 0.6)
 X[modif] <- runifpoint(ex=X[modif])
}

 # Lansing woods data - multitype points
 lan <- lansing


 # Hickory trees
  hicks <- split(lansing)$hickory

 # Trees in subwindow
  win <- owin(c(0.3, 0.6),c(0.2, 0.5))
  lsub <- lan[win]

if(require(spatstat.random)) {
 # Scramble the locations of trees in subwindow, retaining their marks
  lan[win] <- runifpoint(ex=lsub) %mark% marks(lsub)
}
```

```
# Extract oaks only
oaknames <- c("redoak", "whiteoak", "blackoak")
oak <- lan[marks(lan) %in% oaknames, drop=TRUE]
oak <- subset(lan, marks %in% oaknames, drop=TRUE)

# To clip or not to clip
X <- unmark(demopat)
B <- owin(c(5500, 9000), c(2500, 7400))
opa <- par(mfrow=c(1,2))
plot(X, main="X[B]")
plot(X[B], add=TRUE,
     cols="blue", col="pink", border="blue",
     show.all=TRUE, main="")
plot(Window(X), add=TRUE)
plot(X, main="X[B, clip=TRUE]")
plot(B, add=TRUE, lty=2)
plot(X[B, clip=TRUE], add=TRUE,
     cols="blue", col="pink", border="blue",
     show.all=TRUE, main="")
par(opa)
```

`Extract.ppx` *Extract Subset of Multidimensional Point Pattern*

Description

Extract a subset of a multidimensional point pattern.

Usage

```
  ## S3 method for class 'ppx'
x[i, drop=FALSE, clip=FALSE, ...]
```

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; or a spatial domain of class `"boxx"` or `"box3"`.
`drop`	Logical value indicating whether to remove unused levels of the marks, if the marks are a factor.
`clip`	Logical value indicating how to form the domain of the resulting point pattern, when `i` is a box (object of class `"boxx"`). If `clip=FALSE` (the default), the result has domain equal to `i`. If `clip=TRUE`, the resulting domain is the intersection between the domain of `x` and the domain `i`.
`...`	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` may be either

- 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.
- a spatial domain of class `"boxx"` or `"box3"`. Points falling inside this region will be retained.

The argument `drop` determines whether to remove unused levels of a factor, if the point pattern is multitype (i.e. the marks are a factor) or if the marks are a data frame or hyperframe in which some of the columns are factors.

Use the function `unmark` to remove marks from a marked point pattern.

Value

A multidimensional point pattern (of class `"ppx"`).

Author(s)

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

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]
Y <- ppx(coords(cells), domain = boxx(c(0,1),c(0,1)))
dom <- shift(domain(Y), vec = c(.5,.5))
Y[dom]
Y[dom, clip=TRUE]
```

`Extract.psp` *Extract Subset of Line Segment Pattern*

Description

Extract a subset of a line segment pattern.

Usage

```
  ## S3 method for class 'psp'
x[i, j, drop, ..., fragments=TRUE]
```

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.
`fragments`	Logical value indicating whether to retain all pieces of line segments that intersect the new window (`fragments=TRUE`, the default) or to retain only those line segments that lie entirely inside the new window (`fragments=FALSE`).

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, by default, be clipped so that they lie entirely inside the window; but if `fragments=FALSE`, such segments will be removed.

Both "thinning" and "trimming" can be performed together.

Value

A line segment pattern (of class "psp").

Author(s)

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

See Also

psp.object, 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", lwd=2)
   plot(w, add=TRUE)
   u <- a[w, fragments=FALSE]
   plot(u, add=TRUE, col="blue", lwd=3)
```

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

See Also

`quad.object`, `[.ppp`.

Examples

```
Q <- quadscheme(nztrees)
W <- owin(c(0,148),c(0,95)) # a subwindow
Q[W]
```

`Extract.solist` *Extract or Replace Subset of a List of Spatial Objects*

Description

Extract or replace some entries in a list of spatial objects, or extract a designated sub-region in each object.

Usage

```
  ## S3 method for class 'solist'
x[i, ...]

  ## S3 replacement method for class 'solist'
x[i] <- value

  ## S3 replacement method for class 'solist'
x[[i]] <- value
```

Arguments

x	An object of class "solist" representing a list of two-dimensional spatial objects.
i	Subset index. Any valid subset index for vectors in the usual R sense, or a window (object of class "owin").
value	Replacement value for the subset. See Details.
...	Ignored.

Details

These are methods for extracting and replacing subsets for the class "solist".

The argument x should be an object of class "solist" representing a list of two-dimensional spatial objects. See solist.

For the subset method, the subset index i can be either a vector index (specifying some elements of the list) or a spatial window (specifying a spatial sub-region).

For the replacement "[<-" method, i must be a vector index and value must be a list of spatial objects; the designated elements of x will be replaced by the corresponding elements of the list value.

For the replacement "[[<-" method, i must be a single integer, and value must be a spatial object.

If all entries of x belong to the same class, then in the replacement methods, a value of NA is automatically coerced to an NA object of the same class.

Value

Another object of the same class as x.

Author(s)

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

See Also

solist, plot.solist, summary.solist

Examples

```
x <- solist(japanesepines, cells, redwood)
x[2:3]
x[square(0.5)]
x[1] <- list(finpines)
x[[1]] <- cells
x[[3]] <- NA
```

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

See Also

`split.ppp`, `plot.splitppp`, `summary.splitppp`

Examples

```
y <- split(amacrine)
y[[1]]
y[["off"]]
y[[1]] <- rsyst(Window(amacrine), 4, 3)
```

`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[i, ...]
  ## S3 replacement method for class 'tess'
x[i, ...] <- value
```

Arguments

`x`	A tessellation (object of class `"tess"`).
`i`	Subset index for the tiles of the tessellation. Alternatively a window (object of class `"owin"`).
`...`	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.

For `[.tess` only, the subset index can also be a window (object of class `"owin"`). The tessellation `x` is then intersected with the window.

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

See Also

tess, tiles, intersect.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
```

---

extrapolate.psp *Extrapolate Line Segments to Obtain Infinite Lines*

Description

Given a spatial pattern of line segments, extrapolate the segments to infinite lines.

Usage

```
extrapolate.psp(x, ...)
```

Arguments

x	Spatial pattern of line segments (object of class "psp").
...	Ignored.

Details

Each line segment in the pattern x is extrapolated to an infinite line, drawn through its two endpoints. The resulting pattern of infinite lines is returned as an object of class "infline".

If a segment's endpoints are identical (so that it has zero length) the resulting infinite line is vertical (i.e. parallel to the y coordinate axis).

Value

An object of class "infline" representing the pattern of infinite lines. See infline for details of structure.

Author(s)

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

See Also

psp, infline

midpoints.psp, lengths_psp angles.psp, endpoints.psp.

Examples

```
X <- psp(runif(4), runif(4), runif(4), runif(4), window=owin())
Y <- extrapolate.psp(X)
plot(X, col=3, lwd=4)
plot(Y, lty=3)
Y
```

`fardist` *Farthest Distance to Boundary of Window*

Description

Computes the farthest distance from each pixel, or each data point, to the boundary of the window.

Usage

```
  fardist(X, ...)

  ## S3 method for class 'owin'
fardist(X, ..., squared=FALSE)

  ## S3 method for class 'ppp'
fardist(X, ..., squared=FALSE)
```

Arguments

`X`	A spatial object such as a window or point pattern.
`...`	Arguments passed to `as.mask` to determine the pixel resolution, if required.
`squared`	Logical. If `TRUE`, the squared distances will be returned.

Details

The function `fardist` is generic, with methods for the classes `owin` and `ppp`.

For a window `W`, the command `fardist(W)` returns a pixel image in which the value at each pixel is the *largest* distance from that pixel to the boundary of `W`.

For a point pattern `X`, with window `W`, the command `fardist(X)` returns a numeric vector with one entry for each point of `X`, giving the largest distance from that data point to the boundary of `W`.

Value

For `fardist.owin`, a pixel image (object of class `"im"`).

For `fardist.ppp`, a numeric vector.

Author(s)

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

Rolf Turner <rolfturner@posteo.net>

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

Examples

```
fardist(cells)

plot(FR <- fardist(letterR))
```

FeretBox *Feret Box Enclosing a Domain*

Description

Given a spatial domain, calculate its Feret box, the narrowest rectangle enclosing the domain at a given angle (or at any angle).

Usage

```
FeretBox(W, theta = NULL)
```

Arguments

W	Window (object of class "owin") representing the domain.
theta	Either a single numeric value specifying an angle such that the sides of the box should be parallel and perpendicular to theta, or NULL signifying that the orientation of the box is unconstrained.

Details

If theta is given, then the smallest rectangle containing W with sides parallel and perpendicular to angle theta will be computed and returned.

If theta=NULL (the default), then the orientation of the rectangle is unconstrained. The rectangle with the narrowest possible width (at any orientation) is determined by finding the minimum Feret diameter.

Value

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

Author(s)

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

See Also

FeretDiamFun

Examples

```
W <- Window(chorley)
B <- FeretBox(W)
plot(B, main="", type="n")
plot(W, add=TRUE)
plot(B, add=TRUE, border="red")
```

FeretDiamFun *Feret Diameter as a Function of Direction*

Description

Create a function which calculates the Feret diameter of a domain at any given angle. Alternatively, calculate the minimum or maximum value of this function.

Usage

```
FeretDiamFun(W)
minFeretDiam(W)
maxFeretDiam(W)
```

Arguments

W	Window (object of class "owin") representing the domain.

Details

The Feret diameter or caliper diameter of a domain W, at a given angle θ, is length of the projection of W onto a line with orientation θ.

Equivalently, the Feret diameter at angle θ is the distance between the tangents to W that are perpendicular to direction θ.

The Feret diameter at orientation θ can also be written

$$d_W(\theta) = h_W(\theta) - h_W(\theta + \pi)$$

where $h_W(\theta)$ is the support function of W.

The command FeretDiamFun creates a function in the R language, with one argument theta, which can calculate the Feret diameter at any given angle or angles theta.

The commands minFeretDiam and maxFeretDiam calculate the minimum and maximum value, respectively, of Feret diameter at any orientation.

Value

The result of FeretDiamFun is a function in the R language with one argument theta.

The result of minFeretDiam or maxFeretDiam is a single numeric value.

Author(s)

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

See Also

FeretBox, SupportFun

Examples

```
W <- letterR
Frame(W) <- grow.rectangle(Frame(W), 0.2)
plot(W, main="")

d <- FeretDiamFun(W)
d(pi/6)

f <- SupportFun(W)
tt <- pi/6 + c(0, pi)
plot(infline(theta=tt, p=f(tt)), col="red", lwd=2)

## as a function
curve(d, to=2*pi)

minFeretDiam(W)
```

---

fillholes.owin *Remove Small Holes in a Window*

Description

Given a window, this function removes any small holes in the window.

Usage

```
fillholes.owin(W, amin)
```

Arguments

W	Window (object of class "owin").
amin	Numeric value. The smallest permissible area of a hole.

Details

This function simplifies a window W by filling in any holes with area less than amin.

The argument W must be a window (object of class "owin").

If amin=Inf, all holes will be filled in.

Value

Another window (object of class "owin") of the same type as W.

Author(s)

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

See Also

simplify.owin, owin

Examples

```
## make object with small hole
A <- meningitis$kreise
W <- setminus.owin(Window(A), tiles(A)[[102]])
amin <- 500

opa <- par(mfrow=c(1,2))
plot(W)
plot(fillholes.owin(W, amin))

M <- as.mask(W)
plot(M)
plot(fillholes.owin(M, amin))
```

---

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

See Also

affine, reflect, rotate, shift

Examples

```
X <- flipxy(cells)
```

---

fourierbasis *Fourier Basis Functions*

Description

Evaluates the Fourier basis functions on a d-dimensional box with d-dimensional frequencies k_i at the d-dimensional coordinates x_j.

Usage

```
fourierbasis(x, k, win = boxx(rep(list(0:1), ncol(k))))
fourierbasisraw(x, k, boxlengths)
```

Arguments

x	Coordinates. A data.frame or matrix with n rows and d columns giving the d-dimensional coordinates.
k	Frequencies. A data.frame or matrix with m rows and d columns giving the frequencies of the Fourier-functions.
win	window (of class "owin", "box3" or "boxx") giving the d-dimensional box domain of the Fourier functions.
boxlengths	numeric giving the side lengths of the box domain of the Fourier functions.

Details

The result is an m by n matrix where the (i, j)'th entry is the d-dimensional Fourier basis function with frequency k_i evaluated at the point x_j, i.e.,

$$\frac{1}{\sqrt{|W|}} \exp(2\pi i \sum l = 1^d k_{i,l} x_{j,l} / L_l)$$

where L_l, $l = 1, ..., d$ are the box side lengths and $|W|$ is the volume of the domain (window/box). Note that the algorithm does not check whether the coordinates given in x are contained in the given box. Actually the box is only used to determine the side lengths and volume of the domain for normalization.

The stripped down faster version fourierbasisraw doesn't do checking or conversion of arguments and requires x and k to be matrices.

Value

An m by n matrix of complex values.

Author(s)

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

Rolf Turner <rolfturner@posteo.net>

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

Examples

```
## 27 rows of three dimensional Fourier frequencies:
k <- expand.grid(-1:1,-1:1, -1:1)
## Two random points in the three dimensional unit box:
x <- rbind(runif(3),runif(3))
## 27 by 2 resulting matrix:
v <- fourierbasis(x, k)
head(v)
```

Frame *Extract or Change the Containing Rectangle of a Spatial Object*

Description

Given a spatial object (such as a point pattern or pixel image) in two dimensions, these functions extract or change the containing rectangle inside which the object is defined.

Usage

```
   Frame(X)

   ## Default S3 method:
Frame(X)

   Frame(X) <- value

   ## S3 replacement method for class 'owin'
Frame(X) <- value

   ## S3 replacement method for class 'ppp'
Frame(X) <- value

   ## S3 replacement method for class 'im'
Frame(X) <- value

   ## Default S3 replacement method:
Frame(X) <- value
```

Arguments

X	A spatial object such as a point pattern, line segment pattern or pixel image.
value	A rectangular window (object of class "owin" of type "rectangle") to be used as the new containing rectangle for X.

Details

The functions Frame and Frame<- are generic.

Frame(X) extracts the rectangle inside which X is defined.

Frame(X) <- R changes the rectangle inside which X is defined to the new rectangle R.

Value

The result of Frame is a rectangular window (object of class "owin" of type "rectangle").

The result of Frame<- is the updated object X, of the same class as X.

Author(s)

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

See Also

Window

Examples

```
Frame(cells)
X <- demopat
Frame(X)
Frame(X) <- owin(c(0, 11000), c(400, 8000))
```

---

framedist.pixels *Distance to Bounding Frame*

Description

Computes the distances from each pixel to the bounding rectangle.

Usage

```
framedist.pixels(w, ..., style=c("image", "matrix", "coords"))
```

Arguments

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

Details

This function computes, for each pixel u in the rectangular frame Frame(w), the shortest distance to the boundary of $Frame(w)$.

The grid of pixels is determined by the arguments "\dots" passed to as.mask. The distance from each pixel to the boundary is calculated exactly, using analytic geometry.

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

See Also

`bdist.pixels`.

Examples

```
opa <- par(mfrow=c(1,2))
plot(framedist.pixels(letterR))
plot(bdist.pixels(letterR))
par(opa)
```

---

funxy *Spatial Function Class*

Description

A simple class of functions of spatial location

Usage

```
funxy(f, W)
```

Arguments

`f`	A `function` in the R language with arguments `x,y` (at least)
`W`	Window (object of class `"owin"`) inside which the function is well-defined.

Details

This command creates an object of class "funxy". This is a simple mechanism for handling a function of spatial location $f(x, y)$ to make it easier to display and manipulate.

f should be a function in the R language. The first two arguments of f must be named x and y respectively.

W should be a window (object of class "owin") inside which the function f is well-defined.

The function f should be vectorised: that is, if x and y are numeric vectors of the same length n, then v <- f(x,y) should be a vector of length n.

The resulting function g <- funxy(f, W) has the same formal arguments as f and can be called in the same way, v <- g(x,y) where x and y are numeric vectors. However it can also be called as v <- g(X), where X is a point pattern (object of class "ppp" or "lpp") or a quadrature scheme (class "quad"); the function will be evaluated at the points of X.

The result also has a unitname, inherited from W.

Value

A function with the same arguments as f, which also belongs to the class "funxy". This class has methods for print, plot, contour and persp.

Author(s)

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

See Also

plot.funxy, summary.funxy

Examples

```
f <- function(x,y) { x^2 + y^2 - 1}
g <- funxy(f, square(2))
g
## evaluate function at any x, y coordinates
g(0.2, 0.3)
## evaluate function at the points of a point pattern
g(cells[1:4])
```

---

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

quad.object, quadscheme, inside.owin, stratrand

Examples

```
w <- unit.square()
xy <- gridcentres(w, 10,15)
if(human <- interactive()) {
plot(w)
points(xy)
}

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)
```

```
if(human) {
plot(w)
points(xy$x[ok], xy$y[ok])
}
```

`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] * 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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

quad.object, dirichletWeights

Examples

```
Q <- quadscheme(runifrect(15))
X <- as.ppp(Q) # data and dummy points together
w <- gridweights(X, 10)
w <- gridweights(X, c(10, 10))
```

---

grow.boxx *Add margins to box in any dimension*

Description

Adds a margin to a box of class boxx.

Usage

```
grow.boxx(W, left, right = left)
grow.box3(W, left, right = left)
```

Arguments

W	A box (object of class "boxx" or "box3").
left	Width of margin to be added to left endpoint of box side in every dimension. A single nonnegative number, or a vector of same length as the dimension of the box to add different left margin in each dimension.
right	Width of margin to be added to right endpoint of box side in every dimension. A single nonnegative number, or a vector of same length as the dimension of the box to add different right margin in each dimension.

Value

Another object of the same class "boxx" or "box3" representing the window after margins are added.

Author(s)

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

See Also

grow.rectangle, boxx, box3

Examples

```
w <- boxx(c(0,10), c(0,10), c(0,10), c(0,10))
# add a margin of size 1 on both sides in all four dimensions
b12 <- grow.boxx(w, 1)

# add margin of size 2 at left, and margin of size 3 at right,
# in each dimension.
v <- grow.boxx(w, 2, 3)
```

---

grow.rectangle *Add margins to rectangle*

Description

Adds a margin to a rectangle.

Usage

```
grow.rectangle(W, xmargin=0, ymargin=xmargin, fraction=NULL)
```

Arguments

W	A window (object of class "owin"). Must be of type "rectangle".
xmargin	Width of horizontal margin to be added. 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 added. A single nonnegative number, or a vector of length 2 indicating margins of unequal width at bottom and top.
fraction	Fraction of width and height to be added. A number greater than zero, or a numeric vector of length 2 indicating different fractions of width and of height, respectively. Incompatible with specifying xmargin and ymargin.

Details

This is a simple convenience function to add a margin of specified width and height on each side of a rectangular window. Unequal margins can also be added.

Value

Another object of class "owin" representing the window after margins are added.

Author(s)

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

See Also

trim.rectangle, dilation, erosion, owin.object

Examples

```
w <- square(10)
# add a margin of width 1 on all four sides
square12 <- grow.rectangle(w, 1)

# add margin of width 3 on the right side
# and margin of height 4 on top.
v <- grow.rectangle(w, c(0,3), c(0,4))

# grow by 5 percent on all sides
grow.rectangle(w, fraction=0.05)
```

---

harmonise *Make Objects Compatible*

Description

Converts several objects of the same class to a common format so that they can be combined or compared.

Usage

```
harmonise(...)
harmonize(...)
```

Arguments

...	Any number of objects of the same class.

Details

This generic command takes any number of objects of the same class, and *attempts* to make them compatible in the sense of compatible so that they can be combined or compared.

There are methods for the classes "fv" (harmonise.fv) and "im" (harmonise.im).

All arguments ... must be objects of the same class. The result will be a list, of length equal to the number of arguments ..., containing new versions of each of these objects, converted to a common format. If the arguments were named (name=value) then the return value also carries these names.

Value

A list, of length equal to the number of arguments ..., whose entries are objects of the same class. If the arguments were named (name=value) then the return value also carries these names.

Author(s)

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

See Also

compatible, harmonise.fv, harmonise.im

---

harmonise.im *Make Pixel Images Compatible*

Description

Convert several pixel images to a common pixel raster.

Usage

```
## S3 method for class 'im'
harmonise(...)

## S3 method for class 'im'
harmonize(...)
```

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.

The command harmonise is generic. This is the method for objects of class "im".

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

See Also

commonGrid, compatible.im, as.im

Examples

```
Image1 <- setcov(square(1), dimyx=32)
Image2 <- setcov(square(1), dimyx=16)
Function1 <- function(x,y) { x }
Window1 <- shift(letterR, c(-2, -1))
h <- harmonise(X=Image1, Y=Image2, Z=Function1, W=Window1)
plot(h, main="")
```

---

harmonise.owin *Make Windows Compatible*

Description

Convert several windows to a common pixel raster.

Usage

```
## S3 method for class 'owin'
harmonise(...)

## S3 method for class 'owin'
harmonize(...)
```

Arguments

...	Any number of windows (objects of class "owin") or data which can be converted to windows by as.owin.

Details

This function makes any number of windows compatible, by converting them all to a common pixel grid.

This only has an effect if one of the windows is a binary mask. If all the windows are rectangular or polygonal, they are returned unchanged.

The command harmonise is generic. This is the method for objects of class "owin".

Each argument must be a window (object of class "owin"), or data that can be converted to a window by as.owin.

The common pixel grid is determined by inspecting all the windows in the argument list, computing the bounding box of all the windows, then finding the binary mask with the finest 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 windows, use commonGrid.

Value

A list of windows, of length equal to the number of arguments .... The list belongs to the class "solist".

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

commonGrid, harmonise.im, as.owin

Examples

```
harmonise(X=letterR,
          Y=grow.rectangle(Frame(letterR), 0.2),
          Z=as.mask(letterR, eps=0.1),
          V=as.mask(letterR, eps=0.07))
```

---

harmoniseLevels *Harmonise the levels of several factors, or factor-valued pixel images.*

Description

Given several factors (or factor-valued pixel images) convert them so that they all use the same set of levels.

Usage

```
harmoniseLevels(...)
```

Arguments

...	Factors, or factor-valued pixel images.

Details

All of the arguments ... must be factors, or factor-valued pixel images (objects of class "im").

The levels of each factor will be extracted, and combined by taking the union of all the levels. Then each factor will be converted to a new factor so that all of the new factors have exactly the same set of levels.

Value

A list, containing the same number of arguments as the input, consisting of factors or factor-valued pixel images.

Author(s)

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

See Also

levels, levels.im, mergeLevels.

Examples

```
(a <- factor(sample(letters[1:3], 10, replace=TRUE)))
(b <- factor(sample(LETTERS[1:4], 7, replace=TRUE)))
harmoniseLevels(a,b)

(A <- gorillas.extra$vegetation)
(B <- gorillas.extra$slopetype)
harmoniseLevels(A,B)
```

---

has.close *Check Whether Points Have Close Neighbours*

Description

For each point in a point pattern, determine whether the point has a close neighbour in the same pattern.

Usage

```
has.close(X, r, Y=NULL, ...)

## Default S3 method:
has.close(X,r, Y=NULL, ..., periodic=FALSE)

## S3 method for class 'ppp'
has.close(X,r, Y=NULL, ..., periodic=FALSE, sorted=FALSE)

## S3 method for class 'pp3'
has.close(X,r, Y=NULL, ..., periodic=FALSE, sorted=FALSE)
```

Arguments

X, Y	Point patterns of class "ppp" or "pp3" or "lpp".
r	Threshold distance: a number greater than zero.
periodic	Logical value indicating whether to measure distances in the periodic sense, so that opposite sides of the (rectangular) window are treated as identical.
sorted	Logical value, indicating whether the points of X (and Y, if given) are already sorted into increasing order of the x coordinates.
...	Other arguments are ignored.

Details

This is simply a faster version of (nndist(X) <= r) or (nncross(X,Y,what="dist") <= r).

has.close(X,r) determines, for each point in the pattern X, whether or not this point has a neighbour in the same pattern X which lies at a distance less than or equal to r.

has.close(X,r,Y) determines, for each point in the pattern X, whether or not this point has a neighbour in the *other* pattern Y which lies at a distance less than or equal to r.

The function has.close is generic, with methods for "ppp" and "pp3" and a default method.

Value

A logical vector, with one entry for each point of X.

Author(s)

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

See Also

nndist

Examples

```
has.close(redwood, 0.05)
with(split(amacrine), has.close(on, 0.05, off))
with(osteo, sum(has.close(pts, 20)))
```

---

headtail *First or Last Part of a Spatial Pattern*

Description

Returns the first few elements (head) or the last few elements (tail) of a spatial pattern.

Usage

```
  ## S3 method for class 'ppp'
head(x, n = 6L, ...)

  ## S3 method for class 'ppx'
head(x, n = 6L, ...)

  ## S3 method for class 'psp'
head(x, n = 6L, ...)

  ## S3 method for class 'tess'
head(x, n = 6L, ...)

  ## S3 method for class 'ppp'
tail(x, n = 6L, ...)

  ## S3 method for class 'ppx'
tail(x, n = 6L, ...)

  ## S3 method for class 'psp'
tail(x, n = 6L, ...)

  ## S3 method for class 'tess'
tail(x, n = 6L, ...)
```

Arguments

x	A spatial pattern of geometrical figures, such as a spatial pattern of points (an object of class "ppp", "pp3", "ppx" or "lpp") or a spatial pattern of line segments (an object of class "psp") or a tessellation (object of class "tess").
n	Integer. The number of elements of the pattern that should be extracted.
...	Ignored.

Details

These are methods for the generic functions head and tail. They extract the first or last n elements from x and return them as an object of the same kind as x.

To inspect the spatial coordinates themselves, use View(x) or head(as.data.frame(x)).

Value

An object of the same class as x.

Author(s)

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

See Also

View, edit.

Conversion to data frame: as.data.frame.ppp, as.data.frame.ppx, as.data.frame.psp

Examples

```
head(cells)
tail(edges(letterR), 5)
head(dirichlet(cells), 4)
```

---

hextess *Hexagonal Grid or Tessellation*

Description

Construct a hexagonal grid of points, or a hexagonal tessellation.

Usage

```
hexgrid(W, s, offset = c(0, 0), origin=NULL, trim = TRUE)

hextess(W, s, offset = c(0, 0), origin=NULL, trim = TRUE)
```

Arguments

W	Window in which to construct the hexagonal grid or tessellation. An object of class "owin".
s	Side length of hexagons. A positive number.
offset	Numeric vector of length 2 specifying a shift of the hexagonal grid. See Details.
origin	Numeric vector of length 2 specifying the initial origin of the hexagonal grid, before the offset is applied. See Details.
trim	Logical value indicating whether to restrict the result to the window W. See Details.

Details

hexgrid constructs a hexagonal grid of points on the window W. If trim=TRUE (the default), the grid is intersected with W so that all points lie inside W. If trim=FALSE, then we retain all grid points which are the centres of hexagons that intersect W.

hextess constructs a tessellation of hexagons on the window W. If trim=TRUE (the default), the tessellation is restricted to the interior of W, so that there will be some fragmentary hexagons near the boundary of W. If trim=FALSE, the tessellation consists of all hexagons which intersect W.

The points of hexgrid(...) are the centres of the tiles of hextess(...) in the same order.

In the initial position of the grid or tessellation, one of the grid points (tile centres) is placed at the origin, which defaults to the midpoint of the bounding rectangle of W. The grid can be shifted relative to this origin by specifing the offset.

Value

The value of hexgrid is a point pattern (object of class "ppp").

The value of hextess is a tessellation (object of class "tess").

Author(s)

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

See Also

tess

hexagon

Examples

```
if(interactive()) {
  W <- Window(chorley)
  s <- 0.7
} else {
  W <- letterR
  s <- 0.3
}
plot(hextess(W, s))
plot(hexgrid(W, s), add=TRUE)
```

---

hist.funxy *Histogram of Values of a Spatial Function*

Description

Computes and displays a histogram of the values of a spatial function of class "funxy".

Usage

```
  ## S3 method for class 'funxy'
hist(x, ..., xname)
```

Arguments

x	A pixel image (object of class "funxy").
...	Arguments passed to as.im or hist.im.
xname	Optional. Character string to be used as the name of the dataset x.

Details

This function computes and (by default) displays a histogram of the values of the function x.

An object of class "funxy" describes a function of spatial location. It is a function(x,y,..) in the R language, with additional attributes.

The function hist.funxy is a method for the generic function hist for the class "funxy".

The function is first converted to a pixel image using as.im, then hist.im is called to produce the histogram.

Any arguments in ... are passed to as.im to determine the pixel resolution, or to hist.im to determine the histogram breaks and to control or suppress plotting. Useful arguments include W for the spatial domain, eps,dimyx for pixel resolution, main for the main title.

Value

An object of class "histogram" as returned by hist.default. This object can be plotted.

Author(s)

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

See Also

spatialcdf for the cumulative distribution function of an image or function.

hist, hist.default.

For other statistical graphics such as Q-Q plots, use as.im(X)[] to extract the pixel values of image X, and apply the usual statistical graphics commands.

Examples

```
f <- funxy(function(x,y) {x^2}, unit.square())
hist(f)
```

---

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, xname)
```

Arguments

x	A pixel image (object of class "im").
...	Arguments passed to hist.default or barplot.
probability	Logical. If TRUE, the histogram will be normalised to give probabilities or probability densities.
xname	Optional. Character string to be used as the name of the dataset x.

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

See Also

spatialcdf for the cumulative distribution function of an image.

hist, hist.default, barplot.

For other statistical graphics such as Q-Q plots, use X[] to extract the pixel values of image X, and apply the usual statistical graphics commands.

For information about pixel images see im.object, summary.im.

Examples

```
X <- as.im(function(x,y) {x^2}, unit.square())
hist(X)
hist(cut(X,3))
```

`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=NULL)
```

Arguments

`...`	Arguments of the form `value` or `tag=value`. Each `value` is either an atomic vector, a factor, a list of objects of the same class, 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, factor, or an object of class `"Date"` or `"Surv"`)
- 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, split, [, [<-, [[, [[<-, $, $<-, names, as.data.frame as.list, cbind and rbind for the class of hyperframes. There is also is.hyperframe and as.hyperframe.

Handling Character Strings

The argument stringsAsFactors is a logical value (passed to data.frame) specifying how to handle pixel values which are character strings. If TRUE, character values are interpreted as factor levels. If FALSE, they remain as character strings. The default values of stringsAsFactors depends on the version of R.

- In R versions < 4.1.0 the factory-fresh default is stringsAsFactors=FALSE and the default can be changed by setting options(stringsAsFactors=FALSE).
- in R versions >= 4.1.0 the default is stringsAsFactors=FALSE and there is no option to change the default.

Author(s)

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

See Also

as.hyperframe, as.hyperframe.ppx, plot.hyperframe, [.hyperframe, with.hyperframe, split.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)
if(require(spatstat.random)) {
  X <- solapply(as.list(lambda), rpoispp)
} else {
  X <- solapply(as.list(lambda), function(lam) runifrect(rpois(1, lam)))
}
  h <- hyperframe(lambda=lambda, X=X)
  h

  h$lambda2 <- lambda^2
```

```
h[, "lambda3"] <- lambda^3
h[, "Y"] <- X

h[[2, "lambda3"]]
```

`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

`x`	A point pattern (object of class `"ppp"`).
`...`	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@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>

See Also

`identify`, `clickppp`

identify.psp *Identify Segments in a Line Segment Pattern*

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, paint=plot, paint.args=list())
```

Arguments

x	A line segment pattern (object of class "psp").
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.
...	Arguments passed to text.default controlling the plotting of the labels, if plot=TRUE.
paint	Logical. Whether to redraw each identified segment, using a different colour.
paint.args	Optional list of arguments passed to plot.psp determining the colour and style in which each identified segment will be redrawn, if paint=TRUE.

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).

The procedure terminates when the right mouse button is pressed.

Value

Vector containing the serial numbers of the segments in the pattern x that were identified.

Author(s)

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

See Also

identify, identify.ppp.

---

identify.tess *Interactively Identify Tiles of a Tessellation*

Description

If a tessellation is plotted in the graphics window, then each time the left mouse button is pressed, this function will find the tile which contains the mouse position, and print its mark value (or its serial number if there is no mark).

Usage

```
  ## S3 method for class 'tess'
identify(x, ...,
                  labels=tilenames(x),
                  n=nobjects(x), plot=TRUE, paint=plot, paint.args=list())
```

Arguments

x	A tessellation (object of class "tess").
...	Arguments passed to plot.owin and text.default controlling the graphical display.
labels	Labels associated with the tiles, to be plotted when the tiles are identified. A character vector or numeric vector of length equal to the number of tiles in x.
n	Maximum number of tiles to be identified.
plot	Logical. Whether to plot the labels when a tile is identified.
paint	Logical. Whether to redraw each identified tile, using a different colour.
paint.args	Optional list of arguments passed to plot.owin determining the colour and style in which each identified tile will be redrawn, if paint=TRUE.

Details

This is a method for the generic function identify for tessellation objects.

The tessellation x should first be plotted using plot.tess. Then identify(x) reads the position of the graphics pointer each time the left mouse button is pressed. It then determines which tile of x contains the mouse position. The index of this tile (and its mark if any) will be returned as part of the value of the call.

Each time a tile is identified, the tile will be plotted in light blue fill colour, and text will be displayed inside the tile showing the name of the tile.

Value

A `data.frame` with columns `id` and `name` containing the serial numbers and names of the tiles of `x` that were identified, in the order that they were identified; If `x` is marked, subsequent columns contain the marks for these tiles.

Author(s)

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

See Also

`identify`, `plot.tess`

`im` *Create a Pixel Image Object*

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

`mat`	matrix or vector containing the pixel values of the image.
`xcol`	vector of x coordinates for the pixel grid
`yrow`	vector of y coordinates for the pixel grid
`xrange`, `yrange`	Optional. Vectors of length 2 giving the x and y limits of the enclosing rectangle. (Ignored if `xcol`, `yrow` are present.)
`unitname`	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 'pixel image' or two-dimensional array of values.

The pixel grid is rectangular and occupies a rectangular window in the spatial coordinate system. The pixel values are *scalars*: they can be real numbers, integers, complex numbers, single characters or strings, logical values, or categorical values. A pixel's value can also be `NA`, meaning that no value is defined at that location, and effectively that pixel is 'outside' the window. Although the pixel values must be scalar, photographic colour images (i.e., with red, green, and blue brightness

channels) can be represented as character-valued images in **spatstat**, using R's standard encoding of colours as character strings.

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

`im.object` for details of the class.

`as.im` for converting other kinds of data to an image.

`as.matrix.im`, `[.im`, `eval.im` for manipulating images.

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.apply` *Apply Function Pixelwise to List of Images*

Description

Returns a pixel image obtained by applying a given function to corresponding pixels in several pixel images.

Usage

```
im.apply(X, FUN, ..., fun.handles.na=FALSE, check=TRUE, verbose=TRUE)
```

Arguments

`X`	A list of pixel images (objects of class `"im"`).
`FUN`	A function that can be applied to vectors, or a character string giving the name of such a function.
`...`	Additional arguments to `FUN`.
`fun.handles.na`	Logical value specifying what to do when the data include `NA` values. See Details.
`check`	Logical value specifying whether to check that the images in `X` are compatible (for example that they have the same grid of pixel locations) and to convert them to compatible images if necessary.
`verbose`	Logical value specifying whether to print informative messages.

Details

The argument X should be a list of pixel images (objects of class "im"). If the images do not have identical pixel grids, they will be converted to a common grid using harmonise.im.

At each pixel location, the values of the images in X at that pixel will be extracted as a vector; the function FUN will be applied to this vector; and the return value of FUN will become the pixel value of the resulting image. For example im.apply(X, mean) will return a pixel image in which the value of each pixel is the average of the corresponding pixel values in the images in X.

If the result of FUN is a vector, then the result of im.apply will be a list of images. For example im.apply(X, range) will return a list of two images containing the pixelwise minimum and pixelwise maximum, respectively, of the input images in X.

The argument fun.handles.na specifies what to do when some of the pixel values are NA.

- If fun.handles.na=FALSE (the default), the function FUN is never applied to data that include NA values; the result is defined to be NA whenever the data contain NA.
- If fun.handles.na=TRUE, the function FUN will be applied to all pixel data, including those which contain NA values.

Value

A pixel image (object of class "im") or a list of pixel images.

Computation details

The computation is performed efficiently using matrix operations where possible, by constructing a large matrix containing all the pixel data for all images. However, if this matrix would exceed the maximum permitted size of matrices (as given in spatstat.options("maxmatrix")), then the calculation is performed on smaller sub-images, and this is reported in a message if verbose=TRUE.

Author(s)

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

See Also

eval.im for algebraic operations with images.

Examples

```
# list of two pixel images
Y <- solapply(bei.extra, scaletointerval)
plot(Y)
im.apply(Y, max)
im.apply(Y, sum)
im.apply(Y, range)

## Example with incompatible patterns of NA values
B <- owin(c(438, 666), c(80, 310))
Y[[1]][B] <- NA
```

```
opa <- par(mfrow=c(2,2))
plot(Y[[1]])
plot(Y[[2]])
#' Default action: NA -> NA
plot(im.apply(Y, mean))
#' Use NA handling in mean.default
plot(im.apply(Y, mean, na.rm=TRUE, fun.handles.na=TRUE))
par(opa)
```

`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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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

`X`	A pixel image (object of class `"im"`.
`Y`	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

$$C(v) = \int X(u)X(u-v)\,\mathrm{d}u$$

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

$$C(v) = \int X(u)Y(u-v)\,\mathrm{d}u.$$

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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)

inradius(W)
```

Arguments

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

Details

Given a window W of any type and shape, the function incircle determines the largest circle that is contained inside W, while inradius computes its radius only.

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

The result of incircle is a list with entries x,y,r giving the location (x,y) and radius r of the incircle.

The result of inradius is the numerical value of radius.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

centroid.owin

Examples

```
W <- square(1)
Wc <- incircle(W)
plot(W)
plot(disc(Wc$r, c(Wc$x, Wc$y)), add=TRUE)

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)
```

`infline` *Infinite Straight Lines*

Description

Define the coordinates of one or more straight lines in the plane

Usage

```
infline(a = NULL, b = NULL, h = NULL, v = NULL, p = NULL, theta = NULL)

## S3 method for class 'infline'
print(x, ...)

## S3 method for class 'infline'
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 "infline"
...	Extra arguments passed to print for printing or abline for plotting

Details

The class infline 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

$$x \cos\theta + y \sin\theta = p$$

The command infline 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 infline converts between different coordinate systems (e.g. from a,b to p,theta) and returns an object of class "infline" that contains a representation of the lines in each appropriate coordinate system. This object can be printed and plotted.

Value

The value of infline is an object of class "infline" 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@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>.

See Also

rotate.infline, clip.infline, chop.tess, whichhalfplane

Examples

```
infline(a=10:13,b=1)
infline(p=1:3, theta=pi/4)
plot(c(-1,1),c(-1,1),type="n",xlab="",ylab="", asp=1)
plot(infline(p=0.4, theta=seq(0,pi,length=20)))
```

---

inside.boxx *Test Whether Points Are Inside A Multidimensional Box*

Description

Test whether points lie inside or outside a given multidimensional box.

Usage

```
inside.boxx(..., w)
```

Arguments

...	Coordinates of points to be tested. One vector for each dimension (all of same length). (Alternatively, a single point pattern object of class "ppx" or its coordinates as a matrix, data.frame, or "hyperframe")
w	A window. This should be an object of class boxx, or can be given in any format acceptable to as.boxx().

Details

This function tests whether each of the provided points lies inside or outside the window w and returns TRUE if it is inside.

The boundary of the window is treated as being inside.

Normally each argument provided (except w) must be numeric vectors of equal length (length zero is allowed) containing the coordinates of points. Alternatively a single point pattern (object of class "ppx") can be given; then the coordinates of the point pattern are extracted. A single matrix, data.frame, or "hyperframe") with the coordinates is also accepted.

Value

Logical vector whose ith entry is TRUE if the corresponding point is inside w.

Author(s)

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

Rolf Turner <rolfturner@posteo.net>

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

See Also

boxx, as.boxx

Examples

```
# 3D box with side [0,2]
w <- boxx(c(0,2), c(0,2), c(0,2))

# Random points in box with side [-1,3]
x <- runif(30, min=-1, max=3)
y <- runif(30, min=-1, max=3)
z <- runif(30, min=-1, max=3)

# Points falling in smaller box
ok <- inside.boxx(x, y, z, w=w)

# Same using a point pattern as argument:
X <- ppx(data = cbind(x, y, z), domain = boxx(c(0,3), c(0,3), c(0,3)))
ok2 <- inside.boxx(X, w=w)

# Same using the coordinates given as data.frame/matrix/hyperframe
coords_mat <- cbind(x,y,z)
ok_mat <- inside.boxx(coords_mat, w=w)
coords_df <- data.frame(x,y,z)
ok_df <- inside.boxx(coords_mat, w=w)
coords_hyper <- hyperframe(x,y,z)
ok_hyper <- inside.boxx(coords_mat, w=w)
```

---

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 ith entry is TRUE if the corresponding point (x[i],y[i]) is inside w.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

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))
if(human <- interactive()) {
plot(mas)
}

# random points in rectangle
```

```
x <- runif(30,min=-1, max=1)
y <- runif(30,min=-1, max=1)

ok <- inside.owin(x, y, mas)

if(human) {
points(x[ok], y[ok])
points(x[!ok], y[!ok], pch="x")
}
```

---

integral.im *Integral of a Pixel Image*

Description

Computes the integral of a pixel image.

Usage

```
## S3 method for class 'im'
integral(f, domain=NULL, weight=NULL, ...)
```

Arguments

f	A pixel image (object of class "im") with pixel values that can be treated as numeric or complex values.
domain	Optional. Window specifying the domain of integration. Alternatively a tessellation.
...	Ignored.
weight	Optional. A pixel image (object of class "im") or a function(x,y) giving a numerical weight to be applied to the integration.

Details

The function integral is generic, with methods for spatial objects ("im", "msr", "linim", "linfun") and one-dimensional functions ("density", "fv").

The method integral.im treats the pixel image f 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 f 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).

If domain is a window (class "owin") then the integration will be restricted to this window. If domain is a tessellation (class "tess") then the integral of f in each tile of domain will be computed.

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

Value

A single numeric or complex value (or a vector of such values if domain is a tessellation).

Author(s)

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

See Also

integral, 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(Z)
# correct answer is 2

# 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))
integral(Z, W)

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

---

integral.tessfun *Integrate a Function Which is Constant on Each Tile of a Tessellation*

Description

Given a function which is constant on each tile of a tessellation, compute the integral of the function.

Usage

```
 ## S3 method for class 'tessfun'
integral(f, domain = NULL, ...)
```

Arguments

f	Integrand. A function of class "tessfun" (created by as.function.tess).
domain	Optional window (object of class "owin") specifying a subregion to which the integral should be restricted.
...	Ignored.

Details

The command integral is generic. This is the method for objects of class "tessfun".

The integrand f should be a function of class "tessfun" created by as.function.tess. It represents a function which takes a constant value on each tile of a tessellation.

The integral is calculated by multiplying the area of each tile by the value of the function in that tile, and summing. This avoids the need for discretisation and avoids concomitant discretisation errors.

Value

A single numeric value.

Author(s)

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

See Also

as.function.tess.

Examples

```
V <- dirichlet(runifrect(5))
f <- as.function(V, values=1/tile.areas(V))
integral(f) # should be close to 5.
```

---

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

X	A spatial dataset or a spatial point process model.
...	Further arguments depending on the class of X.

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 of class "ppm" is computed by the method intensity.ppm.

Value

Usually a numeric value or vector.

Author(s)

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

See Also

intensity.ppp, intensity.ppm.

Type methods(intensity) to see a list of methods for other classes.

---

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, ..., weights=NULL)

## S3 method for class 'splitppp'
intensity(X, ..., weights=NULL)
```

Arguments

X	A point pattern (object of class "ppp").
weights	Optional. A numeric or logical valued vector, pixel image, function or expression, or one of the strings "x" or "y" representing the cartesian coordinates. The expression may involve the variables x,y,marks representing the coordinates and marks of the point pattern. Numerical weights may be positive, negative or zero.
...	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.

Note that the intensity will be computed as the number of points per square unit, based on the unit of length for X, given by unitname(X). If the unit of length is a strange multiple of a standard unit, like 5.7 metres, then it can be converted to the standard unit using rescale. See the Examples.

If weights are given, then the intensity is computed as the total *weight* per square unit. The argument weights should be a numeric vector of weights for each point of X (weights may be negative or zero).

Alternatively weights can be an expression which will be evaluated for the dataset to yield a vector of weights. The expression may involve the Cartesian coordinates x, y of the points, and the marks of the points, if any. Variable names permitted in the expression include x and y, the name marks if X has a single column of marks, the names of any columns of marks if X has a data frame of marks, and the names of constants or functions that exist in the global environment. See the Examples.

Value

A numeric value (giving the intensity) or numeric vector (giving the intensity for each possible type).

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

intensity, intensity.ppm

Examples

```
japanesepines
intensity(japanesepines)
unitname(japanesepines)
intensity(rescale(japanesepines))

intensity(amacrine)
intensity(split(amacrine))

# numeric vector of weights
volumes <- with(marks(finpines), (pi/4) * height * diameter^2)
intensity(finpines, weights=volumes)

# expression for weights
intensity(finpines, weights=expression((pi/4) * height * diameter^2))
```

`intensity.ppx` *Intensity of a Multidimensional Space-Time Point Pattern*

Description

Calculates the intensity of points in a multi-dimensional point pattern of class `"ppx"` or `"pp3"`.

Usage

```
  ## S3 method for class 'ppx'
intensity(X, ...)
```

Arguments

`X`	Point pattern of class `"ppx"` or `"pp3"`.
`...`	Ignored.

Details

This is a method for the generic function `intensity`. It computes the empirical intensity of a multi-dimensional point pattern (object of class `"ppx"` including `"pp3"`), i.e. the average density of points per unit volume.

If the point pattern is multitype, the intensities of the different types are computed separately.

Value

A single number or a numeric vector.

Author(s)

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

Rolf Turner `<rolfturner@posteo.net>`

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

Examples

```
X <- osteo$pts[[1]]
intensity(X)
marks(X) <- factor(sample(letters[1:3], npoints(X), replace=TRUE))
intensity(X)
```

`intensity.psp` *Empirical Intensity of Line Segment Pattern*

Description

Computes the average total length of segments per unit area in a spatial pattern of line segments.

Usage

```
## S3 method for class 'psp'
intensity(X, ..., weights=NULL)
```

Arguments

`X`	A line segment pattern (object of class `"psp"`).
`weights`	Optional. Numeric vector of weights attached to the segments of `X`. Alternatively, an `expression` which can be evaluated to give a vector of weights.
`...`	Ignored.

Details

This is a method for the generic function `intensity`. It computes the empirical intensity of a line segment pattern (object of class `"psp"`), i.e. the average total segment length per unit area.

If the segment pattern is multitype, the intensities of the different types are computed separately.

Note that the intensity will be computed as the length per area in units per square unit, based on the unit of length for `X`, given by `unitname(X)`. If the unit of length is a strange multiple of a standard unit, like `5.7 metres`, then it can be converted to the standard unit using `rescale`. See the Examples.

If `weights` are given, then the intensity is computed as the total *weight times length* per square unit. The argument `weights` should be a numeric vector of weights for each point of `X` (weights may be negative or zero).

Alternatively `weights` can be an `expression` which will be evaluated for the dataset to yield a vector of weights. The expression may involve the Cartesian coordinates x, y of the points, and the marks of the points, if any. Variable names permitted in the expression include `x0`, `x1`, `y0`, `y1` for the coordinates of the segment endpoint, the name `marks` if `X` has a single column of marks, the names of any columns of marks if `X` has a data frame of marks, and the names of constants or functions that exist in the global environment. See the Examples.

Value

A numeric value (giving the intensity) or numeric vector (giving the intensity for each possible type).

Author(s)

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

See Also

intensity

Examples

```
S <- edges(letterR)
intensity(S)
intensity(S, weights=runif(nsegments(S)))
intensity(S, weights=expression((x0+x1)/2))
```

---

intensity.quadratcount

Intensity Estimates Using Quadrat Counts

Description

Uses quadrat count data to estimate the intensity of a point pattern in each tile of a tessellation, assuming the intensity is constant in each tile.

Usage

```
## S3 method for class 'quadratcount'
intensity(X, ..., image=FALSE)
```

Arguments

X	An object of class "quadratcount".
image	Logical value specifying whether to return a table of estimated intensities (the default) or a pixel image of the estimated intensity (image=TRUE).
...	Arguments passed to as.mask to determine the resolution of the pixel image, if image=TRUE.

Details

This is a method for the generic function intensity. It computes an estimate of the intensity of a point pattern from its quadrat counts.

The argument X should be an object of class "quadratcount". It would have been obtained by applying the function quadratcount to a point pattern (object of class "ppp"). It contains the counts of the numbers of points of the point pattern falling in each tile of a tessellation.

Using this information, intensity.quadratcount divides the quadrat counts by the tile areas, yielding the average density of points per unit area in each tile of the tessellation.

If image=FALSE (the default), these intensity values are returned in a contingency table. Cells of the contingency table correspond to tiles of the tessellation.

If image=TRUE, the estimated intensity function is returned as a pixel image. For each pixel, the pixel value is the estimated intensity in the tile which contains that pixel.

Value

If image=FALSE (the default), a contingency table. If image=TRUE, a pixel image (object of class "im").

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

intensity, quadratcount

Examples

```
qa <- quadratcount(swedishpines, 4,3)
qa
intensity(qa)
plot(intensity(qa, image=TRUE))
```

---

interp.colourmap *Interpolate smoothly between specified colours*

Description

Given a colourmap object which maps numbers to colours, this function interpolates smoothly between the colours, yielding a new colour map.

Usage

```
interp.colourmap(m, n = 512)
```

Arguments

m	A colour map (object of class "colourmap").
n	Number of colour steps to be created in the new colour map.

Details

Given a colourmap object m, which maps numerical values to colours, this function interpolates the mapping, yielding a new colour map.

This makes it easy to build a colour map that has smooth gradation between different colours or shades. First specify a small vector of numbers x which should be mapped to specific colours y. Use m <- colourmap(y, inputs=x) to create a colourmap that represents this simple mapping. Then apply interp.colourmap(m) to obtain a smooth transition between these points.

Value

Another colour map (object of class "colourmap").

Author(s)

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

See Also

colourmap, tweak.colourmap, colourtools.

Examples

```
co <- colourmap(inputs=c(0, 0.5, 1), c("black", "red", "white"))
plot(interp.colourmap(co))
```

---

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=NULL, bilinear=FALSE)
```

Arguments

Z	Pixel image (object of class "im") with numeric or integer values.
x, y	Vectors of Cartesian coordinates. Alternatively x can be a point pattern and y can be missing.
bilinear	Logical value specifying the choice of interpolation rule. If bilinear=TRUE then a bilinear interpolation rule is used. If bilinear=FALSE (the default) then a slightly biased rule is used; this rule is consistent with earlier versions of **spatstat**.

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.

The arguments x,y can be anything acceptable to xy.coords.

Value

Vector of interpolated values, with NA for points that lie outside the domain of the image.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>, with a contribution from an anonymous user.

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)
interp.im(V, 0.5, 0.5, bilinear=TRUE)
# 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.boxx *Intersection Of Boxes Of Arbitrary Dimension*

Description

Yields the intersection of boxes of arbitrary dimension (of class "boxx").

Usage

```
intersect.boxx(..., fatal=FALSE)
```

Arguments

...	Boxes (of class "boxx").
fatal	Logical. Determines what happens if the intersection is empty: If true

Details

If the intersection is empty, then if fatal=FALSE the result is NULL, while if fatal=TRUE an error occurs.

Value

A box (object of class "boxx") or possibly NULL.

Author(s)

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

See Also

intersect.owin, boxx

Examples

```
intersect.boxx(boxx(c(-1,1),c(0,2)), boxx(c(0,3),c(0,1)))
```

---

intersect.owin *Intersection, Union or Set Subtraction of Windows*

Description

Yields the intersection, union or set subtraction of windows.

Usage

```
intersect.owin(..., fatal=FALSE, rescue=FALSE, p)
union.owin(..., rescue=FALSE, p)
setminus.owin(A, B, ..., rescue=FALSE, p)
```

Arguments

A, B	Windows (objects of class "owin").
...	Windows, or arguments passed to as.mask to control the discretisation.
fatal	Logical. Determines what happens if the intersection is empty.
rescue	Logical value indicating whether a result which is equivalent to a rectangle should be returned as a rectangular window.
p	Optional list of parameters passed to polyclip to control the accuracy of polygon geometry.

Details

The function intersect.owin computes the intersection between the windows given in ..., while union.owin computes their union. The function setminus.owin computes the intersection of A with the complement of B.

For intersect.owin and union.owin, the arguments ... must be either

- window objects of class "owin",
- data that can be coerced to this class by as.owin),
- lists of windows, of class "solist",
- named arguments of as.mask to control the discretisation if required.

For setminus.owin, the arguments ... must be named arguments of as.mask.

If the intersection is empty, then if fatal=FALSE the result is an empty window or NULL, while if fatal=TRUE an error occurs.

If rescue=TRUE then rescue.rectangle will be applied to check whether the result is equivalent to a rectangle, in which case the equivalent rectangle will be returned.

Value

A window (object of class "owin") or possibly NULL.

Author(s)

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

See Also

is.subset.owin, overlap.owin, is.empty, boundingbox, owin.object

Examples

```
# rectangles
   u <- unit.square()
   v <- owin(c(0.5,3.5), c(0.4,2.5))
# polygon
   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(boundingbox(A,B), main="intersection")
   w <- intersect.owin(A, B)
   plot(w, add=TRUE, col="lightblue")
   plot(A, add=TRUE)
   plot(B, add=TRUE)

   plot(boundingbox(A,B), main="union")
   w <- union.owin(A,B)
   plot(w, add=TRUE, col="lightblue")
   plot(A, add=TRUE)
   plot(B, add=TRUE)

   plot(boundingbox(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, ..., keepempty=FALSE, keepmarks=FALSE, sep="x")
```

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.
keepempty	Logical value specifying whether empty intersections between tiles should be retained (keepempty=TRUE) or deleted (keepempty=FALSE, the default).
keepmarks	Logical value. If TRUE, the marks attached to the tiles of X and Y will be retained as marks of the intersection tiles.
sep	Character string used to separate the names of tiles from X and from Y, when forming the name of the tiles of the intersection.

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

See Also

tess, as.tess, intersect.owin

Examples

```
  opa <- par(mfrow=c(1,3))
# polygon
  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 <- runifrect(10)
  B <- runifrect(10)
  plot(DA <- dirichlet(A))
  plot(DB <- dirichlet(B))
  plot(intersect.tess(DA, DB))
```

```
par(opa)

marks(DA) <- 1:10
marks(DB) <- 1:10
plot(Z <- intersect.tess(DA,DB, keepmarks=TRUE))
mZ <- marks(Z)
tZ <- tiles(Z)
for(i in which(mZ[,1] == 3)) plot(tZ[[i]], add=TRUE, col="pink")
```

`invoke.metric` *Perform Geometric Task using a Specified Metric*

Description

Perform a desired geometrical operation using a specified distance metric.

Usage

```
invoke.metric(m, task, ..., evaluate=TRUE)
```

Arguments

`m`	Metric (object of class `"metric"`)
`task`	Character string specifying the task. The name of a function that performs the desired operation for the Euclidean metric.
`...`	Input to the function that performs the geometrical operation (matching the arguments of `task`).
`evaluate`	Logical value specifying whether to actually perform the computation and return the result (`evaluate=TRUE`, the default) or to simply return the function which performs the computation (`evaluate=FALSE`).

Details

A 'metric' is a measure of distance between points in space. An object of class `"metric"` represents such a metric, and supports many geometrical computations that involve the metric. See `metric.object`.

The argument `task` should be the name of an existing function in the **spatstat** family representing a geometrical operation, such as computing pairwise distances, nearest-neighbour distances, the distance map, and so on. The code will determine whether this geometrical operation has a counterpart using the specified metric, that is defined and supported in the object `m`. If so, then this operation will be applied to the data specified in `...`, and the result will be returned.

For example, the **spatstat** function `nndist.ppp` computes nearest-neighbour distances using the Euclidean distance metric. To calculate nearest-neighbour distances for a point pattern `X` using another metric `m`, use `invoke.metric(m, "nndist.ppp", X)`.

If `evaluate=FALSE`, the computation is not performed, and `invoke.metric` simply returns a function to perform the desired operation.

Value

If `evaluate=TRUE` (the default), the result of the computation has the same format as the result of the computation using the existing function named `task`.

If `evaluate=FALSE`, the result is a `function` in the R language to perform the desired operation; or `NULL` if the operation is not supported by the metric.

Author(s)

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

See Also

convexmetric

Examples

```
## nearest-neighbour distances using rectangular metric (L^1 metric)
d <- convexmetric(square(c(-1,1)))
y <- invoke.metric(d, "nndist.ppp", cells)
f <- invoke.metric(d, "nndist.ppp", cells, evaluate=FALSE)
y <- f(cells)
invoke.metric(d, "orderPizza", evaluate=FALSE)
```

---

invoke.symbolmap *Plot Data Using Graphics Symbol Map*

Description

Apply a graphics symbol map to a vector of data values and plot the resulting symbols.

Usage

```
invoke.symbolmap(map, values, x=NULL, y = NULL, ...,
                 angleref=NULL,
                 add = FALSE,
                 do.plot = TRUE, started = add && do.plot)
```

Arguments

`map`	Graphics symbol map (object of class `"symbolmap"`).
`values`	Vector of data that can be mapped by the symbol map.
`x, y`	Coordinate vectors for the spatial locations of the symbols to be plotted.
`...`	Additional graphics parameters (which will be applied to the entire plot).
`angleref`	Optional. Reference angle, or vector of reference angles, used when plotting some of the symbols. A numeric value or vector giving angles in degrees between 0 and 360.

add	Logical value indicating whether to add the symbols to an existing plot (add=TRUE) or to initialise a new plot (add=FALSE, the default).
do.plot	Logical value indicating whether to actually perform the plotting.
started	Logical value indicating whether the plot has already been initialised.

Details

A symbol map is an association between data values and graphical symbols.

This command applies the symbol map map to the data values and plots the resulting symbols at the locations given by xy.coords(x,y).

Value

(Invisibly) the maximum diameter of the symbols, in user coordinate units.

Author(s)

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

See Also

plot.symbolmap to plot the graphics map itself.

symbolmap to create a graphics map.

Examples

```
g <- symbolmap(range=c(-1,1),
                 shape=function(x) ifelse(x > 0, "circles", "squares"),
                 size=function(x) sqrt(ifelse(x > 0, x/pi, -x))/15,
                 bg=function(x) ifelse(x > 0, "green", "red"))
plot(square(1), main="")
a <- invoke.symbolmap(g, runif(10, -1, 1), runifrect(10), add=TRUE)
a
```

---

is.boxx *Recognise a Multi-Dimensional Box*

Description

Checks whether its argument is a multidimensional box (object of class "boxx").

Usage

```
is.boxx(x)
```

Arguments

x	Any object.

Details

This function tests whether the object x is a multidimensional box of class "boxx".

The result is determined to be TRUE if x inherits from "boxx", i.e. if x has "boxx" amongst its classes.

Value

A logical value.

Author(s)

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

See Also

methods.boxx, boxx.

Examples

```
B <- boxx(c(0,10),c(0,10),c(0,5),c(0,1), unitname="km")
is.boxx(B)
is.boxx(42)
```

---

is.connected *Determine Whether an Object is Connected*

Description

Determine whether an object is topologically connected.

Usage

```
is.connected(X, ...)

## Default S3 method:
is.connected(X, ...)
```

Arguments

X	A spatial object such as a pixel image (object of class "im"), or a window (object of class "owin").
...	Arguments passed to connected to determine the connected components.

Details

The command `is.connected(X)` returns `TRUE` if the object `X` consists of a single, topologically-connected piece, and returns `FALSE` if `X` consists of several pieces which are not joined together.

The function `is.connected` is generic. The default method `is.connected.default` works for many classes of objects, including windows (class `"owin"`) and images (class `"im"`). There is a method for point patterns, described in `is.connected.ppp`.

Value

A logical value.

Author(s)

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

See Also

`connected`, `is.connected.ppp`.

Examples

```
d <- distmap(cells, dimyx=256)
X <- levelset(d, 0.07)
plot(X)
is.connected(X)
```

---

is.connected.ppp *Determine Whether a Point Pattern is Connected*

Description

Determine whether a point pattern is topologically connected when all pairs of points closer than a threshold distance are joined.

Usage

```
## S3 method for class 'ppp'
is.connected(X, R, ...)
```

Arguments

`X`	A point pattern (object of class `"ppp"`).
`R`	Threshold distance. Pairs of points closer than `R` units apart will be joined together.
`...`	Ignored.

Details

The function `is.connected` is generic. This is the method for point patterns (objects of class `"ppp"`).

The point pattern `X` is first converted into an abstract graph by joining every pair of points that lie closer than `R` units apart. Then the algorithm determines whether this graph is connected.

That is, the result of `is.connected(X)` is `TRUE` if any point in `X` can be reached from any other point, by a series of steps between points of `X`, each step being shorter than `R` units in length.

Value

A logical value.

Author(s)

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

See Also

`is.connected`, `connected.ppp`.

Examples

```
is.connected(redwoodfull, 0.1)
is.connected(redwoodfull, 0.2)
```

---

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

---

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

is.linim *Test Whether an Object is a Pixel Image on a Linear Network*

Description

Tests whether its argument is a pixel image on a linear network (object of class "linim").

Usage

```
is.linim(x)
```

Arguments

x	Any object.

Details

This function tests whether the argument x is a pixel image on a linear network (object of class "linim").

The object is determined to be an image if it inherits from class "linim".

Value

TRUE if x is a pixel image on a linear network, otherwise FALSE.

Author(s)

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

is.linnet *Test Whether An Object Is A Linear Network*

Description

Checks whether its argument is a linear network (object of class "linnet").

Usage

```
is.linnet(x)
```

Arguments

x	Any object.

Details

This function tests whether the object x is a linear network (object of class "linnet").

Value

TRUE if x is of class "linnet", otherwise FALSE.

Author(s)

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

---

is.lpp *Test Whether An Object Is A Point Pattern on a Linear Network*

Description

Checks whether its argument is a point pattern on a linear network (object of class "lpp").

Usage

```
is.lpp(x)
```

Arguments

x	Any object.

Details

This function tests whether the object x is a point pattern object of class "lpp".

Value

TRUE if x is a point pattern of class "lpp", otherwise FALSE.

Author(s)

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

`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@curtin.edu.au>`

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

See Also

`is.marked.ppp`, `is.marked.ppm`

is.marked.ppp *Test Whether A Point Pattern is Marked*

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

See Also

is.marked, is.marked.ppm

Examples

```
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@curtin.edu.au>`

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

See Also

`is.multitype.ppp`, `is.multitype.ppm`

`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@curtin.edu.au>`

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

See Also

`is.multitype`, `is.multitype.ppm`

Examples

```
is.multitype(cells)  #FALSE - no marks
is.multitype(longleaf) #FALSE - real valued marks
is.multitype(amacrine) #TRUE
```

`is.na.hyperframe` *Identify Missing Entries in a Hyperframe*

Description

Given a hyperframe, this function returns a logical matrix specifying which entries of the hyperframe are missing.

Usage

```
## S3 method for class 'hyperframe'
is.na(x)
```

Arguments

`x`	A hyperframe (object of class `"hyperframe"`).

Details

This function returns a logical matrix, with the same dimensions as the hyperframe `x`, with the value `TRUE` in each position where the corresponding entry in the hyperframe is missing.

An entry `x[i,j]` is deemed to be missing if it is either `NA` (representing a missing value in an atomic vector) or is an object of class `"NAobject"` (representing a missing object of a particular class in the **spatstat** package family).

Value

A logical matrix with the same dimensions as `x`.

Author(s)

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

See Also

`NAobject`

Examples

```
A <- c(1, NA, 3)
B <- solist(cells, cells, NA)
h <- hyperframe(A=A, B=B)
h
is.na(h)
```

`is.na.solist` *Identify Missing Entries in a List of Spatial Objects*

Description

Given a list of spatial objects (of class `"solist"`), this function returns a logical vector specifying which entries of the list are missing.

Usage

```
## S3 method for class 'solist'
is.na(x)
```

Arguments

`x`	A list of spatial objects (an object of class `"solist"`).

Details

This function returns a logical vector with the same length as the list `x`, with the value `TRUE` in each position where the corresponding entry in the list is missing.

An entry `x[[i]]` is deemed to be missing if it is an object of class `"NAobject"` (representing a missing object of a particular class in the **spatstat** package family).

Value

A logical vector with the same length as `x`.

Author(s)

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

See Also

`NAobject`

Examples

```
B <- solist(cells, NA, cells)
B[[2]]
is.na(B)
V <- solist(cells, Window(cells), NAobject("im"))
is.na(V)
```

`is.NAobject` *Recognise NA Objects*

Description

Recognises whether an object is an NA object (representing a missing or unavailable object in the **spatstat** package family).

Usage

```
is.NAobject(x)
```

Arguments

`x`	An object.

Details

This function recognises whether an object is a missing or unavailable object belonging to one of the classes in the **spatstat** package family.

In **spatstat**, a missing or unavailable object of class `"foo"` is represented by an object that inherits the classes `"foo"` and `"NAobject"`.

For any object `x`, the command `is.NAobject(x)` will return `TRUE` if `x` is a missing or unavailable object, and `FALSE` otherwise.

Value

A single logical value.

Author(s)

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

See Also

`NAobject`

Examples

```
A <- NAobject("ppp")
is.NAobject(A)

X <- solist(cells, NA, cells)
is.NAobject(X[[2]])
sapply(X, is.NAobject)
```

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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 ppp.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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

---

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

owin

---

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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.
```

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, LayerList=NULL)
```

Arguments

...	Objects which can be plotted by plot.
plotargs	Default values of the plotting arguments for each of the objects. A list of lists of arguments of the form name=value.
LayerList	A list of objects. Incompatible with ....

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, to control the plotting arguments that are passed to each layer, and to zoom in.

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.

The plotargs can also include an argument named .plot specifying (the name of) a function to perform the plotting instead of the generic plot.

The length of plotargs should either be equal to the number of layers, or equal to 1. In the latter case it will be replicated to the appropriate length.

Value

A list, belonging to the class "layered". There are methods for plot, "[", "shift", "affine", "rotate" and "rescale".

There is a method for c which makes it possible to concatenate two or more layered objects.

Author(s)

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

See Also

plot.layered, methods.layered, as.layered, [.layered, layerplotargs.

Examples

```
D <- distmap(cells)
L <- layered(D, cells)
L
L <- layered(D, cells,
 plotargs=list(list(ribbon=FALSE), list(pch=16)))
plot(L)

layerplotargs(L)[[1]] <- list(.plot="contour")
plot(L)
```

---

layerplotargs *Extract or Replace the Plot Arguments of a Layered Object*

Description

Extracts or replaces the plot arguments of a layered object.

Usage

```
layerplotargs(L)

layerplotargs(L) <- value
```

Arguments

L	An object of class "layered" created by the function layered.
value	Replacement value. A list, with the same length as L, whose elements are lists of plot arguments.

Details

These commands extract or replace the plotargs in a layered object. See layered.

The replacement value should normally have the same length as the current value. However, it can also be a list with *one* element which is a list of parameters. This will be replicated to the required length.

For the assignment function layerplotargs<-, the argument L can be any spatial object; it will be converted to a layered object with a single layer.

Value

`layerplotargs` returns a list of lists of plot arguments.

`"layerplotargs<-"` returns the updated object of class `"layered"`.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

`layered`, `methods.layered`, `[.layered`.

Examples

```
W <- square(2)
L <- layered(W=W, X=cells)
## The following are equivalent
layerplotargs(L) <- list(list(), list(pch=16))
layerplotargs(L)[[2]] <- list(pch=16)
layerplotargs(L)$X <- list(pch=16)

## The following are equivalent
layerplotargs(L) <- list(list(cex=2), list(cex=2))
layerplotargs(L) <- list(list(cex=2))
```

---

layout.boxes *Generate a Row or Column Arrangement of Rectangles.*

Description

A simple utility to generate a row or column of boxes (rectangles) for use in point-and-click panels.

Usage

```
layout.boxes(B, n, horizontal = FALSE, aspect = 0.5, usefrac = 0.9)
```

Arguments

`B`	Bounding rectangle for the boxes. An object of class `"owin"`.
`n`	Integer. The number of boxes.
`horizontal`	Logical. If `TRUE`, arrange the boxes in a horizontal row. If `FALSE` (the default), arrange them in a vertical column.
`aspect`	A single finite positive number, giving the aspect ratio (height divided by width) of each box, or `NA` or `Inf`, indicating that the aspect ratio is unconstrained.
`usefrac`	Number between 0 and 1. The fraction of height or width of `B` that should be occupied by boxes.

Details

This simple utility generates a list of boxes (rectangles) inside the bounding box B arranged in a regular row or column. It is useful for generating the positions of the panel buttons in the function simplepanel.

The argument aspect specifies the ratio of height to width (height divided by width). If aspect is a finite numerical value, then the boxes will have the given aspect ratio. If aspect is Inf or NA, aspect ratio is unconstrained; the boxes will have the maximum possible width and height.

Value

A list of rectangles (objects of class "owin" which are rectangles).

Author(s)

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

See Also

simplepanel

Examples

```
B <- owin(c(0,10),c(0,1))
boxes <- layout.boxes(B, 5, horizontal=TRUE)
plot(B, main="", col="blue")
niets <- lapply(boxes, plot, add=TRUE, col="grey")
```

---

lengths_psp *Lengths of Line Segments*

Description

Computes the length of each line segment in a line segment pattern.

Usage

```
lengths_psp(x, squared=FALSE)
```

Arguments

x	A line segment pattern (object of class "psp").
squared	Logical value indicating whether to return the squared lengths (squared=TRUE) or the lengths themselves (squared=FALSE, the default).

Details

The length of each line segment is computed and the lengths are returned as a numeric vector.

Using squared lengths may be more efficient for some purposes, for example, to find the length of the shortest segment, `sqrt(min(lengths.psp(x, squared=TRUE)))` is faster than `min(lengths.psp(x))`.

Value

Numeric vector.

Change of name

The name of this function has changed from `lengths.psp` to `lengths_psp`, because the old name `lengths.psp` could be misinterpreted as a method for `lengths`.

The older function name `lengths.psp` is retained temporarily, for consistency with older code and documentation.

In future versions of **spatstat**, the function name `lengths.psp` will be removed. The newer function name `lengths_psp` should be used.

Author(s)

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

See Also

`marks.psp`, `summary.psp`, `midpoints.psp`, `angles.psp`, `endpoints.psp`, `extrapolate.psp`.

Examples

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

---

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="<=", ...)

## S3 method for class 'im'
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 "<", ">", "==", "<=", ">=", "!=".
...	Arguments passed to other methods (ignored by levelset.im)

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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(levelset(X, 0.1))
```

levelset.distfun *Level Set of a Distance Function*

Description

Given a distance function, find the spatial region where the function value is less than a specified threshold value (or greater than a threshold, etc),

Usage

```
## S3 method for class 'distfun'
levelset(X, thresh, compare="<=", ...)
```

Arguments

X	Object of class "distfun" representing the distance function of a spatial object.
thresh	Single numeric value of distance, defining the level set.
compare	Character string specifying one of the comparison operators "<", ">", "==", "<=", ">=", "!=".
...	Arguments passed to as.mask if it is necessary to convert the result to a binary mask.

Details

This is a method for the generic function levelset which computes level sets of a spatial variable.

A distance function (object of class "distfun" created by distfun) is a function of spatial location which measures the distance to a given spatial object.

If f <- distfun(B), where B is some object, then the level set of f consisting of all spatial locations u where f(u) <= thresh, is simply the dilation of the original object B by a distance equal to thresh. The command levelset(X, thresh) computes this dilation as a spatial window.

In some cases the object must be converted to a binary pixel mask and the computation performed by levelset.im. In that case, any arguments ... are passed to as.mask to determine the pixel resolution.

Value

A spatial window (object of class "win").

Author(s)

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

See Also

distfun, methods.distfun, levelset

Examples

```
f <- distfun(cells)
V <- levelset(f, 0.05)
plot(V)
plot(cells, add=TRUE)
```

---

lut *Lookup Tables*

Description

Create a lookup table.

Usage

```
lut(outputs, ..., range=NULL, breaks=NULL, inputs=NULL,
    gamma=1, compress=NULL, decompress=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 length(outputs)+1. Incompatible with range or inputs.
gamma	Exponent for gamma correction, when range is given. A single positive number. See Details.
compress	Optional. Experimental. An R function determining a nonlinear transformation of the domain of the lookup table. See section on *Nonlinear lookup tables*.
decompress	Experimental. An R function giving the inverse function of compress, if compress is specified.

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]`. The argument `outputs` should have the same length as `inputs`.
- 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. The interval will be divided evenly into bands, each of which is mapped to an entry of `outputs`. (If `gamma` is given, then the bands are equally spaced on a scale where the original values are raised to the power `gamma`.) (See the section on *Nonlinear lookup tables* for the case where `compress` and `decompress` are given.)
- 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 argument `outputs` should have length equal to `length(breaks) - 1`.

It is also permissible for `outputs` to be a single value, representing a trivial lookup table in which all data values are mapped to the same output value.

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"`.

Nonlinear lookup tables

If the arguments `compress` and `decompress` are given, they define a transformation of the range of numbers. A typical example would be a logarithmic lookup table defined by `compress = log10` and `decompress = function(x) { 10^xg }`.

These functions have no effect on the interpretation of the arguments `range`, `breaks` and `inputs`. However, if `range` is given, then the range of values will be divided into intervals which have equal length on the scale defined by `compress`. That is, the range of numbers determined by `compress(range)` will be evenly divided into intervals, and these intervals will be mapped back to the original scale by `decompress` to determine the `breaks`. For a logarithmic lookup table with `compress=log10`, the specified range will be divided into intervals which are equal on a logarithmic scale.

The arguments `compress` and `decompress` should be functions which are vectorised (i.e. if `x` is a vector then `compress(x)` and `decompress(x)` are also vectors of the same length as `x`) and increasing (if `x < y` then `compress(x) < compress(y)` and `decompress(x) < decompress(y)`.

The argument `decompress` is not needed in the following cases:

- If `compress` is the function `log10`, then `decompress` is taken to be its inverse `function(x) { 10^x }`.
- If `compress` is a cumulative distribution function (of class `"ecdf"`, `"ewcdf"` or `"interpolatedCDF"`) then `decompress` is taken to be its inverse function `decompress = quantilefun(compress)`.

Author(s)

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

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)
## logarithmic lookup table
cl <- lut(letters[1:3], range=c(0.1, 100), compress=log10)
```

---

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, drop=TRUE)

## S3 method for class 'ppx'
marks(x, ..., drop=TRUE)

marks(x, ...) <- value

## S3 replacement method for class 'ppp'
marks(x, ..., dfok=TRUE, drop=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	Replacement value. A 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@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>

See Also

ppp.object, ppx, unmark, hyperframe

Examples

```
X <- amacrine
# extract marks
m <- marks(X)
# recode the mark values "off", "on" as 0, 1
marks(X) <- 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

`x`	Line segment pattern dataset (object of class `"psp"`).
`...`	Ignored.
`dfok`	Logical. If `FALSE`, data frames of marks are not permitted and will generate an error.
`value`	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@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

`psp.object`, `marks`, `marks<-`

Examples

```
m <- data.frame(A=1:10, B=letters[1:10])
X <- psp(runif(10), runif(10), runif(10), runif(10), window=owin(), marks=m)

marks(X)
marks(X)[,2]
marks(X) <- 42
marks(X) <- NULL
```

---

marks.tess *Marks of a Tessellation*

Description

Extract or change the marks attached to the tiles of a tessellation.

Usage

```
## S3 method for class 'tess'
marks(x, ...)

## S3 replacement method for class 'tess'
marks(x, ...) <- value

## S3 method for class 'tess'
unmark(X)
```

Arguments

`x`, `X`	Tessellation (object of class `"tess"`)
`...`	Ignored.
`value`	Vector or data frame of mark values, or `NULL`.

Details

These functions extract or change the marks attached to each of the tiles in the tessellation x. They are methods for the generic functions marks, marks<- and unmark for the class "tess" of tessellations

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, 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 tiles in x, or a data frame or hyperframe with as many rows as there are tiles 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 tile.

To remove marks, use marks(x) <- NULL or unmark(x).

Value

For marks(x), the result is a vector, factor, data frame or hyperframe, containing the mark values attached to the tiles of x. If there are no marks, the result is NULL.

For unmark(x), the result is the tessellation without marks.

For marks(x) <- value, the result is the updated tessellation x (with the side-effect that the dataset x is updated in the current environment).

Author(s)

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

See Also

marks, marks<-

Examples

```
D <- dirichlet(cells)
marks(D) <- tile.areas(D)
```

---

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=NULL, R=NULL, ...)
```

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.
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.
...	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 ... 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 the section on **Value**.

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 npoints(X), 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 npoints(X).

Author(s)

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

See Also

applynbd, marktable, ppp.object, apply

Examples

```
trees <- longleaf


# average diameter of 5 closest neighbours of each tree
md <- markstat(trees, mean, N=5)

# range of diameters of trees within 10 metre radius
rd <- markstat(trees, range, R=10)
```

---

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@mathematik.uni-goettingen.de>, URL http://dominic.schuhmache

See Also

pppdist pppmatching.object

Examples

```
# an optimal matching
X <- runifrect(20)
Y <- runifrect(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 <- pppmatching(X, Y, am)
matchingdist(m.nn, type="spa", cutoff=1, q=1)
     # is >= the distance obtained for m.opt
     # in most cases strictly >

opa <- par(mfrow=c(1,2))
```

```
plot(m.opt, main="optimal")
plot(m.nn,  main="nearest neighbour")
text(X, 1:20, pos=1, offset=0.3, cex=0.8)
par(opa)
```

`Math.im` *S3 Group Generic methods for images*

Description

These are group generic methods for images of class `"im"`, which allows for usual mathematical functions and operators to be applied directly to images. See Details for a list of implemented functions.

Usage

```
## S3 methods for group generics have prototypes:
Math(x, ...)
Ops(e1, e2)
Complex(z)
Summary(..., na.rm=FALSE, drop=TRUE)
```

Arguments

`x`, `z`, `e1`, `e2`	objects of class `"im"`.
`...`	further arguments passed to methods.
`na.rm`, `drop`	Logical values specifying whether missing values should be removed. This will happen if either `na.rm=TRUE` or `drop=TRUE`. See Details.

Details

Below is a list of mathematical functions and operators which are defined for images. Not all functions will make sense for all types of images. For example, none of the functions in the `"Math"` group make sense for character-valued images. Note that the `"Ops"` group methods are implemented using `eval.im`, which tries to harmonise images via `harmonise.im` if they aren't compatible to begin with.

1. Group `"Math"`:
   - `abs`, `sign`, `sqrt`, `floor`, `ceiling`, `trunc`, `round`, `signif`
   - `exp`, `log`, `expm1`, `log1p`, `cos`, `sin`, `tan`, `cospi`, `sinpi`, `tanpi`, `acos`, `asin`, `atan` `cosh`, `sinh`, `tanh`, `acosh`, `asinh`, `atanh`

- `lgamma`, `gamma`, `digamma`, `trigamma`
- `cumsum`, `cumprod`, `cummax`, `cummin`

2. Group `"Ops"`:
   - `"+"`, `"-"`, `"*"`, `"/"`, `"^"`, `"%%"`, `"%/%"`
   - `"&"`, `"|"`, `"!"`
   - `"=="`, `"!="`, `"<"`, `"<="`, `">="`, `">"`
3. Group `"Summary"`:
   - `all`, `any`
   - `sum`, `prod`
   - `min`, `max`
   - `range`
4. Group `"Complex"`:
   - `Arg`, `Conj`, `Im`, `Mod`, `Re`

For the `Summary` group, the generic has an argument `na.rm=FALSE`, but for pixel images it makes sense to set `na.rm=TRUE` so that pixels outside the domain of the image are ignored. To enable this, we added the argument `drop`. Pixel values that are `NA` are removed if `drop=TRUE` or if `na.rm=TRUE`.

For the `Ops` group, one of the arguments is permitted to be a single atomic value instead of an image.

Author(s)

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

See Also

`eval.im` for evaluating expressions involving images.

Examples

```
## Convert gradient values to angle of inclination:
V <- atan(bei.extra$grad) * 180/pi
## Make logical image which is TRUE when heat equals 'Moderate':
A <- (gorillas.extra$heat == "Moderate")
## Summary:
any(A)
## Complex:
Z <- exp(1 + V * 1i)
Z
Re(Z)
```

Math.imlist *S3 Group Generic methods for List of Images*

Description

These are group generic methods for the class "imlist" of lists of images. These methods allows the usual mathematical functions and operators to be applied directly to lists of images. See Details for a list of implemented functions.

Usage

```
## S3 methods for group generics have prototypes:
Math(x, ...)
Ops(e1, e2)
Complex(z)
Summary(..., na.rm = TRUE)
```

Arguments

x, z, e1, e2	Lists of pixel images (objects of class "imlist").
...	further arguments passed to methods.
na.rm	logical: should missing values be removed?

Details

An object of class "imlist" represents a list of pixel images. It is a list, whose entries are pixel images (objects of class "im").

The following mathematical functions and operators are defined for lists of images.

Not all functions will make sense for all types of images. For example, none of the functions in the "Math" group make sense for character-valued images. Note that the "Ops" group methods are implemented using eval.im, which tries to harmonise images via harmonise.im if they aren't compatible to begin with.

1. Group "Math":
   - abs, sign, sqrt,
     floor, ceiling, trunc,
     round, signif
   - exp, log, expm1, log1p,
     cos, sin, tan,
     cospi, sinpi, tanpi,
     acos, asin, atan
     cosh, sinh, tanh,
     acosh, asinh, atanh
   - lgamma, gamma, digamma, trigamma
   - cumsum, cumprod, cummax, cummin

2. Group "Ops":
   - "+", "-", "*", "/", "^", "%%", "%/%"
   - "&", "|", "!"
   - "==", "!=", "<", "<=", ">=", ">"
3. Group "Summary":
   - all, any
   - sum, prod
   - min, max
   - range
4. Group "Complex":
   - Arg, Conj, Im, Mod, Re

For the binary operations in "Ops", either

- e1 and e2 are lists of pixel images, and contain the same number of images.
- one of e1,e2 is a list of pixel images, and the other is a single atomic value.

Value

The result of "Math", "Ops" and "Complex" group operations is another list of images. The result of "Summary" group operations is a numeric vector of length 1 or 2.

Author(s)

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

See Also

Math.im or eval.im for evaluating expressions involving images. solapply for a wrapper for lapply.

Examples

```
a <- solist(A=setcov(square(1)), B=setcov(square(2)))
log(a)/2 - sqrt(a)
range(a)
```

maxnndist *Compute Minimum or Maximum Nearest-Neighbour Distance*

Description

A faster way to compute the minimum or maximum nearest-neighbour distance in a point pattern.

Usage

```
minnndist(X, positive=FALSE, by=NULL)
maxnndist(X, positive=FALSE, by=NULL)
```

Arguments

X	A point pattern (object of class "ppp").
positive	Logical. If FALSE (the default), compute the usual nearest-neighbour distance. If TRUE, ignore coincident points, so that the nearest neighbour distance for each point is greater than zero.
by	Optional. A factor, which separates X into groups. The algorithm will compute the distance to the nearest point in each group.

Details

These functions find the minimum and maximum values of nearest-neighbour distances in the point pattern X. minnndist(X) and maxnndist(X) are equivalent to, but faster than, min(nndist(X)) and max(nndist(X)) respectively.

The value is NA if npoints(X) < 2.

Value

A single numeric value (possibly NA).

If by is given, the result is a numeric matrix giving the minimum or maximum nearest neighbour distance between each subset of X.

Author(s)

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

See Also

nndist

Examples

```
min(nndist(swedishpines))
minnndist(swedishpines)

max(nndist(swedishpines))
maxnndist(swedishpines)

minnndist(lansing, positive=TRUE)

if(interactive()) {
   X <- runifrect(1e6)
   system.time(min(nndist(X)))
   system.time(minnndist(X))
}

minnndist(amacrine, by=marks(amacrine))
maxnndist(amacrine, by=marks(amacrine))
```

`mean.im` *Mean and Median of Pixel Values in an Image*

Description

Calculates the mean or median of the pixel values in a pixel image.

Usage

```
## S3 method for class 'im'
## mean(x, trim=0, na.rm=TRUE, ...)

## S3 method for class 'im'
## median(x, na.rm=TRUE)        [R < 3.4.0]
## median(x, na.rm=TRUE, ...)   [R >= 3.4.0]
```

Arguments

`x`	A pixel image (object of class `"im"`).
`na.rm`	Logical value indicating whether `NA` values should be stripped before the computation proceeds.
`trim`	The fraction (0 to 0.5) of pixel values to be trimmed from each end of their range, before the mean is computed.
`...`	Ignored.

Details

These functions calculate the mean and median 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.

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 is are not defined.

Other mathematical operations on images are supported by Math.im, Summary.im and Complex.im.

Other information about an image can be obtained using summary.im or quantile.im.

Value

A single number.

Author(s)

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

See Also

Math.im for other operations.

Generics and default methods: mean, median.

quantile.im, anyNA.im, im.object, summary.im.

Examples

```
X <- as.im(function(x,y) {x^2}, unit.square())
mean(X)
median(X)
mean(X, trim=0.05)
```

---

mergeLevels *Merge Levels of a Factor*

Description

Specified levels of the factor will be merged into a single level.

Usage

```
mergeLevels(.f, ...)
```

Arguments

.f	A factor (or a factor-valued pixel image or a point pattern with factor-valued marks).
...	List of name=value pairs, where name is the new merged level, and value is the vector of old levels that will be merged.

Details

This utility function takes a factor .f and merges specified levels of the factor.

The grouping is specified by the arguments ... which must each be given in the form new=old, where new is the name for the new merged level, and old is a character vector containing the old levels that are to be merged.

The result is a new factor (or factor-valued object), in which the levels listed in old have been replaced by a single level new.

An argument of the form name=character(0) or name=NULL is interpreted to mean that all other levels of the old factor should be mapped to name.

Value

Another factor of the same length as .f (or object of the same kind as .f).

Tips for manipulating factor levels

To remove unused levels from a factor f, just type f <- factor(f).

To change the ordering of levels in a factor, use factor(f, levels=l) or relevel(f, ref).

Author(s)

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

Rolf Turner <rolfturner@posteo.net>

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

See Also

factor, relevel

Examples

```
likert <- c("Strongly Agree", "Agree", "Neutral",
            "Disagree", "Strongly Disagree")
answers <- factor(sample(likert, 15, replace=TRUE), levels=likert)
answers
mergeLevels(answers, Positive=c("Strongly Agree", "Agree"),
                     Negative=c("Strongly Disagree", "Disagree"))
```

mergeTiles *Merge Groups of Tiles in a Tessellation*

Description

Given a tessellation and a factor which classifies the tiles of the tessellation into groups, form the set union of each group of tiles, and construct the tessellation based on these merged tiles.

Usage

```
mergeTiles(x, group, ...)

## S3 method for class 'tess'
mergeTiles(x, group, ...)
```

Arguments

x	A tessellation (object of class "tess").
group	A factor, with one entry for each tile of x, which classifies the tiles into groups.
...	Other arguments, currently ignored.

Value

A tessellation.

Author(s)

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

See Also

tess

Examples

```
D <- dirichlet(cells)
g <- factor(sample(letters[1:4], npoints(cells), replace=TRUE))
plot(D, values=g, main="tiles to be merged")
plot(mergeTiles(D, g), do.col=TRUE, "merged")
```

`methods.box3` *Methods for Three-Dimensional Box*

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 print.default.
value	Name of the unit of length. See unitname.

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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"
```

`methods.boxx` *Methods for Multi-Dimensional Box*

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
  ## S3 method for class 'boxx'
scale(x, center=TRUE, scale=TRUE)
```

Arguments

x	Object of class "boxx" representing a multi-dimensional box.
...	Other arguments passed to print.default.
value	Name of the unit of length. See unitname.
center, scale	Arguments passed to scale.default to determine the rescaling.

Details

These are methods for the generic functions print, unitname, unitname<- and scale for the class "boxx" of multi-dimensional boxes.

The print method prints a description of the box, the unitname method extracts the name of the unit of length in which the box coordinates are expressed, while the assignment method for unitname assigns this unit name.

The scale method rescales each spatial coordinate of x.

Value

For print.boxx the value is NULL. For unitname.boxx an object of class "units". For unitname<-.boxx and scale.boxx the result is the updated "boxx" object x.

Author(s)

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

See Also

boxx, is.boxx, print, unitname, scale

Examples

```
B <- boxx(c(0,10),c(0,10),c(0,5),c(0,1), unitname=c("metre", "metres"))
B
is.boxx(B)
unitname(B)
# Northern European usage
unitname(B) <- "meter"
scale(B)
```

---

methods.distfun *Geometrical Operations for Distance Functions*

Description

Methods for objects of the class "distfun".

Usage

```
## S3 method for class 'distfun'
shift(X, ...)

## S3 method for class 'distfun'
rotate(X, ...)

## S3 method for class 'distfun'
scalardilate(X, ...)

## S3 method for class 'distfun'
affine(X, ...)

## S3 method for class 'distfun'
flipxy(X)

## S3 method for class 'distfun'
reflect(X)

## S3 method for class 'distfun'
rescale(X, s, unitname)
```

Arguments

X	Object of class "distfun" representing the distance function of a spatial object.
...	Arguments passed to the next method for the geometrical operation. See Details.
s, unitname	Arguments passed to the next method for rescale.

Details

These are methods for the generic functions shift, rotate, scalardilate, affine, flipxy and reflect which perform geometrical operations on spatial objects, and for the generic rescale which changes the unit of length.

The argument X should be an object of class "distfun" representing the distance function of a spatial object Y. Objects of class "distfun" are created by distfun.

The methods apply the specified geometrical transformation to the original object Y, producing a new object Z of the same type as Y. They then create a new distfun object representing the distance function of Z.

Value

Another object of class "distfun".

Author(s)

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

See Also

distfun, methods.funxy.

Examples

```
(f <- distfun(letterR))
plot(f)
flipxy(f)
shift(f, origin="midpoint")
plot(rotate(f, angle=pi/2))

(g <- distfun(lansing))
rescale(g)
```

`methods.funxy` *Methods for Spatial Functions*

Description

Methods for objects of the class `"funxy"`.

Usage

```
## S3 method for class 'funxy'
contour(x, ...)
## S3 method for class 'funxy'
persp(x, ...)
## S3 method for class 'funxy'
plot(x, ...)
```

Arguments

`x`	Object of class `"funxy"` representing a function of x, y coordinates.
`...`	Named arguments controlling the plot. See Details.

Details

These are methods for the generic functions `plot`, `contour` and `persp` for the class `"funxy"` of spatial functions.

Objects of class `"funxy"` are created, for example, by the commands `distfun` and `funxy`.

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.

In particular the argument `W` specifies the spatial domain over which the function will be plotted. See the Examples.

Value

Equivalent to the result of `contour.im`, `persp.im` or `plot.im` respectively.

Author(s)

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

See Also

`funxy`, `distfun`, `as.im`, `plot.im`, `persp.im`, `contour.im`, `spatstat.options`

Examples

```
f <- distfun(letterR)
contour(f)
## plot it on a larger region
B <- owin(c(1,5), c(-1, 4))
contour(f, W=B)
persp(f, W=B, theta=40, phi=40, border=NA, shade=0.7)
```

`methods.layered` *Methods for Layered Objects*

Description

Methods for geometrical transformations of layered objects (class `"layered"`).

Usage

```
  ## S3 method for class 'layered'
c(...)

  ## S3 method for class 'layered'
shift(X, vec=c(0,0), ...)

  ## S3 method for class 'layered'
rotate(X, ..., centre=NULL)

  ## S3 method for class 'layered'
affine(X, ...)

  ## S3 method for class 'layered'
reflect(X)

  ## S3 method for class 'layered'
flipxy(X)

  ## S3 method for class 'layered'
rescale(X, s, unitname)

  ## S3 method for class 'layered'
scalardilate(X, ...)
```

Arguments

`X`	Object of class `"layered"`.
`...`	Arguments passed to the relevant methods when applying the operation to each layer of `X`.
`s`	Rescaling factor passed to the relevant method for `rescale`. May be missing.

vec	Shift vector (numeric vector of length 2).
centre	Centre of rotation. Either a vector of length 2, or a character string (partially matched to "centroid", "midpoint" or "bottomleft"). The default is the coordinate origin c(0,0).
unitname	Optional. New name for the unit of length. A value acceptable to the function unitname<-

Details

These are methods for the generic functions c, shift, rotate, reflect, affine, rescale, scalardilate and flipxy for the class of layered objects.

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

The method for c can be used to concatenate two or more layered objects ... into a single layered object, retaining the plot arguments.

The other methods apply geometrical operations to each of the layers of the layered object X.

Value

Another object of class "layered".

Author(s)

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

See Also

layered

Examples

```
B <- owin(c(5500, 9000), c(2500, 7400))
L <- layered(Window(demopat), unmark(demopat)[B])
plot(L)
plot(rotate(L, pi/4))
```

---

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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)
```

methods.ppx *Methods for Multidimensional Space-Time Point Patterns*

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
## S3 method for class 'ppx'
scale(x, center=TRUE, scale=TRUE)
```

Arguments

x	Multidimensional point pattern (object of class "ppx").
...	Additional arguments passed to plot methods.
value	Name of the unit of length. See unitname.
center, scale	Arguments passed to scale.default to determine the rescaling.

Details

These are methods for the generic functions print, plot, unitname, unitname<- and scale 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.

The scale method rescales each spatial coordinate of x.

Value

For print.ppx and plot.ppx the value is NULL. For unitname.ppx the value is an object of class "units". For unitname<-.ppx and scale.ppx the value is another object of class "ppx".

Author(s)

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

See Also

ppx, unitname

`methods.unitname` *Methods for Units*

Description

Methods for class "unitname".

Usage

```
  ## S3 method for class 'unitname'
print(x, ...)
  ## S3 method for class 'unitname'
summary(object, ...)
  ## S3 method for class 'unitname'
rescale(X, s, unitname)
  ## S3 method for class 'unitname'
compatible(A,B, ..., coerce=TRUE)
  ## S3 method for class 'unitname'
harmonise(..., coerce=TRUE, single=FALSE)
  ## S3 method for class 'unitname'
harmonize(..., coerce=TRUE, single=FALSE)
```

Arguments

x, X, A, B, object	Objects of class "unitname" representing units of length.
...	Other arguments. For print.unitname these arguments are passed to print.default. For summary.unitname they are ignored. For compatible.unitname and harmonise.unitname these arguments are other objects of class "unitname".
s	Conversion factor: the new units are s times the old units.
unitname	Optional new name for the unit. If present, this overrides the rescaling operation and simply substitutes the new name for the old one.
coerce	Logical. If TRUE, a null unit of length is compatible with any non-null unit.
single	Logical value indicating whether to return a single unitname, or a list of unitnames.

Details

These are methods for the generic functions print, summary, rescale and compatible for the class "unitname".

An object of class "unitname" 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.

The harmonise method returns the common unit of length if there is one. For consistency with other methods for harmonise, the result is a list of unitname objects, with one entry for each argument in .... All of these entries are identical. This can be overridden by setting single=TRUE when the result will be a single unitname object.

Value

For print.unitname the value is NULL. For summary.unitname the value is an object of class summary.unitname (with its own print method). For rescale.unitname the value is another object of class "unitname". For compatible.unitname the result is logical. For harmonise.unitname the result is a list of identical unitnames if single=FALSE (the default), or a single unitname if single=TRUE.

Author(s)

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

See Also

box3, print, unitname

---

metric.object *Distance Metric*

Description

An object of class "metric" defines a measure of distance between points, and supports many operations that involve distances.

Details

A 'metric' d is a measure of distance between points that satisfies

1. $d(x, x) = 0$ for any point x,
2. $d(x, y) > 0$ for any two distinct points x and y
3. symmetry: $d(x, y) = d(y, x)$ for any two points x and y
4. triangle inequality: $d(x, y) \leq d(x, z) + d(z, y)$ for any three points x, y, z.

The Euclidean distance between points is an example of a metric.

An object of class "metric" is a structure that defines a metric and supports many computations that involve the metric. The internal structure of this object, and the mechanism for performing these computations, are under development.

Objects of class "metric" are produced by the function convexmetric and possibly by other functions.

There are methods for print and summary for the class "metric". The summary method lists the operations that are supported by the metric.

To perform distance calculations (for example, nearest-neighbour distances) using a desired metric instead of the Euclidean metric, first check whether the standard function for this purpose (for example nndist.ppp) has an argument named metric. If so, use the standard function and add the argument metric; if not, use the low-level function invoke.metric.

Author(s)

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

See Also

convexmetric, invoke.metric

Examples

```
m <- convexmetric(square(c(-1,1)))
summary(m)
y <- nndist(cells, metric=m)
```

---

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

marks.psp, summary.psp, lengths_psp angles.psp, endpoints.psp, extrapolate.psp.

Examples

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

MinkowskiSum *Minkowski Sum of Windows*

Description

Compute the Minkowski sum of two spatial windows.

Usage

```
MinkowskiSum(A, B)

A %(+)% B

dilationAny(A, B)
```

Arguments

A, B	Windows (objects of class "owin"), point patterns (objects of class "ppp") or line segment patterns (objects of class "psp") in any combination.

Details

The operator A %(+)% B and function MinkowskiSum(A,B) are synonymous: they both compute the Minkowski sum of the windows A and B. The function dilationAny computes the Minkowski dilation A %(+)% reflect(B).

The Minkowski sum of two spatial regions A and B is another region, formed by taking all possible pairs of points, one in A and one in B, and adding them as vectors. The Minkowski Sum $A \oplus B$ is the set of all points $a + b$ where a is in A and b is in B. A few common facts about the Minkowski sum are:

- The sum is symmetric: $A \oplus B = B \oplus A$.
- If B is a single point, then $A \oplus B$ is a shifted copy of A.
- If A is a square of side length a, and B is a square of side length b, with sides that are parallel to the coordinate axes, then $A \oplus B$ is a square of side length $a + b$.

- If A and B are discs of radius r and s respectively, then $A \oplus B$ is a disc of redius $r + s$.
- If B is a disc of radius r centred at the origin, then $A \oplus B$ is equivalent to the *morphological dilation* of A by distance r. See dilation.

The Minkowski dilation is the closely-related region $A \oplus (-B)$ where $(-B)$ is the reflection of B through the origin. The Minkowski dilation is the set of all vectors z such that, if B is shifted by z, the resulting set $B + z$ has nonempty intersection with A.

The algorithm currently computes the result as a polygonal window using the **polyclip** library. It will be quite slow if applied to binary mask windows.

The arguments A and B can also be point patterns or line segment patterns. These are interpreted as spatial regions, the Minkowski sum is computed, and the result is returned as an object of the most appropriate type. The Minkowski sum of two point patterns is another point pattern. The Minkowski sum of a point pattern and a line segment pattern is another line segment pattern.

Value

A window (object of class "owin") except that if A is a point pattern, then the result is an object of the same type as B (and vice versa).

Author(s)

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

See Also

dilation, erosionAny

Examples

```
B <- square(0.2)
RplusB <- letterR %(+)% B

opa <- par(mfrow=c(1,2))
FR <- grow.rectangle(Frame(letterR), 0.3)
plot(FR, main="")
plot(letterR, add=TRUE, lwd=2, hatch=TRUE, hatchargs=list(texture=5))
plot(shift(B, vec=c(3.675, 3)),
     add=TRUE, border="red", lwd=2)
plot(FR, main="")
plot(letterR, add=TRUE, lwd=2, hatch=TRUE, hatchargs=list(texture=5))
plot(RplusB, add=TRUE, border="blue", lwd=2,
       hatch=TRUE, hatchargs=list(col="blue"))
par(opa)

plot(cells %(+)% square(0.1))
```

multiplicity.ppp *Count Multiplicity of Duplicate Points*

Description

Counts the number of duplicates for each point in a spatial point pattern.

Usage

```
multiplicity(x)

## S3 method for class 'ppp'
multiplicity(x)

## S3 method for class 'ppx'
multiplicity(x)

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

## Default S3 method:
multiplicity(x)
```

Arguments

x	A spatial point pattern (object of class "ppp" or "ppx") or a vector, matrix or data frame.

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, the function multiplicity counts how many points are identical to it, and returns the vector of counts.

The argument x can also be a vector, a matrix or a data frame. When x is a vector, m <- multiplicity(x) is a vector of the same length as x, and m[i] is the number of elements of x that are identical to x[i]. When x is a matrix or data frame, m <- multiplicity(x) is a vector of length equal to the number of rows of x, and m[i] is the number of rows of x that are identical to the ith row.

Value

A vector of integers (multiplicities) of length equal to the number of points in x.

Author(s)

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

, Rolf Turner <rolfturner@posteo.net>

and Sebastian Meyer.

See Also

ppp.object, duplicated.ppp, unique.ppp

Examples

```
X <- ppp(c(1,1,0.5,1), c(2,2,1,2), window=square(3), check=FALSE)
m <- multiplicity(X)

# unique points in X, marked by their multiplicity
first <- !duplicated(X)
Y <- X[first] %mark% m[first]
```

NAobject *Make an NA Object*

Description

Create a object which represents a missing or unavailable object of a particular class in the **spatstat** package family.

Usage

```
NAobject(cls)
```

Arguments

cls	Character string specifying a class.

Details

This function creates an object which represents a missing or unavailable object belonging to one of the classes in the **spatstat** package family.

For example `NAobject("ppp")` creates an object which inherits the classes `"NAobject"` and `"ppp"`, and effectively represents a point pattern dataset that is missing or unavailable.

Typical uses of NA objects include simulation algorithms (where the algorithm may sometimes fail to produce a result) and designed experiments (where data may be missing for some of the experimental cases).

An NA object can be recognised by is.NAobject.

NA entries in a hyperframe can be recognised by is.na.hyperframe. NA entries in a spatial object list can be recognised by is.na.solist.

Value

An object belonging to the classes "NAobject" and cls.

Author(s)

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

See Also

is.NAobject, is.na.hyperframe, is.na.solist.

Examples

```
NAobject("tess")
```

---

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 x coordinates of any points
y	Numeric vector of y coordinates of any points
w	An image (object of class "im") or a binary mask window (an object of class "owin" of type "mask").
indices	Logical flag indicating whether to return the row and column indices, or the actual x, y coordinates.

Details

The argument w should be either a pixel image (object of class "im") or a window (an object of class "owin", see owin.object for details) of type "mask".

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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 <- runifrect(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")
```

---

nearestValue *Image of Nearest Defined Pixel Value*

Description

Given a pixel image defined on a subset of a rectangle, this function assigns a value to every pixel in the rectangle, by looking up the value of the nearest pixel that has a value.

Usage

```
nearestValue(X)
```

Arguments

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

Details

A pixel image in **spatstat** is always stored on a rectangular grid of pixels, but its value may be `NA` on some pixels, indicating that the image is not defined at those pixels.

This function assigns a value to every pixel in the rectangular grid. For each pixel `a` in the grid, if the value of `X` is not defined at `a`, the function finds the nearest other pixel `b` at which the value of `X` is defined, and takes the pixel value at `b` as the new pixel value at `a`.

Value

Another image of the same kind as `X`.

Author(s)

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

See Also

blur, Smooth.ppp

Examples

```
X <- as.im(function(x,y) { x + y }, letterR)
Y <- nearestValue(X)
plot(solist("X"=X,"nearestValue(X)"=Y), main="", panel.end=letterR)
```

---

nestsplit *Nested Split*

Description

Applies two splitting operations to a point pattern, producing a list of lists of patterns.

Usage

```
nestsplit(X, ...)
```

Arguments

X	Point pattern to be split. Object of class "ppp".
...	Data determining the splitting factors or splitting regions. See Details.

Details

This function splits the point pattern X into several sub-patterns using split.ppp, then splits each of the sub-patterns into sub-sub-patterns using split.ppp again. The result is a hyperframe containing the sub-sub-patterns and two factors indicating the grouping.

The arguments ... determine the two splitting factors or splitting regions. Each argument may be:

- a factor (of length equal to the number of points in X)
- the name of a column of marks of X (provided this column contains factor values)
- a tessellation (class "tess")
- a pixel image (class "im") with factor values
- a window (class "owin")
- identified by name (in the form name=value) as one of the formal arguments of quadrats or tess

The arguments will be processed to yield a list of two splitting factors/tessellations. The splits will be applied to X consecutively to produce the sub-sub-patterns.

Value

A hyperframe with three columns. The first column contains the sub-sub-patterns. The second and third columns are factors which identify the grouping according to the two splitting factors.

Author(s)

Original idea by Ute Hahn. Code by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

split.ppp, quantess

Examples

```
# factor and tessellation
Nft <- nestsplit(amacrine, marks(amacrine), quadrats(amacrine, 3, 1))
Ntf <- nestsplit(amacrine, quadrats(amacrine, 3, 1), marks(amacrine))
Ntf

# two factors
big <- with(marks(betacells), area > 300)
Nff <- nestsplit(betacells, "type", factor(big))

# two tessellations
Tx <- quantess(redwood, "x", 4)
Td <- dirichlet(runifrect(5, Window(redwood)))
Ntt <- nestsplit(redwood, Td, Tx)
Ntt2 <- nestsplit(redwood, Td, ny=3)
```

---

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, ...)

## S3 method for class 'ppp'
nncross(X, Y,
         iX=NULL, iY=NULL,
         what = c("dist", "which"),
         ...,
         k = 1, squared=FALSE,
```

```
            sortby=c("range", "var", "x", "y"),
            is.sorted.X = FALSE,
            is.sorted.Y = FALSE,
            metric=NULL)

  ## Default S3 method:
nncross(X, Y, ...)
```

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.
k	Integer, or integer vector. The algorithm will compute the distance to the kth nearest neighbour.
squared	Logical value specifying whether to return the squared nearest neighbour distances, instead of the distances themselves. This reduces computation time.
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.
metric	Optional. A distance metric (object of class "metric", see metric.object) which will be used to compute the distances.
...	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. If the argument k is specified, then the k-th nearest neighbours will be found.

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 ith entry is the distance from the ith point of X to the nearest element of Y). The second column gives the indices of the nearest neighbours (i.e.\ the ith 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.

The argument k may be an integer or an integer vector. If it is a single integer, then the k-th nearest neighbours are computed. If it is a vector, then the k[i]-th nearest neighbours are computed for each entry k[i]. For example, setting k=1:3 will compute the nearest, second-nearest and third-nearest neighbours. The result is a data frame.

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

A data frame, or a vector if the data frame would contain only one column.

By default (if what=c("dist", "which") and k=1) a data frame with two columns:

dist	Nearest neighbour distance
which	Nearest neighbour index in Y

If what="dist" and k=1, a vector of nearest neighbour distances.

If what="which" and k=1, a vector of nearest neighbour indices.

If k is specified, the result is a data frame with columns containing the k-th nearest neighbour distances and/or nearest neighbour indices.

Efficiency, sorting data, and pre-sorted data

Read this section if you care about the speed of computation.

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. Sorting is only an intermediate step; it does not affect the output, which is always given in the same order as the original data.

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.

Programmers should also consider whether their objective can be achieved using the squared distances (obtained by setting squared=TRUE) rather than the distances themselves. This reduces computation time, at least for the Euclidean distance (metric=NULL).

Author(s)

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

See Also

nndist for nearest neighbour distances in a single point pattern.

Examples

```
# two different point patterns
X <- runifrect(15)
Y <- runifrect(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)

# third-nearest neighbour
NXY <- nncross(X, Y, k=3)
NXY[1:3,]
# second and third nearest neighbours
NXY <- nncross(X, Y, k=2:3)
NXY[1:3,]

# two patterns with some points in common
Z <- runifrect(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 <- runifrect(15)
Y <- psp(runif(10), runif(10), runif(10), runif(10), square(1))
N <- nncross(X,Y)
```

---

nncross.pp3 *Nearest Neighbours Between Two Patterns in 3D*

Description

Given two point patterns X and Y in three dimensions, finds the nearest neighbour in Y of each point of X.

Usage

```
  ## S3 method for class 'pp3'
nncross(X, Y,
          iX=NULL, iY=NULL,
```

```
what = c("dist", "which"),
...,
k = 1,
sortby=c("range", "var", "x", "y", "z"),
is.sorted.X = FALSE,
is.sorted.Y = FALSE)
```

Arguments

X, Y	Point patterns in three dimensions (objects of class "pp3").
iX, iY	Optional identifiers, 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.
k	Integer, or integer vector. The algorithm will compute the distance to the kth nearest neighbour.
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 in three dimensions, this function finds, for each point of X, the nearest point of Y. The distance between these points is also computed. If the argument k is specified, then the k-th nearest neighbours will be found.

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 ith entry is the distance from the ith point of X to the nearest element of Y). The second column gives the indices of the nearest neighbours (i.e.\ the ith 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.

The argument k may be an integer or an integer vector. If it is a single integer, then the k-th nearest neighbours are computed. If it is a vector, then the k[i]-th nearest neighbours are computed for each entry k[i]. For example, setting k=1:3 will compute the nearest, second-nearest and third-nearest neighbours. The result is a data frame.

Note that this function is not symmetric in X and Y. To find the nearest neighbour in X of each point in Y, 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

A data frame, or a vector if the data frame would contain only one column.

By default (if `what=c("dist", "which")` and `k=1`) a data frame with two columns:

`dist`	Nearest neighbour distance
`which`	Nearest neighbour index in `Y`

If `what="dist"` and `k=1`, a vector of nearest neighbour distances.

If `what="which"` and `k=1`, a vector of nearest neighbour indices.

If `k` is specified, the result is a data frame with columns containing the `k`-th nearest neighbour distances and/or nearest neighbour indices.

Sorting data and pre-sorted data

Read this section if you care about the speed of computation.

For efficiency, the algorithm sorts both the point patterns `X` and `Y` into increasing order of the x coordinate, or both into increasing order of the y coordinate, or both into increasing order of the z coordinate. Sorting is only an intermediate step; it does not affect the output, which is always given in the same order as the original data.

By default (if `sortby="range"`), the sorting will occur on the coordinate that has the largest 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"` or `sortby = "z"` will specify that sorting should occur on the x, y or z 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"`, `"y"` or `"z"` 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"`, `"y"` or `"z"` 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"`, `"y"` or `"z"` to indicate which coordinate is sorted.

Author(s)

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

, Rolf Turner <rolfturner@posteo.net> , and Jens Oehlschlaegel

See Also

`nndist` for nearest neighbour distances in a single point pattern.

Examples

```
# two different point patterns
X <- pp3(runif(10), runif(10), runif(10), box3(c(0,1)))
Y <- pp3(runif(20), runif(20), runif(20), box3(c(0,1)))
N <- nncross(X,Y)$which
```

```
N <- nncross(X,Y, what="which") #faster
# note that length(N) = 10

# k-nearest neighbours
N3 <- nncross(X, Y, k=1:3)

# two patterns with some points in common
Z <- pp3(runif(20), runif(20), runif(20), box3(c(0,1)))
X <- Z[1:15]
Y <- Z[10:20]
iX <- 1:15
iY <- 10:20
N <- nncross(X,Y, iX, iY, what="which")
```

---

nncross.ppx *Nearest Neighbours Between Two Patterns in Any Dimensions*

Description

Given two point patterns X and Y in many dimensional space, finds the nearest neighbour in Y of each point of X.

Usage

```
  ## S3 method for class 'ppx'
nncross(X, Y,
          iX=NULL, iY=NULL,
          what = c("dist", "which"),
          ...,
          k = 1)
```

Arguments

X, Y	Point patterns in any number of spatial dimensions (objects of class "ppx").
iX, iY	Optional identifiers, 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.
k	Integer, or integer vector. The algorithm will compute the distance to the kth nearest neighbour.
...	Ignored.

Details

Given two point patterns X and Y in m-dimensional space, this function finds, for each point of X, the nearest point of Y. The distance between these points is also computed. If the argument k is specified, then the k-th nearest neighbours will be found.

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 ith entry is the distance from the ith point of X to the nearest element of Y). The second column gives the indices of the nearest neighbours (i.e.\ the ith 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.

The argument k may be an integer or an integer vector. If it is a single integer, then the k-th nearest neighbours are computed. If it is a vector, then the k[i]-th nearest neighbours are computed for each entry k[i]. For example, setting k=1:3 will compute the nearest, second-nearest and third-nearest neighbours. The result is a data frame.

Note that this function is not symmetric in X and Y. To find the nearest neighbour in X of each point in Y, 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

A data frame, or a vector if the data frame would contain only one column.

By default (if what=c("dist", "which") and k=1) a data frame with two columns:

dist	Nearest neighbour distance
which	Nearest neighbour index in Y

If what="dist" and k=1, a vector of nearest neighbour distances.

If what="which" and k=1, a vector of nearest neighbour indices.

If k is specified, the result is a data frame with columns containing the k-th nearest neighbour distances and/or nearest neighbour indices.

Author(s)

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

See Also

nndist for nearest neighbour distances in a single point pattern.

Examples

```
XYZ <- ppx(matrix(runif(80), 20, 4),
          boxx(c(0,1), c(0,1), c(0,1), c(0,1)))

## two different point patterns
X <- XYZ[1:5]
Y <- XYZ[10:20]
nncross(X,Y)
N23 <- nncross(X,Y, k=2:3)

## two patterns with some points in common
X <- XYZ[1:15]
Y <- XYZ[10:20]
iX <- 1:15
iY <- 10:20
N <- nncross(X,Y, iX, iY, what="which")
N4 <- nncross(X,Y, iX, iY, k=4)
```

---

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,
         by=NULL, squared=FALSE, proper=FALSE, method="C", metric=NULL)
  ## Default S3 method:
nndist(X, Y=NULL, ..., k=1,
         by=NULL, squared=FALSE, method="C")
```

Arguments

X, Y	Arguments specifying the locations of a set of points. For nndist.ppp, the argument X should be a point pattern (object of class "ppp"). For nndist.default, 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. Alternatively X can be a three-dimensional point pattern (class "pp3"), a higher-dimensional point pattern (class "ppx"), a point pattern on a linear network (class "lpp"), or a spatial pattern of line segments (class "psp").
...	Ignored by nndist.ppp and nndist.default.
k	Integer, or integer vector. The algorithm will compute the distance to the kth nearest neighbour.

by	Optional. A factor, which separates X into groups. The algorithm will compute the distance to the nearest point in each group. See Details.
squared	Logical value specifying whether to return the squared distances rather than the distances. This reduces computation time.
proper	Logical value specifying whether to measure distances only between distinct points (proper=TRUE) or to include distances between data points at the same spatial location (proper=FALSE, the default). See Details.
method	String specifying which method of calculation to use. Values are "C" and "interpreted".
metric	Optional. A metric (object of class "metric") that will be used to define and compute the distances.

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 kth 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 are also methods for line segment patterns, nndist.psp, three-dimensional point patterns, nndist.pp3, higher-dimensional point patterns, nndist.ppx and point patterns on a linear network, nndist.lpp; these are described in their own help files. Type methods(nndist) to see all available methods.

The method for planar point patterns nndist.ppp 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 kth nearest neighbour distances are computed for each value of k specified in the vector.

If the argument by is given, it should be a factor, of length equal to the number of points in X. This factor effectively partitions X into subsets, each subset associated with one of the levels of X. The algorithm will then compute, for each point of X, the distance to the nearest neighbour *in each subset*.

If proper=FALSE (the default), data points which lie at the same spatial location are treated as different individuals, and the nearest-neighbour distance for any such point is defined to be zero. If proper=TRUE, only distances between distinct points are included in the calculation, so the nearest-neighbour distance is always nonzero (or NA if there are not enough data points).

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 ith data point.

If k is a single integer, then the return value is a numeric vector v such that v[i] is the kth nearest neighbour distance for the ith 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 ith data point.

If the argument by is given, then it should be a factor which separates X into groups (or any type of data acceptable to split.ppp that determines the grouping). The result is a data frame containing the distances described above, from each point of X, to the nearest point in each subset of X defined by the grouping factor by.

Nearest neighbours of each type

If X is a multitype point pattern and by=marks(X), then the algorithm will compute, for each point of X, the distance to the nearest neighbour of each type. See the Examples.

To find the minimum distance from *any* point of type i to the nearest point of type j, for all combinations of i and j, use minnndist, or the R function aggregate as suggested in the Examples.

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).

Distance values

The values returned by nndist(X) are distances, expressed as multiples of the unit of length of the spatial coordinates in X. The unit of length is given by unitname(X).

Note that, if the unit of length in X is a composite expression such as '2 microns', then the values of nndist(X) are expressed as multiples of 2 microns, rather than being expressed in microns.

Author(s)

Pavel Grabarnik <pavel.grabar@issp.serpukhov.su> and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

See Also

nndist.psp, nndist.pp3, nndist.ppx, pairdist, Gest, nnwhich, nncross, minnndist, maxnndist.

Examples

```
# 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), markscale=1)

# distance to nearest neighbour of each type
nnda <- nndist(ants, by=marks(ants))
head(nnda)
# For nest number 1, the nearest Cataglyphis nest is 87.32125 units away

# minimum distance between each pair of types
minnndist(ants, by=marks(ants))

# Use of 'aggregate':
# _minimum_ distance between each pair of types
aggregate(nnda, by=list(from=marks(ants)), min)
# _mean_ nearest neighbour distances
aggregate(nnda, by=list(from=marks(ants)), mean)
# The mean distance from a Messor nest to
# the nearest Cataglyphis nest is 59.02549 units
```

`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, by=NULL)
```

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.
by	Optional. A factor, which separates X into groups. The algorithm will compute the distance to the nearest point in each group.

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.

If the argument by is given, it should be a factor, of length equal to the number of points in X. This factor effectively partitions X into subsets, each subset associated with one of the levels of X. The algorithm will then compute, for each point of X, the distance to the nearest neighbour *in each subset*.

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 G3est.

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 ith data point.

If k is a single integer, then the return value is a numeric vector v such that v[i] is the kth nearest neighbour distance for the ith 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 ith data point.

Distance values

The values returned by nndist(X) are distances, expressed as multiples of the unit of length of the spatial coordinates in X. The unit of length is given by unitname(X).

Note that, if the unit of length in X is a composite expression such as '2 microns', then the values of nndist(X) are expressed as multiples of 2 microns, rather than being expressed in microns.

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@curtin.edu.au> based on code for two dimensions by Pavel Grabarnik <pavel.grabar@issp.serpukhov.su>.

See Also

nndist, pairdist, G3est, nnwhich

Examples

```
X <- pp3(runif(40), runif(40), runif(40), box3(c(0,1)))

# nearest neighbours
d <- nndist(X)

# second nearest neighbours
d2 <- nndist(X, k=2)

# first, second and third nearest
d1to3 <- nndist(X, k=1:3)

# distance to nearest point in each group
marks(X) <- factor(rep(letters[1:4], 10))
dby <- nndist(X, by=marks(X))
```

---

nndist.ppx *Nearest Neighbour Distances in Any Dimensions*

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, by=NULL)
```

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 kth nearest neighbour.
by	Optional. A factor, which separates X into groups. The algorithm will compute the distance to the nearest point in each group.

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 kth 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 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.

If the argument by is given, it should be a factor, of length equal to the number of points in X. This factor effectively partitions X into subsets, each subset associated with one of the levels of X. The algorithm will then compute, for each point of X, the distance to the nearest neighbour *in each subset*.

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 ith data point.

If k is a single integer, then the return value is a numeric vector v such that v[i] is the kth nearest neighbour distance for the ith 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 ith data point.

Distance values

The values returned by nndist(X) are distances, expressed as multiples of the unit of length of the spatial coordinates in X. The unit of length is given by unitname(X).

Note that, if the unit of length in X is a composite expression such as '2 microns', then the values of nndist(X) are expressed as multiples of 2 microns, rather than being expressed in microns.

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@curtin.edu.au>

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)

# nearest neighbour distances to each group
marks(X) <- factor(c("a","a", "b", "b", "b"))
nndist(X, by=marks(X))
nndist(X, by=marks(X), k=1:2)
```

---

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="C")
```

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 "C" 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="C" (the default) then compiled C code is used. The C 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 ith segment.

If k is a single integer, then the return value is a numeric vector v such that v[i] is the kth nearest neighbour distance for the ith 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 ith segment.

Distance values

The values returned by nndist(X) are distances, expressed as multiples of the unit of length of the spatial coordinates in X. The unit of length is given by unitname(X).

Note that, if the unit of length in X is a composite expression such as '2 microns', then the values of nndist(X) are expressed as multiples of 2 microns, rather than being expressed in microns.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

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)
```

---

nnfun *Nearest Neighbour Index Map as a Function*

Description

Compute the nearest neighbour index map of an object, and return it as a function.

Usage

```
  nnfun(X, ...)

  ## S3 method for class 'ppp'
nnfun(X, ..., k=1, value=c("index", "mark"))

  ## S3 method for class 'psp'
nnfun(X, ..., value=c("index", "mark"))
```

Arguments

X	Any suitable dataset representing a two-dimensional collection of objects, such as a point pattern (object of class "ppp") or a line segment pattern (object of class "psp").
k	A single integer. The kth nearest neighbour will be found.
...	Extra arguments are ignored.
value	String (partially matched) specifying whether to return the index of the neighbour (value="index", the default) or the mark value of the neighbour (value="mark").

Details

For a collection X of two dimensional objects (such as a point pattern or a line segment pattern), the "nearest neighbour index function" of X is the mathematical function f such that, for any two-dimensional spatial location (x, y), the function value f(x,y) is the index i identifying the closest member of X. That is, if $i = f(x, y)$ then $X[i]$ is the closest member of the collection X to the location (x, y).

The command f <- nnfun(X) returns a *function* in the R language, with arguments x,y, that represents the nearest neighbour index 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 indices of the nearest neighbours to these locations.

If the argument k is specified then the k-th nearest neighbour will be found.

The result of f <- nnfun(X) also belongs to the class "funxy" and to the special class "nnfun". It can be printed and plotted immediately as shown in the Examples.

A nnfun 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 "nnfun" which has a method for print. It also belongs to the class "funxy" which has methods for plot, contour and persp.

Author(s)

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

See Also

distfun, plot.funxy

Examples

```
f <- nnfun(cells)
f
plot(f)
f(0.2, 0.3)

g <- nnfun(cells, k=2)
g(0.2, 0.3)

plot(nnfun(amacrine, value="m"))

L <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
h <- nnfun(L)
h(0.2, 0.3)
```

---

nnmap *K-th Nearest Point Map*

Description

Given a point pattern, this function constructs pixel images giving the distance from each pixel to its k-th nearest neighbour in the point pattern, and the index of the k-th nearest neighbour.

Usage

```
nnmap(X, k = 1, what = c("dist", "which"),
..., W = as.owin(X),
is.sorted.X = FALSE, sortby = c("range", "var", "x", "y"), squared=FALSE)
```

Arguments

X	Point pattern (object of class "ppp").
k	Integer, or integer vector. The algorithm will find the kth nearest neighbour.
what	Character string specifying what information should be returned. Either the nearest neighbour distance ("dist"), the index of the nearest neighbour ("which"), or both.
...	Arguments passed to as.mask to determine the pixel resolution of the result.
W	Window (object of class "owin") specifying the spatial domain in which the distances will be computed. Defaults to the window of X.
is.sorted.X	Logical value attesting whether the point pattern X has been sorted. See Details.
sortby	Determines which coordinate to use to sort the point pattern. See Details.
squared	Logical value specifying whether to return the squared distances instead of the distances themselves.

Details

Given a point pattern X, this function constructs two pixel images:

- a distance map giving, for each pixel, the distance to the nearest point of X;
- a nearest neighbour map giving, for each pixel, the identifier of the nearest point of X.

If the argument k is specified, then the k-th nearest neighbours will be found.

If what="dist" then only the distance map is returned. If what="which" then only the nearest neighbour map is returned.

The argument k may be an integer or an integer vector. If it is a single integer, then the k-th nearest neighbours are computed. If it is a vector, then the k[i]-th nearest neighbours are computed for each entry k[i]. For example, setting k=1:3 will compute the nearest, second-nearest and third-nearest neighbours.

Value

A pixel image, or a list of pixel images.

By default (if what=c("dist", "which")), the result is a list with two components dist and which containing the distance map and the nearest neighbour map.

If what="dist" then the result is a real-valued pixel image containing the distance map.

If what="which" then the result is an integer-valued pixel image containing the nearest neighbour map.

If k is a vector of several integers, then the result is similar except that each pixel image is replaced by a list of pixel images, one for each entry of k.

If squared=TRUE then squared distances are returned with the name "dist2" instead of "dist".

Sorting data and pre-sorted data

Read this section if you care about the speed of computation.

For efficiency, the algorithm sorts the point pattern `X` into increasing order of the x coordinate or increasing order of the the y coordinate. Sorting is only an intermediate step; it does not affect the output, which is always given in the same order as the original data.

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 `X`). If `sortby = "var"`), sorting will occur on the coordinate that has the greater variance (in the pattern `X`). 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 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.

Warning About Ties

Ties are possible: there may be two data points which lie exactly the same distance away from a particular pixel. This affects the results from `nnmap(what="which")`. The handling of ties is not well-defined: it is not consistent between different computers and different installations of R. If there are ties, then different calls to `nnmap(what="which")` may give inconsistent results. For example, you may get a different answer from `nnmap(what="which",k=1)` and `nnmap(what="which", k=1:2)[[1]]`.

Author(s)

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

See Also

`distmap`

Examples

```
plot(nnmap(cells, 2, what="which"))
```

---

nnmark *Mark of Nearest Neighbour*

Description

Given a marked point pattern dataset `X` this function computes, for each desired location `y`, the mark attached to the nearest neighbour of `y` in `X`. The desired locations `y` can be either a pixel grid or the point pattern `X` itself.

Usage

```
nnmark(X, ..., k = 1, at=c("pixels", "points"),
                  ties=c("first", "mean", "min", "max"),
                  proper=FALSE)
```

Arguments

X	A marked point pattern (object of class "ppp").
...	Arguments passed to as.mask to determine the pixel resolution.
k	Single integer. The kth nearest data point will be used.
at	String specifying whether to compute the values at a grid of pixel locations (at="pixels") or only at the points of X (at="points").
ties	Character string (partially matched) indicating how to handle the case of ties, where there are two or more data points at the same location. See Details.
proper	Logical value specifying how to define nearest neighbours if there are two or more data points at the same location. Applies only when at="points". See Details.

Details

Given a marked point pattern dataset X this function computes, for each desired location y, the mark attached to the point of X that is nearest to y. The desired locations y can be either a pixel grid or the point pattern X itself.

The argument X must be a marked point pattern (object of class "ppp", see ppp.object). The marks are allowed to be a vector or a data frame.

- If at="points", then for each point in X, the algorithm finds the nearest *other* point in X, and extracts the mark attached to it. The result is a vector or data frame containing the marks of the neighbours of each point.
- If at="pixels" (the default), then for each pixel in a rectangular grid, the algorithm finds the nearest point in X, and extracts the mark attached to it. The result is an image or a list of images containing the marks of the neighbours of each pixel. The pixel resolution is controlled by the arguments ... passed to as.mask.

If the argument k is given, then the k-th nearest neighbour will be used.

The arguments ties and proper specify how to handle the case where two or more data points are at the same spatial location.

- ties determines how to pool the mark values. If ties="first" (the default), the mark value for this location is taken to be the mark of the data point that is listed first in sequence in the dataset X. If ties="mean", ties="max" or ties="min", the mark value for this location is taken to be the mean, maximum or minimum (respectively) of the mark values of all the data points at this location (after converting the mark values to numerical values).
- proper determines how to define nearest neighbours, when at="points". If proper=TRUE, the nearest neighbour of a data point must be another data point lying a nonzero distance away from it. If proper=FALSE (the default), then two data points occupying the exact same spatial location can be nearest neighbours.

Value

If X *has a single column of marks:*

- If at="pixels" (the default), the result is a pixel image (object of class "im"). The value at each pixel is the mark attached to the nearest point of X.
- If at="points", the result is a vector or factor of length equal to the number of points in X. Entries are the mark values of the nearest neighbours of each point of X.

If X *has a data frame of marks:*

- If at="pixels" (the default), the result is a named list of pixel images (object of class "im"). There is one image for each column of marks. This list also belongs to the class "solist", for which there is a plot method.
- If at="points", the result is a data frame with one row for each point of X, Entries are the mark values of the nearest neighbours of each point of X.

Author(s)

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

See Also

Smooth.ppp, marktable, nnwhich

Examples

```
plot(nnmark(ants))
v <- nnmark(ants, at="points")
v[1:10]
plot(nnmark(finpines))
vf <- nnmark(finpines, at="points")
vf[1:5,]
```

---

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, by=NULL, proper=FALSE, method="C", metric=NULL)
  ## Default S3 method:
nnwhich(X, Y=NULL, ..., k=1, by=NULL, method="C")
```

Arguments

X, Y	Arguments specifying the locations of a set of points. For nnwhich.ppp, the argument X should be a point pattern (object of class "ppp"). For nnwhich.default, 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 nnwhich.ppp and nnwhich.default.
k	Integer, or integer vector. The algorithm will compute the distance to the kth nearest neighbour.
by	Optional. A factor, which separates X into groups. The algorithm will find the nearest neighbour in each group. See Details.
proper	Logical value specifying whether to measure distances only between distinct points (proper=TRUE) or to include distances between data points at the same spatial location (proper=FALSE, the default). See Details.
method	String specifying which method of calculation to use. Values are "C" and "interpreted".
metric	Optional. A metric (object of class "metric") that will be used to define and compute the distances.

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 kth 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 kth nearest neighbour distances are computed for each value of k specified in the vector.

If the argument by is given, it should be a factor, of length equal to the number of points in X. This factor effectively partitions X into subsets, each subset associated with one of the levels of X. The algorithm will then find, for each point of X, the nearest neighbour *in each subset*.

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.

If proper=FALSE (the default), data points which lie at the same spatial location are treated as different individuals, and the nearest neighbour of any such point is one of the other individuals at the same location. If proper=TRUE, points lying at the same location are treated as identical, so that nearest neighbours are always separated by a nonzero distance.

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 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 neighbout of the ith point is the jth point where j = v[i]).

If k is a single integer, then the return value is a numeric vector giving the indices of the kth 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 ith data point.

If the argument by is given, then it should be a factor which separates X into groups (or any type of data acceptable to split.ppp that determines the grouping). The result is a data frame containing the indices described above, from each point of X, to the nearest point in each subset of X defined by the factor by.

Nearest neighbours of each type

If X is a multitype point pattern and by=marks(X), then the algorithm will find, for each point of X, the nearest neighbour of each type. See the Examples.

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@curtin.edu.au>

See Also

nndist, nncross

Examples

```
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)

# nearest neighbours of each type
head(nnwhich(ants, by=marks(ants)))
```

---

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 neighbout of the ith point is the jth point where j = v[i]).

If k is a single integer, then the return value is a numeric vector giving the indices of the kth 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 ith 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@curtin.edu.au> based on two-dimensional code by Pavel Grabarnik <pavel.grabar@issp.serpukhov.su>.

See Also

nnwhich, nndist, nncross

Examples

```
if(require(spatstat.random)) {
   X <- runifpoint3(30)
} else {
   X <- osteo$pts[[1]]
}
   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").
...	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 kth 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 kth 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 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 neighbout of the ith point is the jth point where j = v[i]).

If k is a single integer, then the return value is a numeric vector giving the indices of the kth 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 ith 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@curtin.edu.au>

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)
```

`nobjects` *Count Number of Geometrical Objects in a Spatial Dataset*

Description

A generic function to count the number of geometrical objects in a spatial dataset.

Usage

```
  nobjects(x)

  ## S3 method for class 'ppp'
nobjects(x)

  ## S3 method for class 'ppx'
nobjects(x)

  ## S3 method for class 'psp'
nobjects(x)

  ## S3 method for class 'tess'
nobjects(x)
```

Arguments

`x`	A dataset.

Details

The generic function `nobjects` counts the number of geometrical objects in the spatial dataset `x`.

The methods for point patterns (classes `"ppp"` and `"ppx"`, embracing `"pp3"` and `"lpp"`) count the number of points in the pattern.

The method for line segment patterns (class `"psp"`) counts the number of line segments in the pattern.

The method for tessellations (class `"tess"`) counts the number of tiles of the tessellation.

Value

A single integer.

Author(s)

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

See Also

npoints

Examples

```
nobjects(redwood)
nobjects(edges(letterR))
nobjects(dirichlet(cells))
```

---

npoints *Number of Points in a Point Pattern*

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

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

See Also

ppp.object, print.pp3, print.ppx.

Examples

```
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, or an object containing a linear network.

Details

This function is generic, with methods for classes psp, linnet and lpp.

Value

Integer.

Author(s)

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

See Also

npoints(), psp.object()

Examples

```
nsegments(copper$Lines)
nsegments(copper$SouthLines)
```

nvertices *Count Number of Vertices*

Description

Count the number of vertices in an object for which vertices are well-defined.

Usage

```
  nvertices(x, ...)

  ## S3 method for class 'owin'
nvertices(x, ...)

  ## Default S3 method:
nvertices(x, ...)
```

Arguments

x	A window (object of class "owin"), or some other object which has vertices.
...	Currently ignored.

Details

This function counts the number of vertices of x as they would be returned by vertices(x). It is more efficient than executing npoints(vertices(x)).

Value

A single integer.

Author(s)

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

See Also

vertices

Examples

```
nvertices(square(2))
nvertices(letterR)
```

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 as.mask 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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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

```
v <- opening(letterR, 0.3)
plot(letterR, type="n", main="opening")
plot(v, add=TRUE, col="grey")
plot(letterR, add=TRUE)
```

---

overlap.owin *Compute Area of Overlap*

Description

Computes the area of the overlap (intersection) of two windows.

Usage

```
overlap.owin(A, B)
```

Arguments

A, B	Windows (objects of class "owin").

Details

This function computes the area of the overlap between the two windows A and B.

If one of the windows is a binary mask, then both windows are converted to masks on the same grid, and the area is computed by counting pixels. Otherwise, the area is computed analytically (using the discrete Stokes theorem).

Value

A single numeric value.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

intersect.owin, area.owin, setcov.

Examples

```
A <- square(1)
B <- shift(A, c(0.3, 0.2))
overlap.owin(A, B)
```

---

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), yrange=c(0,1), ..., poly=NULL, mask=NULL,
unitname=NULL, xy=NULL)
```

Arguments

xrange	x coordinate limits of enclosing box
yrange	y coordinate limits of enclosing box
...	Ignored.
poly	Optional. Polygonal boundary of window. Incompatible with mask.
mask	Optional. Logical matrix giving binary image of window. Incompatible with poly.
unitname	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.
xy	Optional. List with components x and y specifying the pixel coordinates for mask.

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 `yrange` 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 dimensions `xrange=c(0,1)` and `yrange=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.
  - *Specified by logical matrix:* Normally 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 (`NA` entries will be treated as `FALSE`). 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.
  - *Specified by list of pixel coordinates:* Alternatively the argument `mask` can be a data frame with 2 or 3 columns. If it has 2 columns, it is expected to contain the spatial coordinates of all the pixels which are inside the window. If it has 3 columns, it should contain the spatial coordinates (x, y) of every pixel in the grid, and the logical value associated with each pixel. The pixels may be listed in any order.

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.

Validity of polygon data

Polygon data may contain geometrical inconsistencies such as self-intersections and overlaps. These inconsistencies must be removed to prevent problems in other **spatstat** functions. By default, polygon data will be repaired automatically using polygon-clipping code. The repair process may change the number of vertices in a polygon and the number of polygon components. To disable the repair process, set `spatstat.options(fixpolygons=FALSE)`.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

`owin.object`, `as.owin`, `complement.owin`, `ppp.object`, `ppp`

`square`, `hexagon`, `regularpolygon`, `disc`, `ellipse`.

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

# vertices of a polygon (listed anticlockwise)
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))
# vertices could alternatively be read from a file, or use locator()
w <- owin(poly=bdry)

## Not run:
# how to read in a binary mask from a file
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 se
`"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`, `boundingbox`, `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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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, boundingbox, 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))
if(FALSE) {
  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
}
```

---

owin2mask *Convert Window to Binary Mask under Constraints*

Description

Converts a window (object of class "owin") to a binary pixel mask, with options to require that the inside, outside, and/or boundary of the window should be completely covered.

Usage

```
owin2mask(w,
          ...,
          rule.pix = c("sample", "notsample",
                       "cover", "inside",
                       "uncover", "outside",
                       "boundary",
```

```
                    "majority", "minority"),
        W=w)
```

Arguments

w	A window (object of class "owin").
rule.pix	Character string (partially matched) specifying how w should be converted to a binary pixel mask.
...	Optional arguments in name=value format, passed to as.mask to determine the pixel resolution. Unrecognised arguments will be silently ignored.
W	Alternative name for argument w, used by some legacy code.

Details

This function is similar to, but more flexible than, as.mask. It converts the interior, exterior, or boundary of the window w to a binary pixel mask.

- If rule.pix="sample" (the default), the mask consists of all pixels whose **centres** lie inside the window w. This is the same as using as.mask.
- If rule.pix="notsample", the mask consists of all pixels whose *centres lie outside* the window w. This is the same as using as.mask followed by complement.owin.
- If rule.pix="cover", the mask consists of all pixels which overlap the window w, wholly or partially.
- If rule.pix="inside", the mask consists of all pixels which lie entirely inside the window w.
- If rule.pix="uncover", the mask consists of all pixels which overlap the outside of the window w, wholly or partially.
- If rule.pix="outside", the mask consists of all pixels which lie entirely outside the window w.
- If rule.pix="boundary", the mask consists of all pixels which overlap the boundary of the window w.
- If rule.pix="majority", the mask consists of all pixels in which at least half of the pixel area is covered by the window w.
- If rule.pix="minority", the mask consists of all pixels in which less than half of the pixel area is covered by the window w.

These operations are complementary pairs as follows:

"notsample"	is the complement of	"sample"
"uncover"	is the complement of	"inside"
"outside"	is the complement of	"cover"
"minority"	is the complement of	"majority"

They also satisfy the following set relations:

"inside"	is a subset of	"cover"
"outside"	is a subset of	"uncover"
"boundary"	is a subset of	"cover"
"boundary"	is a subset of	"uncover"

The results of "inside", "boundary" and "outside" are disjoint and their union is the entire frame.

Theoretically "sample" should be a subset of "cover", "notsample" should be a subset of "uncover", "inside" should be a subset of "majority" and "outside" should be a subset of "minority", but these could be false due to numerical error in computational geometry.

Value

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

Author(s)

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

See Also

as.mask

Examples

```
W <- Window(chorley)
opa <- par(mfrow=c(2,5))
plot(as.mask(W, dimyx=10), col="grey", main="sample")
plot(W, add=TRUE, lwd=3, border="red")
plot(owin2mask(W, rule.pix="ma", dimyx=10), col="grey", main="majority")
plot(W, add=TRUE, lwd=3, border="red")
plot(owin2mask(W, rule.pix="i", dimyx=10), col="grey", main="inside")
plot(W, add=TRUE, lwd=3, border="red")
plot(owin2mask(W, rule.pix="c", dimyx=10), col="grey", main="cover")
plot(W, add=TRUE, lwd=3, border="red")
plot(owin2mask(W, rule.pix="b", dimyx=10), col="grey", main="boundary")
plot(W, add=TRUE, lwd=3, border="red")
plot(as.mask(complement.owin(W), dimyx=10), col="grey",
     main="notsample")
plot(W, add=TRUE, lwd=3, border="red")
plot(owin2mask(W, rule.pix="mi", dimyx=10), col="grey", main="minority")
plot(W, add=TRUE, lwd=3, border="red")
plot(owin2mask(W, rule.pix="o", dimyx=10), col="grey", main="outside")
plot(W, add=TRUE, lwd=3, border="red")
plot(owin2mask(W, rule.pix="u", dimyx=10), col="grey", main="uncover")
plot(W, add=TRUE, lwd=3, border="red")
plot(owin2mask(W, rule.pix="b", dimyx=10), col="grey", main="boundary")
plot(W, add=TRUE, lwd=3, border="red")
par(opa)
```

padimage *Pad the Border of a Pixel Image*

Description

Fills the border of a pixel image with a given value or values, or extends a pixel image to fill a larger window.

Usage

```
padimage(X, value=NA, n=1, W=NULL)
```

Arguments

X	Pixel image (object of class "im").
value	Single value to be placed around the border of X.
n	Width of border, in pixels. See Details.
W	Window for the resulting image. Incompatible with n.

Details

The image X will be expanded by a margin of n pixels, or extended to fill the window W, with new pixel values set to value.

The argument value should be a single value (a vector of length 1), normally a value of the same type as the pixel values of X. It may be NA. Alternatively if X is a factor-valued image, value can be one of the levels of X.

If n is given, it may be a single number, specifying the width of the border in pixels. Alternatively it may be a vector of length 2 or 4. It will be replicated to length 4, and these numbers will be interpreted as the border widths for the (left, right, top, bottom) margins respectively.

Alternatively if W is given, the image will be extended to the window W.

Value

Another object of class "im", of the same type as X.

Author(s)

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

See Also

im

Examples

```
Z <- setcov(owin())
plot(padimage(Z, 1, 10))

D <- distmap(letterR, invert=TRUE, eps=0.05)[letterR, drop=FALSE]
V <- dilation(letterR, 0.15)
plot(padimage(D, 0.4, W=V))
```

`pairdist` *Pairwise distances*

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`.

Distance values

The values returned by `pairdist(X)` are distances, expressed as multiples of the unit of length of the spatial coordinates in `X`. The unit of length is given by `unitname(X)`.

Note that, if the unit of length in `X` is a composite expression such as '2 microns', then the values of `pairdist(X)` are expressed as multiples of 2 microns, rather than being expressed in microns.

Author(s)

Pavel Grabarnik <pavel.grabar@issp.serpukhov.su> and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

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 in two dimensional space

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 in two dimensional space, 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.

Note: If only pairwise distances within some threshold value are needed the low-level function closepairs may be much faster to use.

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.

For typical input the result is numerically equivalent to (but computationally faster than) as.matrix(dist(x)) where x = cbind(X, Y), but that command is useful for calculating all pairwise distances between points in k-dimensional space when x has k 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@curtin.edu.au>.

See Also

crossdist, nndist, Kest, closepairs

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.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 squared=TRUE, 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.

Distance values

The values returned by pairdist(X) are distances, expressed as multiples of the unit of length of the spatial coordinates in X. The unit of length is given by unitname(X).

Note that, if the unit of length in X is a composite expression such as '2 microns', then the values of pairdist(X) are expressed as multiples of 2 microns, rather than being expressed in microns.

Author(s)

Adrian Baddeley <Adrian.Baddeley@curtin.edu.au> based on two-dimensional code by Pavel Grabarnik <pavel.grabar@issp.serpukhov.su>.

See Also

pairdist, crossdist, nndist, K3est

Examples

```
if(require(spatstat.random)) {
   X <- runifpoint3(20)
} else {
   X <- osteo$pts[[1]]
}
   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, metric=NULL)
```

Arguments

X	A point pattern (object 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". Usually not specified.
squared	Logical. If squared=TRUE, the squared distances are returned instead (this computation is faster).
metric	Optional. A metric (object of class "metric") that will be used to define and compute the distances.

Details

This is a method for the generic function pairdist.

Given a point pattern X (an object of class "ppp"), 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.

Distance values

The values returned by `pairdist(X)` are distances, expressed as multiples of the unit of length of the spatial coordinates in `X`. The unit of length is given by `unitname(X)`.

Note that, if the unit of length in `X` is a composite expression such as '2 microns', then the values of `pairdist(X)` are expressed as multiples of 2 microns, rather than being expressed in microns.

Author(s)

Pavel Grabarnik <pavel.grabar@issp.serpukhov.su> and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

See Also

`pairdist`, `pairdist.default`, `pairdist.psp`, `crossdist`, `nndist`, `Kest`

Examples

```
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.

Distance values

The values returned by pairdist(X, temporal=FALSE) are distances, expressed as multiples of the unit of length of the spatial coordinates in X. The unit of length is given by unitname(X).

Note that, if the unit of length in X is a composite expression such as '2 microns', then the values of pairdist(X, temporal=FALSE) are expressed as multiples of 2 microns, rather than being expressed in microns.

Author(s)

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

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="C", type="Hausdorff")
```

Arguments

X	A line segment pattern (object of class "psp").
...	Ignored.
method	String specifying which method of calculation to use. Values are "C" and "interpreted". Usually not specified.
type	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="C" (the default) then compiled C code is used, which is somewhat faster.

Value

A square matrix whose [i,j] entry is the distance between the line segments numbered i and j.

Distance values

The values returned by pairdist(X) are distances, expressed as multiples of the unit of length of the spatial coordinates in X. The unit of length is given by unitname(X).

Note that, if the unit of length in X is a composite expression such as '2 microns', then the values of pairdist(X) are expressed as multiples of 2 microns, rather than being expressed in microns.

Author(s)

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

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")
```

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

area.owin diameter.owin, owin.object, as.owin

Examples

```
perimeter(square(3))
perimeter(letterR)
if(interactive()) print(perimeter(as.mask(letterR)))
```

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. The result will be a grid of 2 * nx + 1 by 2 * ny + 1 copies of the original object. (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 `nx`, `ny` are given and other arguments are missing, the original object will be copied `nx` times to the right and `nx` times to the left, then `ny` times upward and `ny` times downward, making `(2 * nx + 1) * (2 * ny + 1)` copies altogether, arranged in a grid, centred on the original object.

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

See Also

shift

Examples

```
plot(periodify(cells))
a <- lapply(periodify(Window(cells), combine=FALSE),
      plot, add=TRUE,lty=2)
```

`persp.im` *Perspective Plot of Pixel Image*

Description

Displays a perspective plot of a pixel image. The pixel values are treated as heights of a surface; the surface is displayed in perspective view.

Usage

```
  ## S3 method for class 'im'
persp(x, ...,
                    colmap=NULL, colin=x, valuesAreColours=NULL,
                    apron=FALSE, visible=FALSE)
```

Arguments

x	The pixel image to be plotted as a surface. 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.
colin	Optional. Colour input. Another pixel image (of the same dimensions as x) containing the values that will be mapped to colours.
valuesAreColours	Optional. Logical value indicating whether the pixel values in colin are to be interpreted directly as colour values (valuesAreColours=TRUE) or interpreted as numerical values that should be mapped to colours (valuesAreColours=FALSE). The default is TRUE if colin is a character-valued or factor-valued image, and FALSE otherwise.
apron	Logical. If TRUE, a grey apron is placed around the sides of the perspective plot.
visible	Logical value indicating whether to compute which pixels of x are visible in the perspective view. 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 function in the R language of the form function(n, ...), this function will be called with an appropriate value of n to generate a character vector of n colours. Examples of such functions are heat.colors, terrain.colors, topo.colors and cm.colors.
- If colmap is a function in the R language of the form function(range, ...) then it will be called with range equal to the range of altitudes, to determine the colour values or colour map. Examples of such functions are beachcolours and beachcolourmap.
- If colmap is a list with entries breaks and col, then colmap$breaks determines the breakpoints of the altitude intervals, and colmap$col provides the corresponding colours.

Alternatively, if the argument `colin` (*colour input*) is present, then `colin` determines the colour of each piece of the surface. The result is a perspective view of a surface with heights determined by `x` and colours determined by `colin`:

- If `colin` is a numeric-valued image, or if `valuesAreColours=FALSE` is specified, the pixel values of `colin` will be mapped to colour values to determine the colour of each piece of the surface. The colour map `colmap` controls this mapping.
- If `colin` is a character-valued or factor-valued image, or if `valuesAreColours=TRUE` is specified, the pixel values of `colin` are interpreted directly as colour values.

If `apron=TRUE`, a vertical surface is drawn around the boundary of the perspective plot, so that the terrain appears to have been cut out of a solid material. If colour data were supplied, then the apron is coloured light grey.

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)`.

The vertical scale is controlled by the argument `expand`: setting `expand=1` will interpret the pixel values as being in the same units as the spatial coordinates x and y and represent them at the same scale.

If `visible=TRUE`, the algorithm also computes whether each pixel in `x` is visible in the perspective view. In order to be visible, a pixel must not be obscured by another pixel which lies in front of it (as seen from the viewing direction), and the three-dimensional vector normal to the surface must be pointing toward the viewer. The return value of `persp.im` then has an attribute `"visible"` which is a pixel image, compatible with `x`, with pixel value equal to `TRUE` if the corresponding pixel in `x` is visible, and `FALSE` if it is not visible.

Value

(invisibly) the 3D transformation matrix returned by `persp.default`, together with an attribute `"expand"` which gives the relative scale of the z coordinate.

If argument `visible=TRUE` was given, the return value also has an attribute `"visible"` which is a pixel image, compatible with `x`, with logical values which are *TRUE* when the corresponding pixel is visible in the perspective view, and `FALSE` when it is obscured.

Author(s)

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

See Also

`perspPoints`, `perspLines` for drawing additional points or lines *on the surface*.

`trans3d` for mapping arbitrary (x, y, z) coordinate locations to the plotting coordinates.

`im.object`, `plot.im`, `contour.im`

Examples

```
# an image
Z <- setcov(owin(), dimyx=32)
persp(Z, colmap=terrain.colors(128))
if(interactive()) {
  co <- colourmap(range=c(0,1), col=rainbow(128))
  persp(Z, colmap=co, axes=FALSE, shade=0.3)
}

## Terrain elevation
persp(bei.extra$elev, colmap=terrain.colors(128),
      apron=TRUE, theta=-30, phi=20,
      zlab="Elevation", main="", ticktype="detailed",
      expand=6)

if(interactive()) {
  ## apron for non-rectangular domain
  sto <- subset(stonetools, TYPE == "LITHIC", select=NULL)
  D <- density(sto, 0.65, dimyx=512)
 persp(D, phi=30, apron=TRUE, colmap=terrain.colors, expand=0.5, box=FALSE, theta=-10)
}
```

---

persp.ppp *Perspective Plot of Marked Point Pattern*

Description

For a spatial point pattern with numeric marks, generate a perspective plot in which each data point is shown as a vertical spike, with height proportional to the mark value.

Usage

```
  ## S3 method for class 'ppp'
persp(x, ..., main, type=c("l", "b"),
            grid = TRUE, ngrid = 10,
            col.grid = "grey", col.base = "white",
            win.args=list(),
            spike.args = list(), neg.args = list(),
            point.args=list(),
            which.marks = 1,
            zlab = NULL, zlim = NULL, zadjust = 1,
            show.window=TRUE,
            legend=TRUE, legendpos="bottomleft",
            leg.args=list(lwd=4), leg.col=c("black", "orange"))
```

Arguments

x	A spatial point pattern (object of class "ppp") with numeric marks, or a data frame of marks.
...	Additional graphical arguments passed to persp.default to determine the perspective view (for example the rotation angle theta and the elevation angle phi) or passed to segments to control the drawing of lines (for example lwd for line width) or passed to points.default to control the drawing of points (for example pch for symbol type).
main	Optional main title for the plot.
type	Single character specifying how each observation will be plotted: type="l" for lines, type="b" for both lines and points.
grid	Logical value specifying whether to draw a grid of reference lines on the horizontal plane.
ngrid	Number of grid lines to draw in each direction, if grid=TRUE. An integer, or a pair of integers specifying the number of grid lines along the horizontal and vertical axes respectively.
col.grid	Colour of grid lines, if grid=TRUE.
col.base	Colour with which to fill the horizontal plane. A single colour value, or a pixel image.
win.args	List of arguments passed to plot.owin to control the drawing of the window of x. Applicable only when the window is not a rectangle.
spike.args	List of arguments passed to segments to control the drawing of the spikes.
neg.args	List of arguments passed to segments applicable only to those spikes which have negative height (corresponding to a mark value which is negative).
point.args	List of arguments passed to points.default to control the drawing of the points, when type="b".
which.marks	Integer, or character name, identifying the column of marks which should be used, when marks(x) is a data frame.
zlab	Optional. Label for the vertical axis. Character string or expression.
zlim	Optional. Range of values on the vertical axis. A numeric vector of length 2.
zadjust	Scale adjustment factor controlling the height of spikes.
show.window	Logical value specifying whether to draw the window of x on the horizontal plane.
legend	Logical value specifying whether to draw a reference scale bar for the vertical axis.
legendpos	Position of the reference scale bar. Either a character string matching one of the options "bottomleft", "bottomright", "topleft", "topright", "bottom", "left", "top" or "right", or a numeric vector of length 2 specifing the coordinate position of the base of the reference scale bar.
leg.args	Additional arguments passed to segments to control the drawing of the reference scale bar.
leg.col	A vector (usually of length 2) of colour values for successive intervals in the reference scale. The default is a reference scale consisting of black and orange stripes.

Details

The function persp is generic. This is the method for spatial point patterns (objects of class "ppp"). The argument x must be a point pattern with numeric marks, or with a data frame of marks.

A perspective view will be plotted. The eye position is determined by the arguments theta and phi passed to persp.default.

First the horizontal plane is drawn in perspective view, using a faint grid of lines to help suggest the perspective. Next the observation window of x is placed on the horizontal plane and its edges are drawn in perspective view. Finally for each data point in x, a vertical spike is erected at the spatial location of the data point, with height equal to the mark value of the point.

The horizontal plane is effectively transparent, by default. If the argument col.base is a single colour value, the horizontal plane will be painted in this colour. If col.base is a pixel image, it will be rendered as a colour image shown in perspective view on the horizontal plane:

- If valuesAreColours=TRUE is specified, the pixel values of col.base will be interpreted directly as colour values.
- Otherwise, the argument colmap controls the mapping from pixel values of col.base to physical colours.

After the horizontal plane has been coloured, the observation window Window(x) will be drawn on the horizontal plane (unless show.window=FALSE). By default, only the edges of the window will be drawn, as black line segments. The style of plotting the window can be modified using win.args. The colour of the line segments is controlled by win.args$border. If win.args$col is given, the window will be filled in the specified colour, overwriting any previous colours. See the Examples.

If any mark values are negative, the corresponding spikes will penetrate below the horizontal plane. They can be drawn in a different colour by specifying neg.args as shown in the examples. These would be obscured if col.base is an opaque colour other than white, or if col.base is a pixel image. A transparent colour for col.base can be used if it is supported by the graphics device.

Like all spatial plots in the **spatstat** family, persp.ppp does not independently rescale the x and y coordinates. A long narrow window will be represented as a long narrow window in the perspective view. To override this and allow the coordinates to be independently rescaled, use the argument scale=TRUE which will be passed to persp.default.

Value

(Invisibly) the perspective transformation matrix.

Author(s)

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

Examples

```
persp(longleaf, theta=-30, phi=35, spike.args=list(lwd=3), zadjust=1.5,
      zlab="diameter")

# negative mark values
X <- longleaf
marks(X) <- marks(X) - 20
```

```
persp(X, theta=80, phi=35, neg.args=list(col="red"),
      spike.args=list(lwd=3), zadjust=1.2)

# irregular window
Australia <- Window(austates)
Y <- runifrect(70, Frame(Australia))[Australia]
marks(Y) <- runif(npoints(Y))
persp(Y, theta=30, phi=20, col.base="lightblue",
      win.args=list(col="pink", border=NA),
      spike.args=list(lwd=2), zadjust=1.5, zlab="")

persp(Y, type="b",
      theta=30, phi=20, col.base="lightblue",
      win.args=list(col="pink", border=NA),
      spike.args=list(lty=3), point.args=list(col="blue"), zadjust=1.5)
```

---

perspPoints *Draw Points or Lines on a Surface Viewed in Perspective*

Description

After a surface has been plotted in a perspective view using persp.im, these functions can be used to draw points or lines on the surface.

Usage

```
perspPoints(x, y=NULL, ..., Z, M, occluded=TRUE)

perspLines(x, y = NULL, ..., Z, M, occluded=TRUE)

perspSegments(x0, y0 = NULL, x1 = NULL, y1 = NULL, ..., Z, M, occluded=TRUE)

perspContour(Z, M, ...,
             nlevels=10, levels=pretty(range(Z), nlevels),
             occluded=TRUE)
```

Arguments

x, y	Spatial coordinates, acceptable to xy.coords, for the points or lines on the horizontal plane.
Z	Pixel image (object of class "im") specifying the surface heights.
M	Projection matrix returned from persp.im when Z was plotted.
...	Graphical arguments passed to points, lines or segments to control the drawing.
x0, y0, x1, y1	Spatial coordinates of the line segments, on the horizontal plane. Alternatively x0 can be a line segment pattern (object of class "psp") and y0,x1,y1 can be NULL.

nlevels	Number of contour levels
levels	Vector of heights of contours.
occluded	Logical value specifying whether parts of the surface can be obscured by other parts of the surface. See Details.

Details

After a surface has been plotted in a perspective view, these functions can be used to draw points or lines on the surface.

The user should already have called persp.im to display the perspective view of the surface Z and to obtain the perspective matrix M by typing M <- persp(Z, ...). The points and lines will be drawn in their correct three-dimensional position according to the same perspective.

If occluded=TRUE (the default), then the surface is treated as if it were opaque. The code will draw only those points and lines which are visible from the viewer's standpoint, and not obscured by other parts of the surface lying closer to the viewer. The user should already have called persp.im in the form M <- persp(Z, visible=TRUE, ...) to compute the visibility information.

If occluded=FALSE, then the surface is treated as if it were transparent. All the specified points and lines will be drawn on the surface.

Value

Same as the return value from points or segments.

Author(s)

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

See Also

persp.im

Examples

```
M <- persp(bei.extra$elev, colmap=terrain.colors(128),
           apron=TRUE, theta=-30, phi=20,
           zlab="Elevation", main="",
           expand=6, visible=TRUE, shade=0.3)

perspContour(bei.extra$elev, M=M, col="pink", nlevels=12)
perspPoints(bei, Z=bei.extra$elev, M=M, pch=16, cex=0.3, col="chartreuse")
```

pHcolourmap *Colour Map for pH Values*

Description

Create a colour map for values of pH.

Usage

```
pHcolourmap(range = c(0, 14), ..., n=256, step = FALSE)
pHcolour(pH)
```

Arguments

n	Number of different colour values to be used, when step=FALSE.
range	Range of pH values that will be accepted as inputs to the colour map. A numeric vector of length 2 giving the minimum and maximum values of pH.
step	Logical value. If step=FALSE (the default) the colours change continuously with increasing values of the input. If step=TRUE, the colour is constant on each unit interval of pH values.
...	Ignored.
pH	Numerical value or numeric vector of values of pH.

Details

In chemistry the hydrogen potential pH measures how acidic or basic a solution is.

The function pHcolour calculates the colour associated with a given value of pH, according to a standard mapping in which neutral $pH = 7$ is green, acidic values $pH < 7$ are yellow or red, and basic values $pH > 7$ are blue. The function pHcolour takes a numerical value or vector of values of pH and returns a character vector containing the corresponding colours.

The function pHcolourmap produces a colour map for numerical values of pH, using the same consistent mapping of pH values to colours. The argument range specifies the range of pH values that will be mapped by the resulting colour map. It should be a numeric vector of length 2 giving the minimum and maximum values of pH that the colour map will handle. (Colour maps created with different values of range use essentially the same mapping of colours, but when plotted as colour ribbons, display only the specified range.)

If step=FALSE (the default) the colours change continuously with increasing values of the input. There will be n different colour values in the colour map. Usually n should be a large number.

If step=TRUE, the colour is constant on each unit interval of pH values. That is, any value of pH in the interval $[k, k + 1]$, where k is an integer, will be mapped to the same colour.

Value

The return value of pHcolour is a character string or a vector of character strings representing colours.

The return value of pHcolourmap is a colour map (object of class "colourmap").

Author(s)

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

See Also

colourmap

Examples

```
pHcolour(7)

plot(pHcolourmap())
plot(pHcolourmap(step=TRUE))
plot(pHcolourmap(c(3, 8)))
```

---

pixelcentres *Extract Pixel Centres as Point Pattern*

Description

Given a pixel image or binary mask window, extract the centres of all pixels and return them as a point pattern.

Usage

```
pixelcentres(X, W = NULL, ...)
```

Arguments

X	Pixel image (object of class "im") or window (object of class "owin").
W	Optional window to contain the resulting point pattern.
...	Optional arguments defining the pixel resolution.

Details

If the argument X is a pixel image, the result is a point pattern, consisting of the centre of every pixel whose pixel value is not NA.

If X is a window which is a binary mask, the result is a point pattern consisting of the centre of every pixel inside the window (i.e. every pixel for which the mask value is TRUE).

Otherwise, X is first converted to a window, then converted to a mask using as.mask, then handled as above.

Value

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

Author(s)

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

, Rolf Turner <rolfturner@posteo.net>

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

See Also

raster.xy

Examples

```
pixelcentres(letterR, dimyx=5)
```

---

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

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, ..., DivideByPixelArea=FALSE)
```

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 and owin2mask to determine the pixel raster.
DivideByPixelArea	Logical value, indicating whether the resulting pixel values should be divided by the pixel area.

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").

Boundary pixels

When x is a rectangle or a polygonal window, the default behaviour is to represent the window as the set of pixels whose *centres* fall inside the polygon. A pixel which contains part of the interior of the polygon, but whose centre falls outside the polygon, is excluded by default, for the sake of efficiency.

To ensure that all pixels which intersect the interior of the polygon are included in the pixel image, use the argument rule.pix="cover" which will be passed to owin2mask.

Author(s)

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

See Also

pixellate.ppp, pixellate, as.im

Examples

```
plot(pixellate(letterR, dimyx=15))
plot(letterR, add=TRUE)
plot(pixellate(letterR, dimyx=15, rule.pix="cover"))
plot(letterR, add=TRUE)
W <- grow.rectangle(as.rectangle(letterR), 0.2)
plot(pixellate(letterR, W, dimyx=15))
```

---

pixellate.ppp *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, fractional=FALSE, preserve=FALSE,
                          DivideByPixelArea=FALSE, savemap=FALSE)

## S3 method for class 'ppp'
as.im(X, ...)
```

Arguments

x, X	Point pattern (object of class "ppp").
...	Arguments passed to owin2mask or 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 value indicating whether to set pixel values to zero outside the window.
fractional, preserve	Logical values determining the type of discretisation. See Details.
DivideByPixelArea	Logical value, indicating whether the resulting pixel values should be divided by the pixel area.
savemap	Logical value, indicating whether to save information about the discretised coordinates of the points of x.

Details

The functions pixellate.ppp and as.im.ppp 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.ppp is equivalent to calling pixellate.ppp with its default arguments. Note that pixellate.ppp is more general than as.im.ppp (it has additional arguments for greater flexibility).

The functions as.im.ppp and pixellate.ppp 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.ppp 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.

The discretisation procedure is controlled by the arguments fractional and preserve.

- The argument fractional specifies how data points are mapped to pixels. If fractional=FALSE (the default), each data point is allocated to the nearest pixel centre. If fractional=TRUE, each data point is allocated with fractional weight to four pixel centres (the corners of a rectangle containing the data point).
- The argument preserve specifies what to do with pixels lying near the boundary of the window, if the window is not a rectangle. If preserve=FALSE (the default), any contributions that are attributed to pixel centres lying outside the window are reset to zero. If preserve=TRUE, any such contributions are shifted to the nearest pixel lying inside the window, so that the total mass is preserved.

If savemap=TRUE then the result has an attribute "map" which is a 2-column matrix containing the row and column indices of the discretised positions of the points of x in the pixel grid.

Value

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

Boundary pixels

When Window(x) is a rectangle or a polygonal window, the default behaviour is to represent the window as the set of pixels whose *centres* fall inside the polygon. A pixel which contains part of the interior of the polygon, but whose centre falls outside the polygon, is excluded by default, for the sake of efficiency.

To ensure that all pixels which intersect the interior of the polygon are included in the pixel image, use the argument rule.pix="cover" which will be passed to owin2mask.

Author(s)

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

See Also

pixellate, im, as.im, density.ppp, Smooth.ppp.

Examples

```
plot(pixellate(humberside))
plot(pixellate(humberside, fractional=TRUE))
```

pixellate.psp *Convert Line Segment Pattern to Pixel Image*

Description

Converts a line segment pattern to a pixel image by measuring the length or number of lines intersecting each pixel.

Usage

```
## S3 method for class 'psp'
pixellate(x, W=NULL, ..., weights = NULL,
                          what=c("length", "number", "indicator"),
                          DivideByPixelArea=FALSE)
```

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 or owin2mask to determine the pixel resolution.
weights	Optional vector of weights associated with each line segment, used when what="length".
what	String (partially matched) indicating whether to compute the total length of intersection of segments with each pixel (what="length", the default), the total number of segments intersecting each pixel (what="number") or an indicator variable that equals 1 if any segments intersect the pixel, and 0 otherwise (code="indicator").
DivideByPixelArea	Logical value, indicating whether the resulting pixel values should be divided by the pixel area.

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. Alternatively it can count the number of line segments intersecting each pixel.

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 and what="length", 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.

Boundary pixels

When `Window(x)` is a rectangle or a polygonal window, the default behaviour is to represent the window as the set of pixels whose *centres* fall inside the polygon. A pixel which contains part of the interior of the polygon, but whose centre falls outside the polygon, is excluded by default, for the sake of efficiency.

To ensure that all pixels which intersect the interior of the polygon are included in the pixel image, use the argument `rule.pix="cover"` which will be passed to `owin2mask`.

Author(s)

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

See Also

`pixellate`, `as.mask`, `psp2mask`.

Use `psp2mask` 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(pixellate(X))
plot(X, add=TRUE)
sum(lengths_psp(X))
sum(pixellate(X))
plot(pixellate(X, what="n"))
```

---

pixelquad *Quadrature Scheme Based on Pixel Grid*

Description

Makes a quadrature scheme with a dummy point at every pixel of a pixel image.

Usage

```
pixelquad(X, W = as.owin(X), ...)
```

Arguments

`X`	Point pattern (object of class `"ppp"`) containing the data points for the quadrature scheme.
`W`	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`.
`...`	Optional arguments to `as.mask` controlling the pixel raster dimensions.

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

quadscheme, quad.object, ppm

Examples

```
W <- owin(c(0,1),c(0,1))
X <- runifrect(42, W)
W <- as.mask(W,dimyx=128)
pixelquad(X,W)
```

plot.anylist *Plot a List of Things*

Description

Plots a list of things

Usage

```
  ## S3 method for class 'anylist'
plot(x, ..., main, arrange=TRUE,
   nrows=NULL, ncols=NULL, main.panel=NULL,
   mar.panel=c(2,1,1,2), hsep=0, vsep=0,
   panel.begin=NULL, panel.end=NULL, panel.args=NULL,
   panel.begin.args=NULL, panel.end.args=NULL, panel.vpad=0.2,
   plotcommand="plot",
   do.plot=TRUE,
   adorn.left=NULL, adorn.right=NULL, adorn.top=NULL, adorn.bottom=NULL,
   adorn.size=0.2, adorn.args=list(),
   equal.scales=FALSE, halign=FALSE, valign=FALSE)
```

Arguments

x	An object of the class "anylist". Essentially a list of objects.
...	Arguments passed to plot when generating each plot panel.
main	Overall heading for the plot.
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.
main.panel	Optional. A character string, or a vector of character strings, giving the headings for each of the objects.
mar.panel	Size of the margins outside each plot panel. A numeric vector of length 4 giving the bottom, left, top, and right margins in that order. (Alternatively the vector may have length 1 or 2 and will be replicated to length 4). See the section on *Spacing between plots*.
hsep, vsep	Additional horizontal and vertical separation between plot panels, expressed in the same units as mar.panel.
panel.begin, panel.end	Optional. Functions that will be executed before and after each panel is plotted. See Details.
panel.args	Optional. Function that determines different plot arguments for different panels. See Details.
panel.begin.args	Optional. List of additional arguments for panel.begin when it is a function.

`panel.end.args` Optional. List of additional arguments for `panel.end` when it is a function.

`panel.vpad` Amount of extra vertical space that should be allowed for the title of each panel, if a title will be displayed. Expressed as a fraction of the height of the panel. Applies only when `equal.scales=FALSE` (the default) and requires that the height of each panel can be determined.

`plotcommand` Optional. Character string containing the name of the command that should be executed to plot each panel.

`do.plot` Logical value specifying whether to actually perform the plot.

`adorn.left`, `adorn.right`, `adorn.top`, `adorn.bottom`
Optional. Functions that will be executed to generate additional plots at the margins (left, right, top and/or bottom, respectively) of the array of plots. Alternatively they may be objects of class `"colourmap"` or `"symbolmap"`.

`adorn.size` Relative width (as a fraction of the other panels' widths) of the margin plots.

`adorn.args` Optional list of arguments passed to the functions `adorn.left`, `adorn.right`, `adorn.top` and `adorn.bottom` if they are functions, or arguments passed to `plot.colourmap` or `plot.symbolmap` as appropriate.

`equal.scales` Logical value indicating whether the components should be plotted at (approximately) the same physical scale.

`halign`, `valign` Logical values indicating whether panels in a column should be aligned to the same x coordinate system (`halign=TRUE`) and whether panels in a row should be aligned to the same y coordinate system (`valign=TRUE`). These are applicable only if `equal.scales=TRUE`.

Details

This is the `plot` method for the class `"anylist"`.

An object of class `"anylist"` represents a list of objects intended to be treated in the same way. This is the method for `plot`.

In the **spatstat** package, various functions produce an object of class `"anylist"`, essentially a list of objects of the same kind. These objects can be plotted in a nice arrangement using `plot.anylist`. See the Examples.

The argument `panel.args` determines extra graphics parameters for each panel. It should be a function that will be called as `panel.args(i)` where `i` is the panel number. Its return value should be a list of graphics parameters that can be passed to the relevant `plot` method. These parameters override any parameters specified in the `...` arguments.

The arguments `panel.begin` and `panel.end` determine graphics that will be plotted before and after each panel is plotted. They may be objects of some class that can be plotted with the generic `plot` command. Alternatively they may be functions that will be called as `panel.begin(i, y, main=main.panel[i])` and `panel.end(i, y, add=TRUE)` where `i` is the panel number and `y = x[[i]]`.

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.

If `equal.scales=FALSE` (the default), then the plot panels will have equal height on the plot device (unless there is only one column of panels, in which case they will have equal width on the plot device). This means that the objects are plotted at different physical scales, by default.

If equal.scales=TRUE, then the dimensions of the plot panels on the plot device will be proportional to the spatial dimensions of the corresponding components of x. This means that the objects will be plotted at *approximately* equal physical scales. If these objects have very different spatial sizes, the plot command could fail (when it tries to plot the smaller objects at a tiny scale), with an error message that the figure margins are too large.

The objects will be plotted at *exactly* equal physical scales, and *exactly* aligned on the device, under the following conditions:

- every component of x is a spatial object whose position can be shifted by shift;
- panel.begin and panel.end are either NULL or they are spatial objects whose position can be shifted by shift;
- adorn.left, adorn.right, adorn.top and adorn.bottom are all NULL.

Another special case is when every component of x is an object of class "fv" representing a function. If equal.scales=TRUE then all these functions will be plotted with the same axis scales (i.e. with the same xlim and the same ylim).

Value

Null.

Spacing between plots

The spacing between individual plots is controlled by the parameters mar.panel, hsep and vsep.

If equal.scales=FALSE, the plot panels are logically separate plots. The margins for each panel are determined by the argument mar.panel which becomes the graphics parameter mar described in the help file for par. One unit of mar corresponds to one line of text in the margin. If hsep or vsep are present, mar.panel is augmented by c(vsep, hsep, vsep, hsep)/2.

If equal.scales=TRUE, all the plot panels are drawn in the same coordinate system which represents a physical scale. The unit of measurement for mar.panel[1,3] is one-sixth of the greatest height of any object plotted in the same row of panels, and the unit for mar.panel[2,4] is one-sixth of the greatest width of any object plotted in the same column of panels. If hsep or vsep are present, they are interpreted in the same units as mar.panel[2] and mar.panel[1] respectively.

Error messages

If the error message 'Figure margins too large' occurs, this generally means that one of the objects had a much smaller physical scale than the others. Ensure that equal.scales=FALSE and increase the values of mar.panel.

Author(s)

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

See Also

contour.listof, image.listof, density.splitppp

Examples

```
if(require(spatstat.explore)) {
 trichotomy <- list(regular=cells,
                    random=japanesepines,
                    clustered=redwood)
 K <- lapply(trichotomy, Kest)
 K <- as.anylist(K)
 plot(K, main="")
}

# list of 3D point patterns
 ape1 <- osteo[osteo$shortid==4, "pts", drop=TRUE]
 class(ape1)
 plot(ape1, main.panel="", mar.panel=0.1, hsep=0.7, vsep=1,
      cex=1.5, pch=21, bg='white')
```

---

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, side = if(vertical) "right" else "bottom",
       labelmap=NULL, gap=0.25, add=FALSE, do.plot=TRUE,
       increasing=NULL, nticks=5,
       at=NULL, box=NULL)
```

Arguments

x	Colour map to be plotted. An object of class "colourmap".
...	Graphical arguments passed to image.default or axis.
main	Main title for plot. A character string.
xlim	Optional range of x values for the location of the colour ribbon.
ylim	Optional range of y values for the location of the colour ribbon.
vertical	Logical flag determining whether the colour ribbon is plotted as a horizontal strip (FALSE) or a vertical strip (TRUE).
axis	Logical value determining whether an axis should be plotted showing the numerical values that are mapped to the colours.

side	One of the character strings "bottom", "left", "top" or "right", or one of the integers from 1 to 4, specifying the position of the axis tick marks and labels, if axis=TRUE.
labelmap	Function. If this is present, then the labels on the plot, which indicate the input values corresponding to particular colours, will be transformed by labelmap before being displayed on the plot. Typically used to simplify or shorten the labels on the plot.
gap	Distance between separate blocks of colour, as a fraction of the width of one block, if the colourmap is discrete.
add	Logical value indicating whether to add the colourmap to the existing plot (add=TRUE), or to start a new plot (add=FALSE, the default).
do.plot	Logical value indicating whether to actually perform the plot.
increasing	Logical value indicating whether to display the colour map in increasing order. See Details.
nticks	Optional. Integer specifying the approximate number of tick marks (representing different values of the numerical input) that should be drawn next to the colour map. Applies only when the colour map inputs are numeric values.
at	Optional. Numeric values at which tick marks should be drawn. Applies only when the colour map inputs are numeric values.
box	Optional. Logical value specifying whether to draw a black box around the colour ribbon. Default is box=FALSE when plotting separate blocks of colour (i.e. when the colourmap is discrete and gap > 0) and box=TRUE otherwise.

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 or as a colour legend (a sequence of blocks of colour). This plot can be useful on its own to inspect the colour map.

If the domain of the colourmap is an interval of real numbers, the colourmap is displayed as a continuous ribbon of colour. If the domain of the colourmap is a finite set of inputs, the colours are displayed as separate blocks of colour. The separation between blocks is equal to gap times the width of one block.

To annotate an existing plot with an explanatory colour ribbon or colour legend, specify add=TRUE and use the arguments xlim and/or ylim to control the physical position of the ribbon on the plot.

Labels explaining the colour map are drawn by axis and can be modified by specifying arguments that will be passed to this function.

The argument increasing indicates whether the colourmap should be displayed so that the input values are increasing with the spatial coordinate: that is, increasing from left to right (if vertical=FALSE) or increasing from bottom to top (if vertical=TRUE). If increasing=FALSE, this ordering will be reversed. The default is increasing=TRUE in all cases except when vertical=TRUE and the domain of the colourmap is a finite set of discrete inputs.

Value

(Invisibly) a rectangular window (object of class "owin") giving the region of coordinate space where the colours were drawn (or would have been drawn).

Author(s)

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

See Also

colourmap

Examples

```
co <- colourmap(rainbow(100), breaks=seq(-1,1,length=101))
plot(co)
plot(co, col.ticks="pink")
ca <- colourmap(rainbow(8), inputs=letters[1:8])
plot(ca, vertical=TRUE)
## logarithmic colour map
cl <- colourmap(rainbow(25), range=c(0.1, 1000), compress=log10)
plot(cl)
```

---

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=mar * marsize),
                          marsize=1, mar=c(1,1,3,1))
```

Arguments

x	Data to be plotted. A hyperframe (object of class "hyperframe", see hyperframe).
e	How to plot each row. Optional. An R language call or expression (typically enclosed in quote() 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 around the panels. Incompatible with parargs.
mar	Optional numeric vector of length 1, 2 or 4 controlling the relative sizes of margins between the panels. Incompatible with parargs.

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. (It is typically created using quote or expression.) 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@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

hyperframe, with.hyperframe

Examples

```
H <- hyperframe(id=1:6)
H$X <- with(H, runifrect(100))
H$D <- with(H, distmap(X))
# points only
plot(H[,"X"])
plot(H, quote(plot(X, main=id)))
```

```
# points superimposed on images
plot(H, quote({plot(D, main=id); plot(X, add=TRUE)}))
```

---

plot.im *Plot a Pixel Image*

Description

Plot a pixel image.

Usage

```
  ## S3 method for class 'im'
plot(x, ...,
                  main,
                  add=FALSE, clipwin=NULL,
                  col=NULL, reverse.col=FALSE,
                  valuesAreColours=NULL, log=FALSE,
                  ncolours=256, gamma=1, opacity=1,
                  ribbon=show.all, show.all=!add,
                  drop.ribbon=FALSE,
                  ribside=c("right", "left", "bottom", "top"),
                  ribsep=0.15, ribwid=0.05, ribn=1024,
                  ribscale=1, ribargs=list(), riblab=NULL, colargs=list(),
                  useRaster=NULL, workaround=FALSE, zap=1,
                  do.plot=TRUE,
                  addcontour=FALSE, contourargs=list(),
                  background=NULL, clip.background=FALSE)

  ## S3 method for class 'im'
image(x, ...,
                  main,
                  add=FALSE, clipwin=NULL,
                  col=NULL, reverse.col=FALSE,
                  valuesAreColours=NULL, log=FALSE,
                  ncolours=256, gamma=1, opacity=1,
                  ribbon=show.all, show.all=!add,
                  drop.ribbon=FALSE,
                  ribside=c("right", "left", "bottom", "top"),
                  ribsep=0.15, ribwid=0.05, ribn=1024,
                  ribscale=1, ribargs=list(), riblab=NULL, colargs=list(),
                  useRaster=NULL, workaround=FALSE, zap=1,
                  do.plot=TRUE,
                  addcontour=FALSE, contourargs=list(),
                  background=NULL, clip.background=FALSE)
```

Arguments

x	The pixel image to be plotted. An object of class "im" (see im.object).
...	Extra arguments passed to image.default to control the plot. See Details.
main	Main title for the plot.
add	Logical value indicating whether to superimpose the image on the existing plot (add=TRUE) or to initialise a new plot (add=FALSE, the default).
clipwin	Optional. A window (object of class "owin"). Only this subset of the image will be displayed.
col	Colours for displaying the pixel values. Either a character vector of colour values, an object of class colourmap, or a function as described under Details.
reverse.col	Logical value. If TRUE, the sequence of colour values specified by col will be reversed.
valuesAreColours	Logical value. If TRUE, the pixel values of x are to be interpreted as colour values.
log	Logical value. If TRUE, the colour map will be evenly-spaced on a logarithmic scale.
ncolours	Integer. The default number of colours in the colour map for a real-valued image.
gamma	Exponent for the gamma correction of the colours. A single positive number.
opacity	Numerical value between 0 and 1 specifying the degree of transparency or opaqueness of the colours. opacity=0 would make the colours completely invisible. opacity=1 (the default) makes the colours completely opaque. Not all graphics devices support transparent colours.
ribbon	Logical flag indicating whether to display a ribbon showing the colour map. Default is TRUE for new plots and FALSE for added plots.
show.all	Logical value indicating whether to display all plot elements including the main title and colour ribbon. Default is TRUE for new plots and FALSE for added plots.
drop.ribbon	Logical value. If TRUE, then a ribbon will not be displayed if all pixel values are equal.
ribside	Character string indicating where to display the ribbon relative to the main image.
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.
ribscale	Rescaling factor for tick marks. The values on the numerical scale printed beside the ribbon will be multiplied by this rescaling factor.
ribargs	List of additional arguments passed to image.default, axis and axisTicks to control the display of the ribbon and its scale axis. These may override the ... arguments.
riblab	Text to be plotted in the margin near the ribbon. A character string or expression to be interpreted as text, or a list of arguments to be passed to mtext.

colargs	List of additional arguments passed to col if it is a function.
useRaster	Logical value, passed to image.default. Images are plotted using a bitmap raster if useRaster=TRUE or by drawing polygons if useRaster=FALSE. Bitmap raster display tends to produce better results, but is not supported on all graphics devices. The default is to use bitmap raster display if it is supported.
workaround	Logical value, specifying whether to use a workaround to avoid a bug which occurs with some device drivers in R, in which the image has the wrong spatial orientation. See the section on **Image is Displayed in Wrong Spatial Orientation** below.
zap	Noise threshold factor. A numerical value greater than or equal to 1. If the range of pixel values is less than zap * .Machine$double.eps, the image will be treated as constant. This avoids displaying images which should be constant but contain small numerical errors.
do.plot	Logical value indicating whether to actually plot the image and colour ribbon. Setting do.plot=FALSE will simply return the colour map and the bounding box that were chosen for the plot.
addcontour	Logical value specifying whether to add contour lines to the image plot. The contour lines will also be drawn on the colour ribbon.
contourargs	Optional list of arguments to be passed to contour.default to control the contour plot.
background	Optional. An object to be plotted underneath the image. See the section on *Background*.
clip.background	Logical value specifying whether background should be intersected with the rectangular frame of x.

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.

Arguments ribside, ribsep, ribwid control the placement of the colour ribbon. By default, the ribbon is placed at the right of the main image. This can be changed using the argument ribside. 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 the colours representing ribn different numerical values, evenly spaced between the minimum and maximum pixel values in the image x, rendered according to the chosen colour map.

The argument ribargs controls the annotation of the colour ribbon. It is a list of arguments to be passed to image.default, axis and axisTicks. To plot the colour ribbon without the axis and tick-marks, use ribargs=list(axes=FALSE). To ensure that the numerals or symbols printed next to the colour map are oriented horizontally, use ribargs=list(las=1). To double the size of the numerals or symbols, use ribargs=list(cex.axis=2). To control the number of tick-marks, use

`ribargs=list(nint=N)` where `N` is the desired number of intervals (so there will be `N+1` tickmarks, subject to the vagaries of R internal code). To draw a box around the colour ribbon but not around the main image, use `ribargs=list(box=TRUE)`.

The argument `riblab` contains text that should be displayed in the margin next to the ribbon. Alternatively `riblab` may be a list of arguments to be passed to `mtext`; in that case `riblab$text` should specify the text to be displayed, while other arguments can specify a different location for the text (e.g. `riblab$side = 1` for text at the bottom of the ribbon) and a different font and colour if desired.

The argument `ribscale` is used to rescale the numerical values printed next to the colour map, for convenience. For example if the pixel values in `x` range between 1000 and 4000, it would be sensible to use `ribscale=1/1000` so that the colour map tickmarks would be labelled 1 to 4.

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 command `colourmap`). This is the best way to ensure that when we plot different images, the colour maps are consistent.
- a character vector or integer vector that specifies a set of colours. The colour mapping will be stretched to match the range of pixel values in the image `x`. 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 then passed to the function `image.default`. For examples of the use of these arguments, see `image.default`.
- a `function` in the R language with an argument named `range` or `inputs`.

  If `col` is a function with an argument named `range`, and if the pixel values of `x` are numeric values, then the colour values will be determined by evaluating `col(range=range(x))`. The result of this evaluation should be a character vector containing colour values, or a `"colourmap"` object. Examples of such functions are `beachcolours` and `beachcolourmap`.

  If `col` is a function with an argument named `inputs`, and if the pixel values of `x` are discrete values (integer, logical, factor or character), then the colour values will be determined by evaluating `col(inputs=p)` where `p` is the set of possible pixel values. The result should be a character vector containing colour values, or a `"colourmap"` object.
- a `function` in the R language with first argument named `n`. The colour values will be determined by evaluating `col(n)` where `n` is the number of distinct pixel values, up to a maximum

of 128. The result of this evaluation should be a character vector containing color values. Examples of such functions are heat.colors, terrain.colors, topo.colors and cm.colors.

If col is missing or col=NULL, the default colour values are the linear, perceptually uniform colour sequence given by Kovesi[[29,"values"]].

If spatstat.options("monochrome") has been set to TRUE then **all colours will be converted to grey scale values**.

If reverse.col=TRUE, the sequence of colour values specified by col will be reversed (unless valuesAreColours=TRUE).

Finally if opacity is a value less than 1, the **colours are transformed to semi-transparent colours**.

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", "xlab", "ylab" described in plot.default.
- one of the parameters "ann", "cex", "font", "cex.axis", "cex.lab", "cex.main", "cex.sub", "col.axis", "col.lab", "col.main", "col.sub", "font.axis", "font.lab", "font.main", "font.sub" described in par.
- the argument box, a logical value specifying whether a box should be drawn.

Images are plotted using a bitmap raster if useRaster=TRUE or by drawing polygons if useRaster=FALSE. Bitmap raster display (performed by rasterImage) tends to produce better results, but is not supported on all graphics devices. The default is to use bitmap raster display if it is supported according to dev.capabilities.

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".

Also has an attribute "bbox" giving a bounding box for the plot (containing the main colour image, the colour ribbon if plotted, and the background if given). If a ribbon was plotted, there is also an attribute "bbox.legend" giving a bounding box for the ribbon image. Text annotation occurs outside these bounding boxes.

Also has an attribute "at" giving the axis positions of the tick marks next to the colour ribbon.

Adding contour lines

If addcontour=TRUE, contour lines will be superimposed on the image. Lines will also be superimposed on the colour ribbon at the corresponding positions. The display of the contour lines can be controlled by arguments specified in contourargs.

Main title, box and white space

The left-right placement of the main title is controlled by the argument adj.main passed to plot.owin.

A rectangular box surrounding the image domain can be added by setting box=TRUE.

White space around the plot is controlled by par('mar').

Complex-valued images

If the pixel values in x are complex numbers, they will be converted into four images containing the real and imaginary parts and the modulus and argument, and plotted side-by-side using plot.imlist.

Monochrome colours

If spatstat.options("monochrome") has been set to TRUE, then **the image will be plotted in greyscale**. The colours are converted to grey scale values using to.grey. The choice of colour map still has an effect, since it determines the final grey scale values.

Monochrome display can also be achieved by setting the graphics device parameter colormodel="grey" when starting a new graphics device, or in a call to ps.options or pdf.options.

Background

The argument background specifies an object that will be plotted before the image x is plotted, and will therefore appear underneath it. The background can be any spatial object in the **spatstat** package family. To specify graphical arguments for the background, make it into a layered object using layered or as.layered, and assign graphical parameters to it using layerplotargs.

Troubleshooting: Image Looks Like Noise

An image plot which looks like digital noise can be produced when the pixel values are almost exactly equal but include a tiny amount of numerical error. To check this, look at the numerals plotted next to the colour ribbon, or compute diff(range(x)), to determine whether the range of pixel values is almost zero. The behaviour can be suppressed by picking a larger value of the argument zap.

Troubleshooting: Image Rendering Errors and Problems

The help for image.default and rasterImage 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.

If the image is not displayed at all, try setting useRaster=FALSE in the call to plot.im. If the ribbon colours are not displayed, set ribargs=list(useRaster=FALSE).

Errors 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)`.

Troubleshooting: Image is Displayed in Wrong Spatial Orientation

If the image is displayed in the wrong spatial orientation, and you created the image data directly, please check that you understand the **spatstat** convention for the spatial orientation of pixel images. The row index of the matrix of pixel values corresponds to the increasing y coordinate; the column index of the matrix corresponds to the increasing x coordinate (Baddeley, Rubak and Turner, 2015, section 3.6.3, pages 66–67).

Images can be displayed in the wrong spatial orientation on some devices, due to a bug in the device driver. This occurs only when the plot coordinates are *reversed*, that is, when the plot was initialised with coordinate limits `xlim, ylim` such that `xlim[1] > xlim[2]` or `ylim[1] > ylim[2]` or both. This bug is reported to occur only when `useRaster=TRUE`. To fix this, try setting `workaround=TRUE`, or if that is unsuccessful, `useRaster=FALSE`.

Troubleshooting: text is clipped

Text that is intended to be drawn in the margins of the plot can sometimes be clipped so that it is only partly visible or is completely missing. Clipping is not an artefact of **spatstat**; it is a feature of base graphics in R. To prevent clipping of text, first ensure that the plot margins are large enough by setting `par('mar')`. In base graphics, some plot functions have the side-effect that they re-set the clipping region to be a sub-rectangle of the plotting region. To override this and ensure that the clipping region is as large as possible, set `par(xpd=NA)` immediately before calling `plot.im`.

Author(s)

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

References

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

See Also

`im.object`, `colourmap`, `contour.im`, `persp.im`, `hist.im`, `image.default`, `spatstat.options`, `default.image.colours`

Examples

```
# an image
Z <- setcov(owin())
plot(Z)
```

```
plot(Z, ribside="bottom")
# stretchable colour map
plot(Z, col=rainbow)
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)
# colour map function, with argument 'range'
plot(Z, col=beachcolours, colargs=list(sealevel=0.5))
# tweaking the plot
plot(Z, main="La vie en bleu", col.main="blue", cex.main=1.5,
     box=TRUE,
     ribargs=list(col.axis="blue", col.ticks="blue", cex.axis=0.75, box=FALSE),
     riblab=list(text="couleur", side=1, col="blue", offset=1, font=2))
# add axes and axis labels
plot(Z, axes=TRUE, ann=TRUE, xlab="Easting", ylab="Northing")
# add contour lines
plot(Z, addcontour=TRUE, contourargs=list(col="white", drawlabels=FALSE))
# log scale
V <- eval.im(exp(exp(Z+2))/1e4)
plot(V, log=TRUE, main="Log scale")
# it's complex
Y <- exp(Z + V * 1i)
plot(Y)
```

`plot.imlist` *Plot a List of Images*

Description

Plots an array of pixel images.

Usage

```
  ## S3 method for class 'imlist'
plot(x, ..., plotcommand="image",
                          equal.ribbon=FALSE,
                          equal.scales=FALSE,
                          ribmar=NULL)

  ## S3 method for class 'imlist'
image(x, ..., equal.ribbon=FALSE,
                                  equal.scales=FALSE, ribmar=NULL)

  ## S3 method for class 'listof'
image(x, ..., equal.ribbon=FALSE,
                                  equal.scales=FALSE, ribmar=NULL)
```

Arguments

x	An object of the class "imlist" representing a list of pixel images. Alternatively x may belong to the outdated class "listof".
...	Arguments passed to plot.solist to control the spatial arrangement of panels, and arguments passed to plot.im to control the display of each panel.
equal.ribbon	Logical. If TRUE, the colour maps of all the images will be the same. If FALSE, the colour map of each image is adjusted to the range of values of that image.
equal.scales	Logical. If TRUE, the images will be plotted using the same physical scale, and the plots will be aligned neatly where possible.
ribmar	Numeric vector of length 4 specifying the margins around the colour ribbon, if equal.ribbon=TRUE. Entries in the vector give the margin at the bottom, left, top, and right respectively, as a multiple of the height of a line of text.
plotcommand	Character string giving the name of a function to be used to display each image. Recognised by plot.imlist only.

Details

These are methods for the generic plot commands plot and image for the class "imlist". They are currently identical.

An object of class "imlist" represents a list of pixel images. (The outdated class "listof" is also handled.)

Each entry in the list x will be displayed as a pixel image, in an array of panels laid out on the same graphics display, using plot.solist. Individual panels are plotted by plot.im.

If equal.ribbon=FALSE (the default), the images are rendered using different colour maps, which are displayed as colour ribbons beside each image. If equal.ribbon=TRUE, the images are rendered using the same colour map, and a single colour ribbon will be displayed at the right side of the array. The colour maps and the placement of the colour ribbons are controlled by arguments ... passed to plot.im.

If equal.scales=TRUE, the images are plotted using the same physical scale, and the plots will be aligned neatly where possible. If equal.scales=FALSE (the default), images are plotted using equal amounts of space in the available plotting area, so they may be plotted at different physical scales.

Value

Null.

Author(s)

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

See Also

plot.solist, plot.im

Examples

```
## bei.extra is a list of pixel images on the same spatial domain
Y <- solapply(bei.extra, scaletointerval)
image(Y, equal.ribbon=TRUE, equal.scales=TRUE,
         main="",
         mar.panel=0, hsep=1,
         ribside="bottom",
         col.ticks="blue", col.axis="blue", cex.axis=1.2)
```

`plot.layered` *Layered Plot*

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,
                          add=FALSE, show.all=!add, main=NULL,
                          do.plot=TRUE)
```

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`.
`add`	Logical value indicating whether to add the graphics to an existing plot.
`show.all`	Logical value indicating whether the *first* layer should be displayed in full (including the main title, bounding window, coordinate axes, colour ribbon, and so on).
`main`	Main title for the plot
`do.plot`	Logical value indicating whether to actually do the plotting.

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, to control the plotting arguments that are passed to each layer, and to zoom in on a subregion.

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`.

To zoom in on a subregion, apply the subset operator [.layered to x before plotting.

Graphics parameters for each layer are determined by (in order of precedence) ..., plotargs, and layerplotargs(x).

The graphics parameters may also include the special argument .plot specifying (the name of) a function which will be used to perform the plotting instead of the generic plot.

The argument show.all is recognised by many plot methods in **spatstat**. It determines whether a plot is drawn with all its additional components such as the main title, bounding window, coordinate axes, colour ribbons and legends. The default is TRUE for new plots and FALSE for added plots.

In plot.layered, the argument show.all applies only to the **first** layer. The subsequent layers are plotted with show.all=FALSE.

To override this, that is, if you really want to draw all the components of **all** layers of x, insert the argument show.all=TRUE in each entry of plotargs or layerplotargs(x).

Value

(Invisibly) a list containing the return values from the plot commands for each layer. This list has an attribute "bbox" giving a bounding box for the entire plot.

Author(s)

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

See Also

layered, layerplotargs, [.layered, plot.

Examples

```
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")))
# plot a subregion
plot(L[, square(0.5)])
```

---

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), hsep=0, vsep=0,
   panel.begin=NULL, panel.end=NULL, panel.args=NULL,
   panel.begin.args=NULL, panel.end.args=NULL, panel.vpad=0.2,
   plotcommand="plot",
   do.plot=TRUE,
   adorn.left=NULL, adorn.right=NULL, adorn.top=NULL, adorn.bottom=NULL,
   adorn.size=0.2, adorn.args=list(),
   equal.scales=FALSE, halign=FALSE, valign=FALSE)
```

Arguments

x	An object of the class "listof". Essentially a list of objects.
...	Arguments passed to plot when generating each plot panel.
main	Overall heading for the plot.
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.
main.panel	Optional. A character string, or a vector of character strings, giving the headings for each of the objects.
mar.panel	Size of the margins outside each plot panel. A numeric vector of length 4 giving the bottom, left, top, and right margins in that order. (Alternatively the vector may have length 1 or 2 and will be replicated to length 4). See the section on *Spacing between plots*.
hsep, vsep	Additional horizontal and vertical separation between plot panels, expressed in the same units as mar.panel.
panel.begin, panel.end	Optional. Functions that will be executed before and after each panel is plotted. See Details.
panel.args	Optional. Function that determines different plot arguments for different panels. See Details.
panel.begin.args	Optional. List of additional arguments for panel.begin when it is a function.
panel.end.args	Optional. List of additional arguments for panel.end when it is a function.
panel.vpad	Amount of extra vertical space that should be allowed for the title of each panel, if a title will be displayed. Expressed as a fraction of the height of the panel. Applies only when equal.scales=FALSE (the default) and requires that the height of each panel can be determined.
plotcommand	Optional. Character string containing the name of the command that should be executed to plot each panel.

do.plot	Logical value specifying whether to actually perform the plot.
adorn.left, adorn.right, adorn.top, adorn.bottom	Optional. Functions (with no arguments) that will be executed to generate additional plots at the margins (left, right, top and/or bottom, respectively) of the array of plots.
adorn.size	Relative width (as a fraction of the other panels' widths) of the margin plots.
adorn.args	Optional list of arguments passed to the functions adorn.left, adorn.right, adorn.top and adorn.bottom.
equal.scales	Logical value indicating whether the components should be plotted at (approximately) the same physical scale.
halign, valign	Logical values indicating whether panels in a column should be aligned to the same x coordinate system (halign=TRUE) and whether panels in a row should be aligned to the same y coordinate system (valign=TRUE). These are applicable only if equal.scales=TRUE.

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, various functions produce an object of class "listof", essentially a list of spatial objects of the same kind. These objects can be plotted in a nice arrangement using plot.listof. See the Examples.

The argument panel.args determines extra graphics parameters for each panel. It should be a function that will be called as panel.args(i) where i is the panel number. Its return value should be a list of graphics parameters that can be passed to the relevant plot method. These parameters override any parameters specified in the ... arguments.

The arguments panel.begin and panel.end determine graphics that will be plotted before and after each panel is plotted. They may be objects of some class that can be plotted with the generic plot command. Alternatively they may be functions that will be called as panel.begin(i, y, main=main.panel[i]) and panel.end(i, y, add=TRUE) where i is the panel number and y = x[[i]].

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.

If equal.scales=FALSE (the default), then the plot panels will have equal height on the plot device (unless there is only one column of panels, in which case they will have equal width on the plot device). This means that the objects are plotted at different physical scales, by default.

If equal.scales=TRUE, then the dimensions of the plot panels on the plot device will be proportional to the spatial dimensions of the corresponding components of x. This means that the objects will be plotted at *approximately* equal physical scales. If these objects have very different spatial sizes, the plot command could fail (when it tries to plot the smaller objects at a tiny scale), with an error message that the figure margins are too large.

The objects will be plotted at *exactly* equal physical scales, and *exactly* aligned on the device, under the following conditions:

- every component of x is a spatial object whose position can be shifted by shift;
- panel.begin and panel.end are either NULL or they are spatial objects whose position can be shifted by shift;
- adorn.left, adorn.right, adorn.top and adorn.bottom are all NULL.

Another special case is when every component of x is an object of class "fv" representing a function. If equal.scales=TRUE then all these functions will be plotted with the same axis scales (i.e. with the same xlim and the same ylim).

Value

Null.

Spacing between plots

The spacing between individual plots is controlled by the parameters mar.panel, hsep and vsep.

If equal.scales=FALSE, the plot panels are logically separate plots. The margins for each panel are determined by the argument mar.panel which becomes the graphics parameter mar described in the help file for par. One unit of mar corresponds to one line of text in the margin. If hsep or vsep are present, mar.panel is augmented by c(vsep, hsep, vsep, hsep)/2.

If equal.scales=TRUE, all the plot panels are drawn in the same coordinate system which represents a physical scale. The unit of measurement for mar.panel[1,3] is one-sixth of the greatest height of any object plotted in the same row of panels, and the unit for mar.panel[2,4] is one-sixth of the greatest width of any object plotted in the same column of panels. If hsep or vsep are present, they are interpreted in the same units as mar.panel[2] and mar.panel[1] respectively.

Error messages

If the error message 'Figure margins too large' occurs, this generally means that one of the objects had a much smaller physical scale than the others. Ensure that equal.scales=FALSE and increase the values of mar.panel.

Author(s)

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

See Also

print.listof, contour.listof, image.listof, density.splitppp

Examples

```
 D <- solapply(split(mucosa), distfun)

 plot(D)
 plot(D, main="", equal.ribbon=TRUE,
      panel.end=function(i,y,...){contour(y, ..., drawlabels=FALSE)})

# list of 3D point patterns
 ape1 <- osteo[osteo$shortid==4, "pts", drop=TRUE]
```

```
class(ape1)
plot(ape1, main.panel="", mar.panel=0.1, hsep=0.7, vsep=1,
     cex=1.5, pch=21, bg='white')
```

---

plot.onearrow *Plot an Arrow*

Description

Plots an object of class "onearrow".

Usage

```
 ## S3 method for class 'onearrow'
plot(x, ...,
  add = FALSE, main = "",
  retract = 0.05, headfraction = 0.25, headangle = 12, headnick = 0.1,
  col.head = NA, lwd.head = lwd, lwd = 1, col = 1,
  zap = FALSE, zapfraction = 0.07,
  pch = 1, cex = 1, do.plot = TRUE, do.points = FALSE, show.all = !add)
```

Arguments

x	Object of class "onearrow" to be plotted. This object is created by the command onearrow.
...	Additional graphics arguments passed to segments to control the appearance of the line.
add	Logical value indicating whether to add graphics to the existing plot (add=TRUE) or to start a new plot (add=FALSE).
main	Main title for the plot.
retract	Fraction of length of arrow to remove at each end.
headfraction	Length of arrow head as a fraction of overall length of arrow.
headangle	Angle (in degrees) between the outer edge of the arrow head and the shaft of the arrow.
headnick	Size of the nick in the trailing edge of the arrow head as a fraction of length of arrow head.
col.head, lwd.head	Colour and line style of the filled arrow head.
col, lwd	Colour and line style of the arrow shaft.
zap	Logical value indicating whether the arrow should include a Z-shaped (lightning-bolt) feature in the middle of the shaft.
zapfraction	Size of Z-shaped deviation as a fraction of total arrow length.
pch, cex	Plot character and character size for the two end points of the arrow, if do.points=TRUE.
do.plot	Logical. Whether to actually perform the plot.
do.points	Logical. Whether to display the two end points of the arrow as well.
show.all	Internal use only.

Details

The argument x should be an object of class "onearrow" created by the command onearrow.

Value

A window (class "owin") enclosing the plotted graphics.

Author(s)

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

See Also

onearrow, yardstick

Examples

```
oa <- onearrow(cells[c(1, 42)])
oa
plot(oa)
plot(oa, zap=TRUE, do.points=TRUE, col.head="pink", col="red")
```

---

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,
                       type=c("w","n"), show.all=!add,
                       hatch=FALSE,
                       hatchargs=list(),
                       invert=FALSE, do.plot=TRUE,
                       claim.title.space=FALSE, use.polypath=TRUE,
                       adj.main=0.5,
                       background=NULL)
```

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 controlling the appearance of the plot. These arguments are passed to polygon if x is a polygonal or rectangular window, or passed to image.default if x is a binary mask. Some arguments are passed to plot.default. See Details.
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.
type	Type of plot: either "w" or "n". If type="w" (the default), the window is plotted. If type="n" and add=TRUE, a new plot is initialised and the coordinate system is established, but nothing is drawn.
show.all	Logical value indicating whether to plot everything including the main title.
hatch	logical flag; if TRUE, the interior of the window will be shaded by texture, such as a grid of parallel lines.
hatchargs	List of arguments passed to add.texture to control the texture shading when hatch=TRUE.
invert	logical flag; when the window is a binary pixel mask, the mask colours will be inverted if invert=TRUE.
do.plot	Logical value indicating whether to actually perform the plot.
claim.title.space	Logical value indicating whether extra space for the main title should be allocated when declaring the plot dimensions. Should be set to FALSE under normal conditions.
use.polypath	Logical value indicating what graphics capabilities should be used to draw a polygon filled with colour when the polygon has holes. If TRUE (the default), then the polygon will be filled using polypath, provided the graphics device supports this function. If FALSE, the polygon will be decomposed into simple closed polygons, which will be colour filled using polygon.
adj.main	Numeric value specifying the justification of the text in the main title. Possible values are adj.main=0.5 (the default) specifying that the main title will be centred, adj.main=0 specifying left-justified text, and adj.main=1 specifying right-justified text.
background	Optional. An object to be plotted underneath the main plot. See the section on *Background*.

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.

If add=FALSE (the default), the plot is initialised by calling the base graphics function plot.default to create the plot area. By default, coordinate axes and axis labels are not plotted. To plot coordinate axes, use the argument axes=TRUE; to plot axis labels, use the argument ann=TRUE and then specify the labels with xlab and ylab; see the help file for plot.default for information on these arguments, and for additional arguments controlling the appearance of the axes. See the Examples also.

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.

Background

The argument background specifies an object that will be plotted before the window x is plotted, and will therefore appear underneath it. The background can be any spatial object in the **spatstat** package family. To specify graphical arguments for the background, make it into a layered object using layered or as.layered, and assign graphical parameters to it using layerplotargs.

Alternatively background can be a single colour value, indicating that the plot should have this background colour.

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.

polygon decomposition: The other workaround involves decomposing the polygonal window into pieces which do not have holes. This code is experimental but works in all our test cases. If this code fails, a warning will be issued, and the filled colours will not be plotted.

Troubleshooting: text is clipped

Text that is intended to be drawn in the margins of the plot can sometimes be clipped so that it is only partly visible or is completely missing. Clipping is not an artefact of **spatstat**; it is a feature of base graphics in R. To prevent clipping of text, first ensure that the plot margins are large enough by setting `par('mar')`. In base graphics, some plot functions have the side-effect that they re-set the clipping region to be a sub-rectangle of the plotting region. To override this and ensure that the clipping region is as large as possible, set `par(xpd=NA)` immediately before calling `plot.owin`.

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 `www.cairographics.org` for instructions on obtaining and installing **cairo**. After having installed **cairo** one needs to re-install R from source so that it has **cairo** capabilites. 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@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

`owin.object`, `plot.ppp`, `polygon`, `image.default`, `spatstat.options`

Examples

```
# rectangular window
 plot(Window(nztrees))
 abline(v=148, lty=2)

# polygonal window
w <- Window(demopat)
plot(w)
plot(w, col="red", border="green", lwd=2)
plot(w, hatch=TRUE, lwd=2)

# binary mask
we <- as.mask(w)
plot(we)
```

```
op <- spatstat.options(par.binary=list(col=grey(c(0.5,1))))
plot(we)
spatstat.options(op)

## axis annotation
plot(letterR, axes=TRUE, ann=TRUE, xlab="Easting", ylab="Northing")
plot(letterR,            ann=TRUE, xlab="Declination", ylab="Right Ascension")
```

`plot.pp3` *Plot a Three-Dimensional Point Pattern*

Description

Plots a three-dimensional point pattern.

Usage

```
 ## S3 method for class 'pp3'
plot(x, ..., eye=NULL, org=NULL, theta=25, phi=15,
                    type=c("p", "n", "h"),
                    box.back=list(col="pink"),
                    box.front=list(col="blue", lwd=2))
```

Arguments

`x`	Three-dimensional point pattern (object of class `"pp3"`).
`...`	Arguments passed to `points` controlling the appearance of the points.
`eye`	Optional. Eye position. A numeric vector of length 3 giving the location from which the scene is viewed.
`org`	Optional. Origin (centre) of the view. A numeric vector of length 3 which will be at the centre of the view.
`theta, phi`	Optional angular coordinates (in degrees) specifying the direction from which the scene is viewed: `theta` is the azimuth and `phi` is the colatitude. Ignored if `eye` is given.
`type`	Type of plot: `type="p"` for points, `type="h"` for points on vertical lines, `type="n"` for box only.
`box.front, box.back`	How to plot the three-dimensional box that contains the points. A list of graphical arguments passed to `segments`, or a logical value indicating whether or not to plot the relevant part of the box. See Details.

Details

This is the plot method for objects of class "pp3". It generates a two-dimensional plot of the point pattern x and its containing box as if they had been viewed from the location specified by eye (or from the direction specified by theta and phi).

The edges of the box at the 'back' of the scene (as viewed from the eye position) are plotted first. Then the points are added. Finally the remaining 'front' edges are plotted. The arguments box.back and box.front specify graphical parameters for drawing the back and front edges, respectively. Alternatively box.back=FALSE specifies that the back edges shall not be drawn.

Note that default values of arguments to plot.pp3 can be set by spatstat.options("par.pp3").

Value

Null.

Author(s)

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

See Also

pp3, spatstat.options.

Examples

```
X <- osteo$pts[[1]]
plot(X, main="Osteocyte lacunae, animal 1, brick 1",
     cex=1.5, pch=16)
plot(X, type="h", main="", box.back=list(lty=3))
```

---

plot.ppp *plot a Spatial Point Pattern*

Description

Plot a two-dimensional spatial point pattern

Usage

```
 ## S3 method for class 'ppp'
plot(x, main, ..., clipwin=NULL,
                   chars=NULL, cols=NULL,
                   use.marks=TRUE, which.marks=NULL,
                   add=FALSE, type=c("p","n"),
                   legend=TRUE,
                   leg.side=c("left", "bottom", "top", "right"),
                   leg.args=list(),
```

```
symap=NULL, maxsize=NULL, meansize=NULL, markscale=NULL,
minsize=NULL, zerosize=NULL,
zap=0.01,
show.window=show.all, show.all=!add,
show.rejects=TRUE, do.plot=TRUE,
multiplot=TRUE,
background=NULL, clip.background=FALSE,
scramble.cols=FALSE)
```

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. Not all arguments will be recognised.
clipwin	Optional. A window (object of class "owin"). Only this subset of the image will be displayed.
chars	the plotting character(s) used to plot points. Either a single character, an integer, or a vector of single characters or integers. Ignored if symap is given.
cols	the colour(s) used to plot points. Either an integer index from 1 to 8 (indexing the standard colour palette), a character string giving the name of a colour, or a string giving the hexadecimal representation of a colour, or a vector of such integers or strings. See the section on *Colour Specification* in the help for par. Ignored if symap is given.
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 or integer vector identifying one or more columns of marks. If add=FALSE then the default is to plot all columns of marks, in a series of separate plots. If add=TRUE then only one column of marks can be plotted, and the default is which.marks=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.
type	Type of plot: either "p" or "n". If type="p" (the default), both the points and the observation window are plotted. If type="n", only the window is plotted.
legend	Logical value indicating whether to add a legend showing the mapping between mark values and graphical symbols (for a marked point pattern).
leg.side	Position of legend relative to main plot.
leg.args	List of additional arguments passed to plot.symbolmap or symbolmap to control the legend. In addition to arguments documented under plot.symbolmap, and graphical arguments recognised by symbolmap, the list may also include the argument sep giving the separation between the main plot and the legend, or sep.frac giving the separation as a fraction of the largest dimension (maximum of width and height) of the main plot.

symap	The graphical symbol map to be applied to the marks. An object of class "symbolmap"; see symbolmap.
maxsize	*Maximum* physical size of the circles/squares plotted when x is a marked point pattern with numerical marks. Incompatible with meansize and markscale. Ignored if symap is given.
meansize	*Average* physical size of the circles/squares plotted when x is a marked point pattern with numerical marks. Incompatible with maxsize and markscale. Ignored if symap is given.
markscale	physical scale factor determining the sizes of the circles/squares plotted when x is a marked point pattern with numerical marks. Mark value will be multiplied by markscale to determine physical size. Incompatible with maxsize and meansize. Ignored if symap is given.
minsize	*Minimum* physical size of the circles/squares plotted when x is a marked point pattern with numerical marks. Incompatible with zerosize. Ignored if symap is given.
zerosize	Physical size of the circle/square representing a mark value of zero, when x is a marked point pattern with numerical marks. Incompatible with minsize. Defaults to zero. Ignored if symap is given.
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(abs(marks(x))) will not be plotted.
show.window	Logical value indicating whether to plot the observation window of x.
show.all	Logical value indicating whether to plot everything including the main title and the observation window of x.
show.rejects	Logical value indicating whether to plot attr(x, "rejects"). See the section on Rejected Points.
do.plot	Logical value determining whether to actually perform the plotting.
multiplot	Logical value giving permission to display multiple plots.
background	Optional. An object to be plotted underneath the main plot. See the section on *Background*.
clip.background	Logical value specifying whether background should be intersected with the rectangular frame of x.
scramble.cols	Logical value. If TRUE, the sequence of colour values will be randomly permuted. This is a useful trick when it is desired that adjacent colours in the sequence should be easily distinguishable. Applicable only when x is a multitype point pattern.

Details

This is the plot method for point pattern datasets (of class "ppp", see ppp.object).

First the observation window Window(x) is plotted (if show.window=TRUE). 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 `marks(x)` is a factor, then each level of the factor is represented by a different plot character.

continuous marks: If `marks(x)` is a numeric vector, the marks are rescaled to the unit interval and each point is represented by a circle with *diameter* proportional to the rescaled mark (if the value is positive) or a square with *side length* proportional to the absolute value of the rescaled mark (if the value is negative).

other kinds of marks: If `marks(x)` is neither numeric nor a factor, then each possible mark will be represented by a different plotting character. The default is to represent the ith smallest mark value by `points(..., pch=i)`.

If there are several columns of marks, and if `which.marks` is missing or `NULL`, then

- if `add=FALSE` and `multiplot=TRUE` the default is to plot all columns of marks, in a series of separate plots, placed side-by-side. The plotting is coordinated by `plot.listof`, which calls `plot.ppp` to make each of the individual plots.
- Otherwise, only one column of marks can be plotted, and the default is `which.marks=1` indicating the first column of marks.

Plotting of the window `Window(x)` is performed by `plot.owin`. This plot may be modified through the `...` arguments. In particular the extra argument `border` determines the colour of the window, if the window is not a binary mask.

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 `...` arguments.

If the argument `symap` is given, then it determines the graphical display of the points. It should be a symbol map (object of class `"symbolmap"`) created by the function `symbolmap`.

If `symap` is not given, then the following arguments can be used to specify how the points are plotted:

- 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(marks(x))`, and then the ith 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, `cols` 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(marks(x))`: that is, there is one colour for each possible mark value. The ith level or type will be plotted using colour `cols[i]`. For a point pattern with continuous marks, `cols` can be either a character string or a character vector specifying colour values: the range of mark values will be mapped to the specified colours.
- If `cols` is absent, the colours used to plot 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 specifying `col` will also apply this colour to the window itself.

- The default colour for the points is a semi-transparent grey, if this is supported by the plot device. This behaviour can be suppressed (so that the default colour is non-transparent) by setting `spatstat.options(transparent=FALSE)`.
- The arguments `maxsize`, `meansize` and `markscale` are incompatible with each other (and incompatible with `symap`). The arguments `minsize` and `zerosize` are incompatible with each other (and incompatible with `symap`). Together, these arguments control the physical size of the circles and squares which represent the marks in a point pattern with continuous marks. The size of a circle is defined as its *diameter*; the size of a square is its side length. If `markscale` is given, then a mark value of `m` is plotted as a circle of diameter `m * markscale + zerosize` (if `m` is positive) or a square of side `abs(m) * markscale + zerosize` (if `m` is negative). If `maxsize` is given, then the largest mark in absolute value, `mmax=max(abs(marks(x)))`, will be scaled to have physical size `maxsize`. If `meansize` is given, then the average absolute mark value, `mmean=mean(abs(marks(x)))`, will be scaled to have physical size `meansize`. If `minsize` is given, then the minimum mark value, `mmean=mean(abs(marks(x)))`, will be scaled to have physical size `minsize`.
- 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 is an object of class `"symbolmap"` representing the mapping from mark values to graphical symbols. See `symbolmap`. It can be used to make a suitable legend, or to ensure that two plots use the same graphics map.

Value

(Invisible) object of class `"symbolmap"` giving the correspondence between mark values and plotting characters.

Layout of the plot

- **Placement of main title:** The left-right placement of the main title is controlled by the argument `adj.main` passed to `plot.owin`. To remove the main title, set `main=""`.
- **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.
- **Drawing coordinate axes and axis labels:** Coordinate axes and axis labels are not drawn, by default. To draw coordinate axes, set `axes=TRUE`. To draw axis labels, set `ann=TRUE` and give values to the arguments `xlab` and `ylab`. See the Examples. Only the default style of axis is supported; for more control over the placement and style of axes, use the graphics commands `axis` and `mtext`.

The Symbol Map

The behaviour of `plot.ppp` is different from the behaviour of the base R graphics functions `points` and `symbols`.

In the base graphics functions points and symbols, arguments such as col, pch and cex can be vectors which specify the *representation of each successive point*. For example col[3] would specify the colour of the third point in the sequence of points. If there are 100 points then col should be a vector of length 100.

In the **spatstat** function plot.ppp, arguments such as col, pch and cex specify the *mapping from point characteristics to graphical parameters* (called the symbol map). For example col[3] specifies the colour of the third **type of point** in a pattern of points of different types. If there are 4 types of points then col should be a vector of length 4.

To modify a symbol map, for example to change the colours used without changing anything else, use update.symbolmap.

Scrambling the colours

When x is a multitype point pattern and cols is given, the argument scramble.cols=TRUE will cause the sequence of colours in the colour map to be randomly permuted. This is a useful trick when it is desired that adjacent colours in the sequence should be easily distinguishable. The return value of plot.ppp will include the randomised colour sequence rather than the original colour sequence.

Randomisation implies that the appearance of the plot, and the return value, will be different each time the command is executed. For reproducible results, set the random generator seed using set.seed.

Background

The argument background specifies an object that will be plotted before the point pattern x is plotted, and will therefore appear underneath it. The background can be any spatial object in the **spatstat** package family. To specify graphical arguments for the background, make it into a layered object using layered or as.layered, and assign graphical parameters to it using layerplotargs.

Alternatively background can be a single colour value, indicating that the plot should have this background colour.

Troubleshooting: text is clipped

Text that is intended to be drawn in the margins of the plot can sometimes be clipped so that it is only partly visible or is completely missing. Clipping is not an artefact of **spatstat**; it is a feature of base graphics in R. To prevent clipping of text, first ensure that the plot margins are large enough by setting par('mar'). In base graphics, some plot functions have the side-effect that they re-set the clipping region to be a sub-rectangle of the plotting region. To override this and ensure that the clipping region is as large as possible, set par(xpd=NA) immediately before calling plot.ppp.

Rejected Points

A point pattern object x may have an attribute attr(x, "rejects"). This contains points that were rejected when x was created, because they lie outside the window of x. See ppp for more explanation.

When plot(x) is executed, the rejected points will also be plotted if show.rejects=TRUE (the default).

Author(s)

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

See Also

ppp.object, default.symbolmap.ppp.

plot, par, points, text.ppp, plot.owin, symbols.

See also the command iplot in the **spatstat.gui** package.

Examples

```
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
plot(lansing)

# marked by a real number
plot(longleaf)

# just plot the points
plot(longleaf, use.marks=FALSE)
plot(unmark(longleaf)) # equivalent

# point pattern with multiple marks
plot(finpines)
plot(finpines, which.marks="height")

# controlling COLOURS of points
plot(cells, cols="blue")
plot(lansing, cols=c("black", "yellow", "green",
                     "blue","red","pink"))
plot(longleaf, fg="blue")
plot(lansing, cols=rainbow, scramble.cols=TRUE)

# make window purple
plot(lansing, border="purple")
# make everything purple
plot(lansing, border="purple", cols="purple", col.main="purple",
              leg.args=list(col.axis="purple"))
```

```
# controlling PLOT CHARACTERS for multitype pattern
plot(lansing, chars = 11:16)
plot(lansing, chars = c("o","h","m",".","o","o"))

## multitype pattern mapped to symbols
plot(amacrine, shape=c("circles", "squares"), size=0.04)
plot(amacrine, shape="arrows", direction=c(0,90), size=0.07)

## plot trees as trees!
plot(lansing, shape="arrows", direction=90, cols=1:6)

# controlling MARK SCALE for pattern with numeric marks
plot(longleaf, markscale=0.1)
plot(longleaf, maxsize=5)
plot(longleaf, meansize=2)
plot(longleaf, minsize=2)

# draw circles of diameter equal to nearest neighbour distance
plot(cells %mark% nndist(cells), markscale=1, legend=FALSE)

# inspecting the symbol map
v <- plot(amacrine)
v

## variable colours ('cols' not 'col')
plot(longleaf, cols=function(x) ifelse(x < 30, "red", "black"))

## re-using the same mark scale
a <- plot(longleaf)
juveniles <- longleaf[marks(longleaf) < 30]
plot(juveniles, symap=a)

## numerical marks mapped to symbols of fixed size with variable colour
ra <- range(marks(longleaf))
colmap <- colourmap(terrain.colors(20), range=ra)
## filled plot characters are the codes 21-25
## fill colour is indicated by 'bg'
## outline colour is 'fg'
sy <- symbolmap(pch=21, bg=colmap, fg=colmap, range=ra)
plot(longleaf, symap=sy)

## or more compactly..
plot(longleaf, bg=terrain.colors(20), pch=21, cex=1)

## plot only the colour map (since the symbols have fixed size and shape)
plot(longleaf, symap=sy, leg.args=list(colour.only=TRUE))

## plot with a background
plot(bei %mark% nndist(bei), background=bei.extra$elev)

## clipping
plot(humberside)
B <- owin(c(4810, 5190), c(4180, 4430))
```

```
plot(B, add=TRUE, border="red")
plot(humberside, clipwin=B, main="Humberside (clipped)")

## coordinate axes and labels
plot(humberside, axes=TRUE)
plot(humberside,            ann=TRUE, xlab="Easting", ylab="Northing")
plot(humberside, axes=TRUE, ann=TRUE, xlab="Easting", ylab="Northing")
```

`plot.pppmatching` *Plot a Point Matching*

Description

Plot an object of class `"pppmatching"` which represents a matching of two planar point patterns.

Usage

```
## S3 method for class 'pppmatching'
plot(x, addmatch = NULL, main = NULL, ..., adjust = 1)
```

Arguments

`x`	Point pattern matching object (class `"pppmatching"`) to be plotted.
`addmatch`	Optional. A matrix indicating additional pairs of points that should be matched. See Details.
`main`	Main title for the plot.
`...`	Additional arguments passed to other plot methods.
`adjust`	Adjustment factor for the widths of line segments. A positive number.

Details

The object `x` represents a matching found between two point patterns `X` and `Y`. The matching may be incomplete. See `pppmatching.object` for further description.

This function plots the matching by drawing the two point patterns `X` and `Y` as red and blue dots respectively, and drawing line segments between each pair of matched points. The width of the line segments is proportional to the strength of matching. The proportionality constant can be adjusted using the argument `adjust`.

Additional graphics arguments `...` control the plotting of the window (and are passed to `plot.owin`) and the plotting of the line segments (and are passed to `plot.psp`, and ultimately to the base graphics function `polygon`).

The argument `addmatch` is for use mainly by developers to study algorithms which update the matching. If `addmatch` is given, it should be a matrix with dimensions `npoints(X) * npoints(Y)`. If `addmatch[i,j] > 0` then a light grey line segment will be drawn between `X[i]` and `Y[j]`.

Value

Null.

Author(s)

Dominic Schuhmacher and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>.

See Also

pppmatching.object

Examples

```
X <- runifrect(7)
Y <- runifrect(7)
am <- r2dtable(1, rep(10,7), rep(10,7))[[1]]/10
m2 <- pppmatching(X, Y, am)
plot(m2, adjust=0.3)
```

---

plot.psp *plot a Spatial Line Segment Pattern*

Description

Plot a two-dimensional line segment pattern

Usage

```
  ## S3 method for class 'psp'
plot(x, ..., main, add=FALSE,
                    show.all=!add, show.window=show.all, do.plot=TRUE,
                    use.marks=TRUE,
                    which.marks=1,
                    style=c("colour", "width", "none"),
                    col=NULL,
                    ribbon=show.all,
                    ribsep=0.15, ribwid=0.05, ribn=1024,
                    scale=NULL, adjust=1,
                    legend=TRUE,
                    leg.side=c("right", "left", "bottom", "top"),
                    leg.sep=0.1,
                    leg.wid=0.1,
                    leg.args=list(),
                    leg.scale=1,
                    negative.args=list(col=2),
                    background=NULL,
                    scramble.cols=FALSE)
```

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).
main	Character string giving a title for the plot.
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 (by default).
show.all	Logical value specifying whether to plot everything including the window, main title, and colour ribbon.
show.window	Logical value specifying whether to plot the window.
do.plot	Logical value indicating whether to actually perform the plot.
use.marks	Logical value specifying whether to use the marks attached to the segments (use.marks=TRUE, the default) or to ignore them (use.marks=FALSE).
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.
style	Character string specifying how to represent the mark value of each segment. If style="colour" (the default) segments are coloured according to their mark value. If style="width", segments are drawn with a width proportional to their mark value. If style="none" the mark values are ignored.
col	Colour information. If style="width" or style="none", then col should be a single value, interpretable as a colour; the line segments will be plotted using this colour. If style="colour" and x has marks, then the mark values will be mapped to colours using the information in col, which should be a colour map (object of class "colourmap") or a vector of colour values.
ribbon	Logical value indicating whether to display a ribbon showing the colour map (in which mark values are associated with colours) when style="colour".
ribsep	Factor controlling the space between the colour ribbon and the image.
ribwid	Factor controlling the width of the colour ribbon.
ribn	Number of different values to display in the colour ribbon.
scale	Optional. Physical scale for representing the mark values of x as physical widths on the plot, when style="width". There is a sensible default.
adjust	Optional adjustment factor for scale.
legend	Logical value indicating whether to display a legend showing the width map (in which mark values are associated with segment widths) when style="width".
leg.side	Character string (partially matched) specifying where the legend should be plotted, when style="width".
leg.sep	Factor controlling the space between the legend and the main plot, when style="width".
leg.wid	Factor controlling the width of the legend, when style="width".
leg.args	Optional list of additional arguments passed to axis and text.default controlling the appearance of the legend, when style="width".

leg.scale	Rescaling factor for labels, when style="width". The values on the numerical scale printed beside the legend will be multiplied by this rescaling factor.
negative.args	Optional list of arguments to polygon to be used when the mark values are negative.
background	Optional. An object to be plotted underneath the main plot. See the section on *Background*.
scramble.cols	Logical value. If TRUE, the sequence of colour values will be randomly permuted. This is a useful trick when it is desired that adjacent colours in the sequence should be easily distinguishable. Applicable only when style="colour".

Details

This is the plot method for line segment pattern datasets (of class "psp", see psp.object). It plots both the observation window Window(x) and the line segments themselves.

Plotting of the window Window(x) 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.

There are three different styles of plotting which apply when the segments have marks (i.e. when marks(x) is not null):

style="colour" **(the default):** Segments are plotted with different colours depending on their mark values. The colour map, associating mark values with colours, is determined by the argument col. The colour map will be displayed as a vertical colour ribbon to the right of the plot, if ribbon=TRUE (the default).

style="width"**:** Segments are plotted with different widths depending on their mark values. The expanded segments are plotted using the base graphics function polygon. The width map, associating mark values with line widths, can be specified by giving the physical scale factor scale. There is a sensible default scale, which can be adjusted using the adjustment factor adjust. The width map will be displayed as a vertical stack of lines to the right of the plot, if legend=TRUE (the default).

style="none" **or** use.marks=FALSE**:** Mark information is ignored and the segments are plotted as thin lines using segments.

If marks(x) is a data frame, the default is to use the first column of marks(x) to determine the colours or widths. To specify another column, use the argument which.marks.

Value

If style="colour", the result is a colourmap object specifying the association between marks and colours, if any.

If style="width", the result is a numeric value giving the scaling between the mark values and the physical widths.

In all cases, the return value also has an attribute "bbox" giving a bounding box for the plot.

Scrambling the colours

When `style="colour"`, the argument `scramble.cols=TRUE` will cause the sequence of colours in the colour map to be randomly permuted. This is a useful trick when it is desired that adjacent colours in the sequence should be easily distinguishable. The return value of `plot.psp` will be the randomised colour map.

Randomisation implies that the appearance of the plot, and the return value, will be different each time the command is executed. For reproducible results, set the random generator seed using `set.seed`.

Background

The argument `background` specifies an object that will be plotted before the pattern `x` is plotted, and will therefore appear underneath it. The background can be any spatial object in the **spatstat** package family. To specify graphical arguments for the background, make it into a `layered` object using `layered` or `as.layered`, and assign graphical parameters to it using `layerplotargs`.

Alternatively `background` can be a single colour value, indicating that the plot should have this background colour.

Author(s)

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

See Also

`psp.object`, `plot`, `par`, `plot.owin`, `text.psp`, `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(X, style="width", col="grey")
marks(X) <- factor(letters[1:20])
plot(X)
plot(X, scramble.cols=TRUE)
```

---

plot.quad *Plot a Spatial Quadrature Scheme*

Description

Plot a two-dimensional spatial quadrature scheme.

Usage

```
  ## S3 method for class 'quad'
plot(x, ..., main, add=FALSE, dum=list(), tiles=FALSE)
```

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.
main	text to be displayed as a title above the plot.
add	Logical value indicating whether the graphics should be added to the current plot if there is one (add=TRUE) or whether a new plot should be initialised (add=FALSE, the default).
dum	list of extra arguments controlling the plotting of the dummy points of the quadrature scheme. See below.
tiles	Logical value indicating whether to display the tiles used to compute the quadrature weights.

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

See Also

quad.object, plot.ppp, par

Examples

```
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.quadratcount` *Plot Quadrat Counts*

Description

Given a table of quadrat counts for a spatial point pattern, plot the quadrats which were used, and display the quadrat count as text in the centre of each quadrat.

Usage

```
## S3 method for class 'quadratcount'
plot(x, ..., add = FALSE,
             entries=as.integer(t(x)),
             dx = 0, dy = 0, show.tiles = TRUE,
             textargs = list())
```

Arguments

`x`	Object of class `"quadratcount"` produced by the function `quadratcount`.
`...`	Additional arguments passed to `plot.tess` to plot the quadrats.
`add`	Logical. Whether to add the graphics to an existing plot.
`entries`	Vector of numbers to be plotted in each quadrat. The default is to plot the quadrat counts.
`dx, dy`	Horizontal and vertical displacement of text relative to centroid of quadrat.
`show.tiles`	Logical value indicating whether to plot the quadrats.
`textargs`	List containing extra arguments passed to `text.default` to control the annotation.

Details

This is the plot method for the objects of class `"quadratcount"` that are produced by the function `quadratcount`. Given a spatial point pattern, `quadratcount` divides the observation window into disjoint tiles or quadrats, counts the number of points in each quadrat, and stores the result as a contingency table which also belongs to the class `"quadratcount"`.

First the quadrats are plotted (provided `show.tiles=TRUE`, the default). This display can be controlled by passing additional arguments `...` to `plot.tess`.

Then the quadrat counts are printed using text.default. This display can be controlled using the arguments dx,dy and textargs.

If entries is given, it should be a vector of length equal to the number of quadrats (the number of tiles in the tessellation as.tess(x)) containing integer or character values to be displayed in each quadrat, in the same sequence as tiles(as.tess(x)) or tilenames(as.tess(x)) or the counts in the transposed table t(x).

Value

Null.

Author(s)

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

See Also

quadratcount, plot.tess, text.default, plot.quadrattest.

Examples

```
plot(quadratcount(swedishpines, 5))
```

---

plot.solist *Plot a List of Spatial Objects*

Description

Plots a list of two-dimensional spatial objects.

Usage

```
  ## S3 method for class 'solist'
plot(x, ..., main, arrange=TRUE,
   nrows=NULL, ncols=NULL, main.panel=NULL,
   mar.panel=c(2,1,1,2), hsep=0, vsep=0,
   panel.begin=NULL, panel.end=NULL, panel.args=NULL,
   panel.begin.args=NULL, panel.end.args=NULL, panel.vpad = 0.2,
   plotcommand="plot",
   do.plot=TRUE,
   adorn.left=NULL, adorn.right=NULL, adorn.top=NULL, adorn.bottom=NULL,
   adorn.size=0.2, adorn.args=list(),
   equal.scales=FALSE, halign=FALSE, valign=FALSE)
```

Arguments

x	An object of the class "solist", essentially a list of two-dimensional spatial datasets.
...	Arguments passed to plot when generating each plot panel.
main	Overall heading for the plot.
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.
main.panel	Optional. A character string, or a vector of character strings, or a vector of expressions, giving the headings for each plot panel.
mar.panel	Size of the margins outside each plot panel. A numeric vector of length 4 giving the bottom, left, top, and right margins in that order. (Alternatively the vector may have length 1 or 2 and will be replicated to length 4). See the section on *Spacing between plots*.
hsep, vsep	Additional horizontal and vertical separation between plot panels, expressed in the same units as mar.panel.
panel.begin, panel.end	Optional. Functions that will be executed before and after each panel is plotted. See Details.
panel.args	Optional. Function that determines different plot arguments for different panels. See Details.
panel.begin.args	Optional. List of additional arguments for panel.begin when it is a function.
panel.end.args	Optional. List of additional arguments for panel.end when it is a function.
panel.vpad	Amount of extra vertical space that should be allowed for the title of each panel, if a title will be displayed. Expressed as a fraction of the height of the panel. Applies only when equal.scales=FALSE (the default).
plotcommand	Optional. Character string containing the name of the command that should be executed to plot each panel.
do.plot	Logical value specifying whether to actually perform the plot.
adorn.left, adorn.right, adorn.top, adorn.bottom	Optional. Functions (with no arguments) that will be executed to generate additional plots at the margins (left, right, top and/or bottom, respectively) of the array of plots. Alternatively they may be objects of class "colourmap" or "symbolmap".
adorn.size	Relative width (as a fraction of the other panels' widths) of the margin plots.
adorn.args	Optional list of arguments passed to the functions adorn.left, adorn.right, adorn.top and adorn.bottom if they are functions, or arguments passed to plot.colourmap or plot.symbolmap as appropriate.
equal.scales	Logical value indicating whether the components should be plotted at (approximately) the same physical scale.

halign, valign Logical values indicating whether panels in a column should be aligned to the same x coordinate system (halign=TRUE) and whether panels in a row should be aligned to the same y coordinate system (valign=TRUE). These are applicable only if equal.scales=TRUE.

Details

This is the plot method for the class "solist".

An object of class "solist" represents a list of two-dimensional spatial datasets. This is the plot method for such objects.

In the **spatstat** package, various functions produce an object of class "solist". These objects can be plotted in a nice arrangement using plot.solist. See the Examples.

The argument panel.args determines extra graphics parameters for each panel. It should be a function that will be called as panel.args(i) where i is the panel number. Its return value should be a list of graphics parameters that can be passed to the relevant plot method. These parameters override any parameters specified in the ... arguments.

The arguments panel.begin and panel.end determine graphics that will be plotted before and after each panel is plotted. They may be objects of some class that can be plotted with the generic plot command. Alternatively they may be functions that will be called as panel.begin(i, y, main=main.panel[i]) and panel.end(i, y, add=TRUE) where i is the panel number and y = x[[i]].

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.

If equal.scales=FALSE (the default), then the plot panels will have equal height on the plot device (unless there is only one column of panels, in which case they will have equal width on the plot device). This means that the objects are plotted at different physical scales, by default.

If equal.scales=TRUE, then the dimensions of the plot panels on the plot device will be proportional to the spatial dimensions of the corresponding components of x. This means that the objects will be plotted at *approximately* equal physical scales. If these objects have very different spatial sizes, the plot command could fail (when it tries to plot the smaller objects at a tiny scale), with an error message that the figure margins are too large.

The objects will be plotted at *exactly* equal physical scales, and *exactly* aligned on the device, under the following conditions:

- every component of x is a spatial object whose position can be shifted by shift;
- panel.begin and panel.end are either NULL or they are spatial objects whose position can be shifted by shift;
- adorn.left, adorn.right, adorn.top and adorn.bottom are all NULL.

Another special case is when every component of x is an object of class "fv" representing a function. If equal.scales=TRUE then all these functions will be plotted with the same axis scales (i.e. with the same xlim and the same ylim).

Value

Null.

Spacing between plots

The spacing between individual plots is controlled by the parameters mar.panel, hsep and vsep.

If equal.scales=FALSE, the plot panels are logically separate plots. The margins for each panel are determined by the argument mar.panel which becomes the graphics parameter mar described in the help file for par. One unit of mar corresponds to one line of text in the margin. If hsep or vsep are present, mar.panel is augmented by c(vsep, hsep, vsep, hsep)/2.

If equal.scales=TRUE, all the plot panels are drawn in the same coordinate system which represents a physical scale. The unit of measurement for mar.panel[1,3] is one-sixth of the greatest height of any object plotted in the same row of panels, and the unit for mar.panel[2,4] is one-sixth of the greatest width of any object plotted in the same column of panels. If hsep or vsep are present, they are interpreted in the same units as mar.panel[2] and mar.panel[1] respectively.

Error messages

If the error message 'Figure margins too large' occurs, this generally means that one of the objects had a much smaller physical scale than the others. Ensure that equal.scales=FALSE and increase the values of mar.panel.

Author(s)

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

See Also

plot.anylist, contour.listof, image.listof, density.splitppp

Examples

```
D <- solapply(split(amacrine), distmap)
plot(D)
plot(D, main="", equal.ribbon=TRUE,
     panel.end=function(i,y,...){contour(y, ...)})
```

---

plot.splitppp *Plot a List of Point Patterns*

Description

Plots a list of point patterns.

Usage

```
  ## S3 method for class 'splitppp'
plot(x, ..., main)
```

Arguments

x	A named list of point patterns, typically obtained from split.ppp.
...	Arguments passed to plot.listof which control the layout of the plot panels, their appearance, and the plot behaviour in individual plot panels.
main	Optional main title for the plot.

Details

This is the plot method for the class "splitppp". It is typically used to plot the result of the function split.ppp.

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.

Plotting is performed by plot.listof.

Value

Null.

Error messages

If the error message 'Figure margins too large' occurs, ensure that equal.scales=FALSE and increase the values of mar.panel.

Author(s)

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

See Also

plot.listof for arguments controlling the plot.

split.ppp, plot.ppp, ppp.object.

Examples

```
# Multitype point pattern
 plot(split(amacrine))
 plot(split(amacrine), main="",
     panel.begin=function(i, y, ...) { plot(distmap(y), ribbon=FALSE, ...) })
```

plot.symbolmap *Plot a Graphics Symbol Map*

Description

Plot a representation of a graphics symbol map, similar to a plot legend.

Usage

```
## S3 method for class 'symbolmap'
plot(x, ..., main, xlim = NULL, ylim = NULL,
                        vertical = FALSE,
                        side = c("bottom", "left", "top", "right"),
                        annotate = TRUE, labelmap = NULL,
                        add = FALSE, do.plot=TRUE,
                        nsymbols = NULL, warn = TRUE,
                        colour.only=FALSE,
                        representatives=NULL)
```

Arguments

x	Graphics symbol map (object of class "symbolmap").
...	Additional graphics arguments passed to points, symbols or axis.
main	Main title for the plot. A character string.
xlim, ylim	Coordinate limits for the plot. Numeric vectors of length 2.
vertical	Logical. Whether to plot the symbol map in a vertical orientation.
side	Character string specifying the position of the text that annotates the symbols. Alternatively one of the integers 1 to 4.
annotate	Logical. Whether to annotate the symbols with labels.
labelmap	Transformation of the labels. A function or a scale factor which will be applied to the data values corresponding to the plotted symbols.
add	Logical value indicating whether to add the plot to the current plot (add=TRUE) or to initialise a new plot.
do.plot	Logical value indicating whether to actually perform the plot.
nsymbols	Optional. The maximum number of symbols that should be displayed. Ignored if representatives are given.
warn	Logical value specifying whether to issue a warning when the plotted symbol map does not represent every possible discrete value.
colour.only	Logical value. If TRUE, the colour map information will be extracted from the symbol map, and only this colour map will be plotted. If FALSE (the default) the entire symbol map is plotted, including information about symbol shape and size as well as colour.
representatives	Optional. Vector containing the values of the input data which should be shown on the plot.

Details

A graphics symbol map (object of class "symbolmap") is an association between data values and graphical symbols.

This command plots the graphics symbol map itself, in the style of a plot legend.

For a map of continuous values (a symbol map which represents a range of numerical values) the plot will select about nsymbols different values within this range, and plot their graphical representations.

For a map of discrete inputs (a symbol map which represents a finite set of elements, such as categorical values) the plot will try to display the graphical representation of every possible input, up to a maximum of nsymbols items. If there are more than nsymbols possible inputs, a warning will be issued (if warn=TRUE, the default).

Value

(Invisibly) The symbolmap x with an attribute "bbox" which is a rectangular window (object of class "owin") giving the region in coordinate space where the symbol map was plotted (or would have been plotted).

Nonlinear transformation

For a map of continuous values, if representatives is not specified, then the default is to choose approximately nsymbols values equally spaced within the numerical range.

However, if the symbol map includes a nonlinear transformation (arguments compress and decompress were passed to symbolmap) or if one of its graphical parameters is a colour map which includes a nonlinear transformation (arguments compress and decompress were passed to colourmap) then the representative values will be equally spaced on the compressed scale.

Author(s)

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

See Also

symbolmap to create a symbol map.

invoke.symbolmap to apply the symbol map to some data and plot the resulting symbols.

Examples

```
g <- symbolmap(inputs=letters[1:10], pch=11:20)
plot(g)

## squares and circles with area proportional to |x|
## For 'squares', size is side length; size = sqrt(area)
## For 'circles', size is diameter;    size = sqrt(area * 4/pi)
g2 <- symbolmap(range=c(-1,1),
                shape=function(x) ifelse(x > 0, "circles", "squares"),
                size=function(x) ifelse(x > 0,
```

```
                                        sqrt(abs(x)*4/pi),
                                        sqrt(abs(x))),
                bg = function(x) ifelse(abs(x) < 1, "red", "black"))
plot(g2, vertical=TRUE, side="left", col.axis="blue", cex.axis=2)
plot(g2, representatives=c(-1,0,1))

## logarithmic display scale
gl <- symbolmap(range=c(1,1000), pch=21,
                bg=function(x) ifelse(x < 50, "red", "blue"),
                cex=function(x) ifelse(x < 100, 1, 2),
                compress=log10)
gl(10)
plot(gl, nsymbols=4)

## logarithmic symbol map
gll <- symbolmap(range=c(1,1000),
                 transform=log10, size=function(x) { 1 + x },
                 compress=log10)
gll(10)
plot(gll, nsymbols=4)
```

`plot.tess` *Plot a Tessellation*

Description

Plots a tessellation, with optional labels for the tiles, and optional filled colour in each tile.

Usage

```
  ## S3 method for class 'tess'
plot(x, ..., main, add=FALSE,
                    show.all=!add,
                    border=NULL,
                    do.plot=TRUE,
                    do.labels=!missing(labels),
                    labels=tilenames(x), labelargs=list(),
                    do.col=!missing(values),
                    values=marks(x),
                    multiplot=TRUE,
                    col=NULL, ribargs=list())
```

Arguments

`x`	Tessellation (object of class `"tess"`) to be plotted.
`...`	Arguments controlling the appearance of the plot.
`main`	Heading for the plot. A character string.
`add`	Logical. Determines whether the tessellation plot is added to the existing plot.

show.all	Logical value indicating whether to plot everything including the main title and the observation window of x.
border	Colour of the tile boundaries. A character string or other value specifying a single colour. Ignored for pixel tessellations.
do.plot	Logical value indicating whether to actually perform the plot.
do.labels	Logical value indicating whether to show a text label for each tile of the tessellation. The default is TRUE if labels are given, and FALSE otherwise.
labels	Character vector of labels for the tiles.
labelargs	List of arguments passed to text.default to control display of the text labels.
do.col	Logical value indicating whether tiles should be filled with colour (for tessellations where the tiles are rectangles or polygons). The default is TRUE if values are given, and FALSE otherwise.
values	A vector of numerical values (or a factor, or vector of character strings) that will be associated with each tile of the tessellation and which determine the colour of the tile. The default is the marks of x. If the tessellation is not marked, or if the argument values=NULL is given, the default is a factor giving the tile identifier.
multiplot	Logical value giving permission to display multiple plot panels. This applies when do.col=TRUE and ncol(values) > 1.
col	A vector of colours for each of the values, or a colourmap that maps these values to colours.
ribargs	List of additional arguments to control the plot of the colour map, if do.col=TRUE. See explanation in plot.im.

Details

This is a method for the generic plot function for the class "tess" of tessellations (see tess).

The window of the tessellation is plotted, and then the tiles of the tessellation are plotted in their correct positions in the window.

Rectangular or polygonal tiles are plotted individually using plot.owin, while a tessellation represented by a pixel image is plotted using plot.im. The arguments ... control the appearance of the plot, and are passed to segments, plot.owin or plot.im as appropriate.

If do.col=TRUE, then the tiles of the tessellation are filled with colours determined by the argument values. By default, these values are the marks associated with each of the tiles. If there is more than one column of marks or values, then the default behaviour (if multiplot=TRUE) is to display several plot panels, one for each column of mark values. Then the arguments ... are passed to plot.solist to determine the arrangement of the panels.

If do.labels=TRUE, a text label is plotted in the middle of each tile. The text labels are determined by the argument labels, and default to the names of the tiles given by tilenames(x).

Value

(Invisible) window of class "owin" specifying a bounding box for the plot, or an object of class "colourmap" specifying the colour map. (In the latter case, the bounding box information is available as an attribute, and can be extracted using as.owin.)

Author(s)

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

See Also

tess

Examples

```
Rect <- tess(xgrid=0:4,ygrid=0:4)
Diri <- dirichlet(runifrect(7))
plot(Diri)
plot(Rect, border="blue", lwd=2, lty=2)
plot(Rect, do.col=TRUE, border="white")
plot(Rect, do.col=TRUE, values=runif(16), border="white")
B <- Rect[c(1, 2, 5, 7, 9)]
plot(B, hatch=TRUE)
plot(Diri, do.col=TRUE)
plot(Diri, do.col=TRUE, do.labels=TRUE, labelargs=list(col="white"),
           ribbon=FALSE)
v <- as.im(function(x,y){factor(round(5 * (x^2 + y^2)))}, W=owin())
levels(v) <- letters[seq(length(levels(v)))]
Img <- tess(image=v)
plot(Img)
plot(Img, col=rainbow(11), ribargs=list(las=1))
a <- tile.areas(Diri)
marks(Diri) <- data.frame(area=a, random=runif(7, max=max(a)))
plot(Diri, do.col=TRUE, equal.ribbon=TRUE)
```

---

plot.textstring *Plot a Text String*

Description

Plots an object of class "textstring".

Usage

```
## S3 method for class 'textstring'
plot(x, ..., do.plot = TRUE)
```

Arguments

x	Object of class "textstring" to be plotted. This object is created by the command textstring.
...	Additional graphics arguments passed to text to control the plotting of text.
do.plot	Logical value indicating whether to actually plot the text.

Details

The argument x should be an object of class "textstring" created by the command textstring.

This function displays the text using text.

Value

A window (class "owin") enclosing the plotted graphics.

Author(s)

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

See Also

onearrow, yardstick

Examples

```
W <- Window(humberside)
te <- textstring(centroid.owin(W), txt="Humberside", cex=2.5)
te
plot(layered(W, te), main="")
```

---

plot.texturemap *Plot a Texture Map*

Description

Plot a representation of a texture map, similar to a plot legend.

Usage

```
 ## S3 method for class 'texturemap'
plot(x, ..., main, xlim = NULL, ylim = NULL,
                           vertical = FALSE, axis = TRUE,
                           side = if(vertical) "right" else "bottom",
                           labelmap = NULL, gap = 0.25,
                           spacing = NULL, add = FALSE)
```

Arguments

x	Texture map object (class "texturemap").
...	Additional graphics arguments passed to add.texture or axis.
main	Main title for plot.
xlim, ylim	Optional vectors of length 2 giving the x and y limits of the plot.

vertical	Logical value indicating whether to arrange the texture boxes in a vertical column (vertical=TRUE or a horizontal row (vertical=FALSE, the default).
axis	Logical value indicating whether to plot axis-style labels next to the texture boxes.
side	One of the character strings "bottom", "left", "top" or "right", or one of the integers from 1 to 4, specifying the position of the axis-style labels, if axis=TRUE.
labelmap	Optional. A function which will be applied to the data values (the inputs of the texture map) before they are displayed on the plot.
gap	Separation between texture boxes, as a fraction of the width or height of a box.
spacing	Argument passed to add.texture controlling the density of lines in a texture. Expressed in spatial coordinate units.
add	Logical value indicating whether to add the graphics to an existing plot (add=TRUE) or to initialise a new plot (add=FALSE, the default).

Details

A texture map is an association between data values and graphical textures. An object of class "texturemap" represents a texture map. Such objects are returned from the plotting function textureplot, and can be created directly by the function texturemap.

This function plot.texturemap is a method for the generic plot for the class "texturemap". It displays a sample of each of the textures in the texture map, in a separate box, annotated by the data value which is mapped to that texture.

The arrangement and position of the boxes is controlled by the arguments vertical, xlim, ylim and gap.

Value

Null.

Author(s)

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

Rolf Turner <rolfturner@posteo.net>

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

See Also

texturemap, textureplot, add.texture.

Examples

```
tm <- texturemap(c("First", "Second", "Third"), 2:4, col=2:4)
plot(tm, vertical=FALSE)
## abbreviate the labels
plot(tm, labelmap=function(x) substr(x, 1, 2))
```

plot.yardstick *Plot a Yardstick or Scale Bar*

Description

Plots an object of class "yardstick".

Usage

```
 ## S3 method for class 'yardstick'
plot(x, ...,
        style=c("arrows", "zebra"),
        angle = 20, frac = 1/8,
        split = FALSE, shrink = 1/4,
        pos = NULL,
        txt.args=list(),
        txt.shift=c(0,0),
        zebra.step=NULL, zebra.width=NULL,
        zebra.col="black", zebra.scale=1,
        zebra.args=list(), zebra.shift=c(0,0),
        do.plot = TRUE, do.txt=TRUE)
```

Arguments

x	Object of class "yardstick" to be plotted. This object is created by the command yardstick.
...	Additional graphics arguments passed to segments to control the appearance of the line.
style	Character string (partially matched) specifying the style of plot. See Details.
angle	Angle between the arrows and the line segment, in degrees. Applies when style="arrows".
frac	Length of arrow as a fraction of total length of the line segment. Applies when style="arrows".
split	Logical. If TRUE, then the line will be broken in the middle, and the text will be placed in this gap. If FALSE, the line will be unbroken, and the text will be placed beside the line. Applies when style="arrows".
shrink	Fraction of total length to be removed from the middle of the line segment, if split=TRUE. Applies when style="arrows".
pos	Integer (passed to text) determining the position of the annotation text relative to the line segment, if split=FALSE. Values of 1, 2, 3 and 4 indicate positions below, to the left of, above and to the right of the line, respectively.
txt.args	Optional list of additional arguments passed to text controlling the appearance of the text. Examples include adj, srt, col, cex, font.
txt.shift	Optional numeric vector of length 2 specifying displacement of the text position relative to the centre of the yardstick.

zebra.step	Length of each bar in the zebra pattern. Applies when style="zebra".
zebra.width	Width of each bar in the zebra pattern. Applies when style="zebra".
zebra.col	Colour of each bar in the zebra pattern. Applies when style="zebra".
zebra.scale	Scale value for numerical labels in the zebra pattern. Physical lengths will be divided by zebra.scale to determine the corresponding numerical labels.
zebra.args	Optional list of additional arguments passed to text controlling the appearance of the text annotation for each bar in the zebra pattern. Examples include adj, srt, col, cex, font.
zebra.shift	Optional numeric vector of length 2 specifying displacement of the text annotation for each bar in the zebra pattern.
do.plot	Logical values specifying whether to actually perform the plot.
do.txt	Logical value specifying whether to draw text annotation.

Details

A yardstick or scale bar is a line segment, drawn on any spatial graphics display, indicating the scale of the plot.

- If style="arrows", the line segment is drawn as a pair of arrows pointing from the middle of the line to the ends of the line. This style is often used in architectural drawings. If angle=0, the arrow heads are replaced by parallel bars marking the two ends of the line.
- If style="zebra", the line segment is divided into block of length zebra.step and width zebra.width units. Blocks are drawn alternately as filled rectangles and outlined rectangles, so that the result resembles a zebra crossing. This style is often used in maps and charts. There are sensible defaults for zebra.step and zebra.width.

The argument x should be an object of class "yardstick" created by the command yardstick.

Value

A window (class "owin") enclosing the plotted graphics.

Author(s)

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

See Also

yardstick

Examples

```
plot(owin(), main="Yardsticks")
ys <- yardstick(as.psp(list(xmid=0.5, ymid=0.1, length=0.4, angle=0),
                       window=owin(c(0.2, 0.8), c(0, 0.2))),
                txt="1 km")
plot(ys)
```

```
ys <- shift(ys, c(0, 0.3))
plot(ys, angle=90, frac=0.08)
ys <- shift(ys, c(0, 0.3))
plot(ys, split=TRUE)

yt <- shift(ys, c(0, 0.2))
plot(yt, style="z", pos=3, zebra.step=0.1, txt.args=list(offset=0.1))
```

`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 `eps = len/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`. The result also has an attribute called `"map"` which maps the points to their parent line segments.

Author(s)

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

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="+")
```

---

polarbody *Polar Body and Voronoi Flower*

Description

Compute the polar body or the Voronoi flower of a two-dimensional domain.

Usage

```
polarbody(W, origin = c(0, 0), npoly = 512)

voronoiFlower(W, origin = c(0, 0), npoly = 512)
```

Arguments

W	Window (object of class "owin") representing the domain.
origin	The location of the origin for the support function. Either a numeric vector of length 2 giving the location, or a point pattern containing only one point, or a list with two entries named x and y, or one of the character strings "centroid", "midpoint", "left", "right", "top", "bottom", "topleft", "bottomleft", "topright" or "bottomright" (partially matched).
npoly	Number of edges in the approximating polygon.

Details

The polar body and Voronoi flower of a set W are geometrical constructions based on the support function $h_W(\theta)$ of W.

The polar body of a two-dimensional domain W is the set whose boundary has polar coordinates $(\theta, 1/h_W(\theta))$ for all directions θ.

The Voronoi flower is the set whose boundary has polar coordinates $(\theta, 2h_W(\theta))$.

Value

A polygonal window (object of class "owin") representing the polar body or the Voronoi flower.

Author(s)

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

See Also

SupportFun

Examples

```
plot(polarbody(letterR, "midpoint"))

plot(voronoiFlower(letterR, "midpoint"))
```

---

polartess *Tessellation Using Polar Coordinates*

Description

Create a tessellation with tiles defined by polar coordinates (radius and angle).

Usage

```
polartess(W, ..., nradial = NULL, nangular = NULL,
          radii = NULL, angles = NULL,
          origin = NULL, sep = "x")
```

Arguments

W	A window (object of class "owin") or anything that can be coerced to a window using as.owin, such as a point pattern.
...	Ignored.
nradial	Number of *tiles* in the radial direction. A single integer. Ignored if radii is given.
nangular	Number of *tiles* in the angular coordinate. A single integer. Ignored if angles is given.
radii	The numeric values of the radii, defining the tiles in the radial direction. A numeric vector, of length at least 2, containing nonnegative numbers in increasing order. The value Inf is permitted.
angles	The numeric values of the angles defining the tiles in the angular coordinate. A numeric vector, of length at least 2, in increasing order, containing angles in radians.
origin	Location to be used as the origin of the polar coordinates. Either a numeric vector of length 2 giving the spatial location of the origin, or one of the strings "centroid", "midpoint", "left", "right", "top", "bottom", "topleft", "bottomleft", "topright" or "bottomright" indicating the location in the window.
sep	Argument passed to intersect.tess specifying the character string to be used as a separator when forming the names of the tiles.

Details

A tessellation will be formed from tiles defined by intervals in the polar coordinates r (radial distance from the origin) or θ (angle from the horizontal axis) or both. These tiles look like the cells on a dartboard.

If the argument `radii` is given, tiles will be demarcated by circles centred at the origin, with the specified radii. If `radii` is absent but `nradial` is given, then `radii` will default to a sequence of `nradial+1` radii equally spaced from zero to the maximum possible radius. If neither `radii` nor `nradial` are given, the tessellation will not include circular arc boundaries.

If the argument `angles` is given, tiles will be demarcated by lines emanating from the origin at the specified angles. The angular values can be any real numbers; they will be interpreted as angles in radians modulo `2*pi`, but they must be an increasing sequence of numbers. If `angles` is absent but `nangular` is given, then `angles` will default to a sequence of `nangular+1` angles equally spaced from 0 to `2*pi`. If neither `angles` nor `nangular` are given, the tessellation will not include linear boundaries.

Value

A tessellation (object of class `"tess"`).

Author(s)

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

See Also

`intersect.tess`

To construct other kinds of tessellations, see `tess`, `quadrats`, `hextess`, `venn.tess`, `dirichlet`, `delaunay`, `quantess`, `bufftess` and `rpoislinetess`.

Examples

```
Y <- c(2.8, 1.5)
plot(polartess(letterR, nangular=6, radii=(0:4)/2, origin=Y),
     do.col=TRUE)
```

---

pp3 *Three Dimensional Point Pattern*

Description

Create a three-dimensional point pattern

Usage

```
pp3(x, y, z, ..., marks=NULL)
```

Arguments

x, y, z	Numeric vectors of equal length, containing Cartesian coordinates of points in three-dimensional space.
...	Arguments passed to as.box3 to determine the three-dimensional box in which the points have been observed.
marks	Optional. Vector, data frame, or hyperframe of mark values associated with the points.

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

See Also

box3, print.pp3, ppx

Examples

```
X <- pp3(runif(10), runif(10), runif(10),
         box3(c(0,1)),
         marks=rnorm(10))
X
```

---

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, checkdup=check, drop=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 value indicating whether to check that all the (x, y) points lie inside the specified window. Do not set this to FALSE unless you are absolutely sure that this check is unnecessary. See Warnings below.
checkdup	Logical value indicating whether to check for duplicated coordinates. See Warnings below.
drop	Logical flag indicating whether to simplify data frames of marks. See Details.

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. Note that x and y are permitted to have length zero, corresponding to an empty point pattern; this is the default if these arguments are missing.

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, yrange 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 ith 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" specifying the window. A window object can be created by owin from raw coordinate data. Special shapes of windows can be created by the functions square, hexagon, regularpolygon, disc and ellipse. See the Examples.

The arguments xrange, yrange 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 ith row of the data frame is interpreted as containing the mark values for the ith point in the point pattern. The columns of the data frame correspond to different mark variables (e.g. tree species and tree diameter).

If drop=TRUE (the default), then a data frame with only one column will be converted to a vector, and a data frame with no columns will be converted to NULL.

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).

Invalid coordinate values

The coordinate vectors x and y must contain only finite numerical values. If the coordinates include any of the values NA, NaN, Inf or -Inf, these will be removed.

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 larger window). However, rejected points in a point pattern will be ignored by all other functions except plot.ppp.

To delete the rejected points altogether from the point pattern X, use as.ppp(X) or attr(X, "rejects") <- NULL. To extract the rejected points, use R <- attr(X, "rejects"). Use as.layered(X) to extract the rejected and accepted points as separate layers in a layered object.

To automatically merge the rejected points into the dataset using a larger window, use `do.call(superimpose, unname(as.layered(X)))`. The resulting window may be unnecessarily large, since it contains the convex hull of the rejected points.

Warnings

The code will check for problems with the data, and issue a warning if any problems are found. The checks and warnings can be switched off, for efficiency's sake, but this should only be done if you are confident that the data do not have these problems.

Setting `check=FALSE` will disable all the checking procedures: the check for points outside the window, and the check for duplicated points. This is extremely dangerous, because points lying outside the window will break many of the procedures in **spatstat**, causing crashes and strange errors. Set `check=FALSE` only if you are absolutely sure that there are no points outside the window.

If duplicated points are found, a warning is issued, but no action is taken. Duplicated points are not illegal, but may cause unexpected problems later. Setting `checkdup=FALSE` will disable the check for duplicated points. Do this only if you already know the answer.

Methodology and software for spatial point patterns often assume that all points are distinct so that there are no duplicated points. If duplicated points are present, the consequence could be an incorrect result or a software crash. To the best of our knowledge, all **spatstat** code handles duplicated points correctly. However, if duplicated points are present, we advise using `unique.ppp` or `multiplicity.ppp` to eliminate duplicated points and re-analyse the data.

Author(s)

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

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"))

# 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)
```

```
# polygonal window
X <- ppp(x, y, poly=list(x=c(0,10,0), y=c(0,0,10)))

# circular window of radius 2
X <- ppp(x, y, window=disc(2))

# copy the window from another pattern
X <- ppp(x, y, window=Window(cells))
```

---

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.

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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
if(human <- interactive()) plot(X)
mar <- sample(1:3, 100, replace=TRUE)
mm <- ppp(x, y, c(0,1), c(0,1), marks=mar)
if(human) plot(mm)
# points with mark equal to 2
ss <- mm[ mm$marks == 2 , ]
if(human) plot(ss)
# left half of pattern 'mm'
lu <- owin(c(0,0.5),c(0,1))
mmleft <- mm[ , lu]
if(human) plot(mmleft)
if(FALSE) {
# 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)
}
```

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, auction = 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.
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 FALSE, R code is used which allows for higher precision, but is much slower.
auction	Logical. By default a version of Bertsekas' auction algorithm is used to compute an optimal point matching if type is either "spa" or "ace". If auction is FALSE (or type is "mat") a specialized primal-dual algorithm is used instead. This was the standard in earlier versions of **spatstat**, but is several orders of magnitudes slower.
precision	Index controlling accuracy of algorithm. The q-th powers of interpoint distances will be rounded to the nearest multiple of 10^(-precision). There is a sensible default which depends on ccode.
approximation	If q = 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 plot the progress of the primal-dual algorithm. If TRUE, slow primal-dual R code is used, regardless of the arguments ccode and auction.
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=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 central problem to be solved is the assignment problem (for types `"spa"` and `"ace"`) or the more general transport problem (for type `"mat"`). Both are well-known problems in discrete optimization, see e.g. Luenberger (2003).

For the assignment problem `pppdist` uses by default the forward/backward version of Bertsekas' auction algorithm with automated epsilon scaling; see Bertsekas (1992). The implemented version gives good overall performance and can handle point patterns with several thousand points.

For the transport problem a specialized primal-dual algorithm is employed; see Luenberger (2003), Section 5.9. 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 a broader selection of optimal transport algorithms that are not restricted to spatial point patterns and allow for additional fine tuning, we recommend the R package **transport**.

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% 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 = 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@mathematik.uni-goettingen.de>, URL http://dominic.schuhmache

References

Bertsekas, D.P. (1992). Auction algorithms for network flow problems: a tutorial introduction. Computational Optimization and Applications 1, 7-66.

Luenberger, D.G. (2003). *Linear and nonlinear programming.* Second edition. Kluwer.

Schuhmacher, D. (2014). *transport: optimal transport in various forms.* R package version 0.6-2 (or later)

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, plot.pppmatching

Examples

```
# equal cardinalities
set.seed(140627)
X <- runifrect(500)
Y <- runifrect(500)
m <- pppdist(X, Y)
m
if(interactive()) {
plot(m)}

# differing cardinalities
X <- runifrect(14)
Y <- runifrect(10)
m1 <- pppdist(X, Y, type="spa")
m2 <- pppdist(X, Y, type="ace")
m3 <- pppdist(X, Y, type="mat", auction=FALSE)
summary(m1)
summary(m2)
summary(m3)
if(interactive()) {
```

```
m1$matrix
m2$matrix
m3$matrix}

# q = Inf
X <- runifrect(10)
Y <- runifrect(10)
mx1 <- pppdist(X, Y, q=Inf, matching=FALSE)
mx2 <- pppdist(X, Y, q=Inf, matching=FALSE, ccode=FALSE, approximation=50)
mx3 <- pppdist(X, Y, q=Inf, matching=FALSE, approximation=Inf)
all.equal(mx1,mx2,mx3)
# sometimes TRUE
all.equal(mx2,mx3)
# very often TRUE
```

---

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 npoints(X) by npoints(Y) 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 as.matrix.
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@mathematik.uni-goettingen.de>, URL http://dominic.schuhmache

See Also

pppmatching.object matchingdist

Examples

```
# a random unweighted complete matching
X <- runifrect(10)
Y <- runifrect(10)
am <- r2dtable(1, rep(1,10), rep(1,10))[[1]]
      # generates a random permutation matrix
m <- pppmatching(X, Y, am)
summary(m)
m$matrix
plot(m)

# a random weighted complete matching
X <- runifrect(7)
Y <- runifrect(7)
am <- r2dtable(1, rep(10,7), rep(10,7))[[1]]/10
      # generates a random doubly stochastic matrix
m2 <- pppmatching(X, Y, am)
summary(m2)
m2$matrix
plot(m2)
m3 <- pppmatching(X, Y, am, "ace")
m4 <- pppmatching(X, Y, am, "mat")
```

---

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 pppmatching, 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@mathematik.uni-goettingen.de>, URL http://dominic.schuhmache

See Also

matchingdist, pppmatching, plot.pppmatching

Examples

```
# a random complete unweighted matching
X <- runifrect(10)
Y <- runifrect(10)
am <- r2dtable(1, rep(1,10), rep(1,10))[[1]]
      # generates a random permutation matrix
m <- pppmatching(X, Y, am)
summary(m)
m$matrix
if(interactive()) {
  plot(m)
}

# an optimal complete unweighted matching
m2 <- pppdist(X,Y)
summary(m2)
```

```
m2$matrix
if(interactive()) {
  plot(m2)
}
```

---

ppx *Multidimensional Space-Time Point Pattern*

Description

Creates a multidimensional space-time point pattern with any kind of coordinates and marks.

Usage

```
ppx(data, domain=NULL, coord.type=NULL, simplify=FALSE)
```

Arguments

`data`	The coordinates and marks of the points. A `data.frame` or `hyperframe`.
`domain`	Optional. The space-time domain containing the points. An object in some appropriate format, or `NULL`.
`coord.type`	Character vector specifying how each column of `data` 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"`.
`simplify`	Logical value indicating whether to simplify the result in special cases. If `simplify=TRUE`, a two-dimensional point pattern will be returned as an object of class `"ppp"`, and a three-dimensional point pattern will be returned as an object of class `"pp3"`. If `simplify=FALSE` (the default) then the result is always an object of class `"ppx"`.

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

Usually an object of class "ppx". If simplify=TRUE the result may be an object of class "ppp" or "pp3".

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

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

#' one-dimensional points
#' with marks which are two-dimensional point patterns
val <- sample(10:20, 4)
E <- lapply(val, runifrect)
E
hf <- hyperframe(num=val, e=as.listof(E))
Z <- ppx(data=hf, domain=c(10,20))
Z
```

---

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

print, im.object, summary.im

Examples

```
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, ..., prefix="window: ")
```

Arguments

x	Window (object of class "owin").
...	Ignored.
prefix	Character string to be printed at the start of the output.

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

See Also

print, print.ppp, summary.owin

Examples

```
owin()  # the unit square

W <- Window(demopat)
W                    # just says it is polygonal
as.mask(W)           # just says it is a binary image
```

---

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

See Also

print, print.owin, summary.ppp

Examples

```
cells      # plain vanilla point pattern

lansing    # multitype point pattern

longleaf    # numeric marks

demopat     # weird polygonal window
```

`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@curtin.edu.au>`

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

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

See Also

quadscheme, quad.object, plot.quad, summary.quad

Examples

```
Q <- quadscheme(cells)
Q
```

progressreport *Print Progress Reports*

Description

Prints Progress Reports during a loop or iterative calculation.

Usage

```
progressreport(i, n,
               every = min(100,max(1, ceiling(n/100))),
               tick = 1,
               nperline = NULL,
               charsperline = getOption("width"),
               style = spatstat.options("progress"),
               showtime = NULL,
               state=NULL,
               formula = (time ~ i),
               savehistory=FALSE)
```

Arguments

i	Integer. The current iteration number (from 1 to n).
n	Integer. The (maximum) number of iterations to be computed.
every	Optional integer. Iteration number will be printed when i is a multiple of every.
tick	Optional integer. A tick mark or dot will be printed when i is a multiple of tick.
nperline	Optional integer. Number of iterations per line of output.
charsperline	Optional integer. The number of characters in a line of output.
style	Character string determining the style of display. Options are "tty" (the default), "tk" and "txtbar". See Details.
showtime	Optional. Logical value indicating whether to print the estimated time remaining. Applies only when style="tty".
state	Optional. A list containing the internal data.
formula	Optional. A model formula expressing the expected relationship between the iteration number i and the clock time time. Used for predicting the time remaining.
savehistory	Optional. Logical value indicating whether to save the elapsed times at which progressreport was called.

Details

This is a convenient function for reporting progress during an iterative sequence of calculations or a suite of simulations.

- If style="tk" then tcltk::tkProgressBar is used to pop-up a new graphics window showing a progress bar. This requires the package **tcltk**. As i increases from 1 to n, the bar will lengthen. The arguments every, tick, nperline, showtime are ignored.
- If style="txtbar" then txtProgressBar is used to represent progress as a bar made of text characters in the R interpreter window. As i increases from 1 to n, the bar will lengthen. The arguments every, tick, nperline, showtime are ignored.
- If style="tty" (the default), then progress reports are printed to the console. This only seems to work well under Linux. As i increases from 1 to n, the output will be a sequence of dots (one dot for every tick iterations), iteration numbers (printed when iteration number is a multiple of every or is less than 4), and optionally the estimated time remaining and the estimated completion time.

  The estimated time remaining will be printed only if style="tty", and the argument state is given, and either showtime=TRUE, or showtime=NULL and the iterations are slow (defined as: the estimated time remaining is longer than 3 minutes, or the average time per iteration is longer than 20 seconds).

  The estimated completion time will be printed only if the estimated time remaining is printed and the remaining time is longer than 10 minutes.

  By default, the estimated time remaining is calculated by assuming that each iteration takes the same amount of time, and extrapolating. Alternatively, if the argument formula is given, then it should be a model formula, stating the expected relationship between the iteration number i and the clock time time. This model will be fitted to the history of clock times recorded so far, and used to predict the time remaining. (The default formula states that clock time is a linear function of the iteration number, which is equivalent to assuming that each iteration takes the same amount of time.)

It is optional, but strongly advisable, to use the argument state to store and update the internal data for the progress reports (such as the cumulative time taken for computation) as shown in the last example below. This avoids conflicts with other programs that might be calling progressreport at the same time.

Value

If state was NULL, the result is NULL. Otherwise the result is the updated value of state.

Author(s)

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

Examples

```
for(i in 1:40) {
   #
   # code that does something...
   #
   progressreport(i, 40)
}

# saving internal state: *recommended*
```

```
sta <- list()
for(i in 1:20) {
   # some code ...
   sta <- progressreport(i, 20, state=sta)
}

#' use text progress bar
sta <- list()
for(i in 1:10) {
   # some code ...
   sta <- progressreport(i, 10, state=sta, style="txtbar")
}
```

`project2segment` *Move Point To Nearest Line*

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 mapXY[2] = 5 and 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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

nearestsegment for a faster way to determine which segment is closest to each point.

Examples

```
X <- rsyst(square(1), nx=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")
```

---

project2set *Find Nearest Point in a Region*

Description

For each data point in a point pattern X, find the nearest location in a given spatial region W.

Usage

```
project2set(X, W, ..., polygonal=is.polygonal(W))
```

Arguments

X	Point pattern (object of class "ppp").
W	Window (object of class "owin") or something acceptable to as.owin.
...	Arguments passed to as.mask controlling the pixel resolution, if polygonal=FALSE.
polygonal	Logical value specifying whether W should be treated as a polygonal window (polygonal=TRUE) or a binary mask window (polygonal=FALSE), and converted to the desired kind of window if necessary.

Details

This function alters the point pattern X so that every point lies inside the window W.

- If polygonal=FALSE the window W is first discretised as a binary mask using as.mask with pixel resolution determined by the arguments .... Then for each data point X[i] in the point pattern X, the algorithm finds the nearest pixel centre inside W, and moves X[i] to this location.
- If polygonal=TRUE, the window W is first converted to a polygonal window using as.polygonal. Then for each data point X[i] in the point pattern X, the algorithm finds the nearest location along the boundary of W, moves X[i] to that location, and then perturbs X[i] slightly so that it lies a short distance inside the window W.

The result is a point pattern Y containing these nearest points, that is, Y[i] is the nearest point in W to the point X[i].

Value

A point pattern (object of class "ppp") with the same number of points as X in the window W.

Author(s)

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

See Also

project2segment, nncross

Examples

```
## discrete case
  He <- heather$fine[owin(c(2.8, 7.4), c(4.0, 7.8))]
  plot(He, main="project2set")
  W <- erosion(complement.owin(He), 0.2)
if(require(spatstat.random)) {
  X <- runifpoint(4, W)
} else {
  X <- ppp(c(6.1, 4.3, 5.7, 4.7), c(5.0, 6.6, 7.5, 4.9), window=W)
}
  points(X, col="red")
  Y <- project2set(X, He)
  points(Y, col="green")
  arrows(X$x, X$y, Y$x, Y$y, angle=15, length=0.2)

## polygonal case
  A <- runifrect(100, Frame(letterR))
  A <- A[!inside.owin(A, , letterR)]
  plot(A, cols="red")
  B <- project2set(A, letterR)
  points(B, col="green")
  arrows(A$x, A$y, B$x, B$y, angle=15, length=0.2)
```

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

See Also

psp.object, as.psp, owin.object, owin, as.owin.

Function for extracting information from a segment pattern: marks.psp, summary.psp, midpoints.psp, lengths_psp angles.psp, endpoints.psp

Convert line segments to infinite lines: extrapolate.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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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.
```

psp2mask *Convert Line Segment Pattern to Binary Pixel Mask*

Description

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

Usage

```
psp2mask(x, W=NULL, ...)
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

The functions psp2mask and as.mask.psp are currently identical. In future versions of the package, as.mask.psp will be deprecated, and then removed.

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

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(psp2mask(X))
plot(X, add=TRUE, col="red")
```

quad.object *Class of Quadrature Schemes*

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 Window(dummy) while the observation window (if this needs to be specified separately) is taken to be Window(data).

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

quadscheme, ppm

---

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, left.open=TRUE,
               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 x and y directions. Incompatible with xbreaks and ybreaks.
...	Additional arguments passed to quadratcount.ppp.
xbreaks	Numeric vector giving the x coordinates of the boundaries of the rectangular quadrats. Incompatible with nx.
ybreaks	Numeric vector giving the y coordinates of the boundaries of the rectangular quadrats. Incompatible with ny.
tess	Tessellation (object of class "tess" or something acceptable to as.tess) determining the quadrats. Incompatible with nx,ny,xbreaks,ybreaks.
left.open	Logical value specifying whether rectangular quadrats are left-open and right-closed (left.open=TRUE, the default) or left-closed and right-open (left.open=FALSE).

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 * 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 Window(X).

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.

To perform a chi-squared test based on the quadrat counts, use quadrat.test.

To calculate an estimate of intensity based on the quadrat counts, use intensity.quadratcount.

To extract the quadrats used in a quadratcount object, use as.tess.

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 "solist" which has print and plot methods.

Treament of data points on the boundary

If the quadrats are rectangular, they are assumed to be left-open and right-closed, by default (left.open=TRUE). Alternatively if left.open=FALSE then rectangular quadrats are left-closed and right-open.

If the quadrats are not rectangular, the treatment of points which lie on the boundary of two quadrats is undefined, and may depend on the hardware.

Hypothesis testing

To perform a chi-squared test based on the quadrat counts, use `quadrat.test`.

Warning

If `Q` is a `quadratcount` object, the ordering of entries in the table `Q` **may be different from** the ordering of quadrats (tiles in the tessellation `as.tess(Q)`).

To obtain the entries of the table in the same order as the quadrats, use `counts <- as.numeric(t(Q))` or `counts <- marks(as.tess(Q))`.

Author(s)

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

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

`plot.quadratcount`, `intensity.quadratcount`, `quadrats`, `quadrat.test`, `tess`, `hextess`, `quadratresample`, `miplot`

Examples

```
X <- runifrect(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
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(runifrect(6))
qX <- quadratcount(X, tess=B)
plot(X, pch="+")
plot(qX, add=TRUE, col="red", cex=1.5, lty=2)
```

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 as.owin, such as a point pattern.
nx, ny	Numbers of quadrats in the x and y directions. Incompatible with xbreaks and ybreaks.
xbreaks	Numeric vector giving the x coordinates of the boundaries of the quadrats. Incompatible with nx.
ybreaks	Numeric vector giving the y coordinates of the boundaries of the quadrats. Incompatible with ny.
keepempty	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 Window(X).

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

See Also

For calculations using quadrats, see quadratcount, quadrat.test, quadratresample

For other kinds of tessellations, see tess, hextess, venn.tess, polartess, dirichlet, delaunay, quantess, bufftess and rpoislinetess.

Examples

```
W <- square(10)
Z <- quadrats(W, 4, 5)
plot(Z)

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

data	The observed data point pattern. An object of class "ppp" or in a format recognised by as.ppp()
dummy	The pattern of dummy points for the quadrature. An object of class "ppp" or in a format recognised by as.ppp() Defaults to default.dummy(data, ...)
method	The name of the method for calculating quadrature weights: either "grid" or "dirichlet".
...	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 Window(dummy). If dummy is just a list of x, y coordinates then the quadrature region defaults to the observation window of the data pattern, Window(data).

If dummy is missing, then a pattern of dummy points will be generated using default.dummy, taking account of the optional arguments .... By default, the dummy points are arranged in a rectangular grid; recognised arguments include nd (the number of grid points in the horizontal and vertical directions) and eps (the spacing between dummy points). If random=TRUE, a systematic random pattern of dummy points is generated instead. See default.dummy for details.

If method = "grid" then the optional arguments (for ...) are (nd, ntile, eps). The quadrature region (defined above) 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, eps). 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".

Error Messages

The following error messages need some explanation. (See also the list of error messages in ppm.ppp).

"Some tiles with positive area do not contain any quadrature points: relative error = X%" This is not important unless the relative error is large. In the default rule for computing the quadrature weights, space is divided into rectangular tiles, and the number of quadrature points (data and dummy points) in each tile is counted. It is possible for a tile with non-zero area to contain no quadrature points; in this case, the quadrature scheme will contribute a bias to the model-fitting procedure. **A small relative error (less than 2 percent) is not important.** Relative errors of a few percent can occur because of the shape of the window. If the relative error is greater than about 5 percent, we recommend trying different parameters for the quadrature scheme, perhaps setting a larger value of `nd` to increase the number of dummy points. A relative error greater than 10 percent indicates a major problem with the input data. The quadrature scheme should be inspected by plotting and printing it. (The most likely cause of this problem is that the spatial coordinates of the original data were not handled correctly, for example, coordinates of the locations and the window boundary were incompatible.)

"Some tiles with zero area contain quadrature points" This error message is rare, and has no consequences. It is mainly of interest to programmers. It occurs when the area of a tile is calculated to be equal to zero, but a quadrature point has been placed in the tile.

Author(s)

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

References

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

Berman, M. and Turner, T.R. Approximating point process likelihoods with GLIM. *Applied Statistics* **41** (1992) 31–38.

See Also

`ppm`, `as.ppp`, `quad.object`, `gridweights`, `dirichletWeights`, `corners`, `gridcentres`, `stratrand`, `spokes`

Examples

```
# grid weights
Q <- quadscheme(simdat)
Q <- quadscheme(simdat, method="grid")
Q <- quadscheme(simdat, eps=0.5)         # dummy point spacing 0.5 units

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
# D <- runifrect(250, Window(simdat))
# Q <- quadscheme(simdat, D, method="dirichlet", exact=FALSE)
```

```
# polygonal window
data(demopat)
X <- unmark(demopat)
Q <- quadscheme(X)

# mask window
Window(X) <- as.mask(Window(X))
Q <- quadscheme(X)
```

---

quadscheme.logi *Generate a Logistic Regression Quadrature Scheme from a Point Pattern*

Description

Generates a logistic regression quadrature scheme (an object of class "logiquad" inheriting from "quad") from point patterns of data and dummy points.

Usage

```
quadscheme.logi(data, dummy, dummytype = "stratrand",
                nd = NULL, mark.repeat = FALSE, ...)
```

Arguments

data	The observed data point pattern. An object of class "ppp" or in a format recognised by as.ppp()
dummy	The pattern of dummy points for the quadrature. An object of class "ppp" or in a format recognised by as.ppp(). If missing a sensible default is generated.
dummytype	The name of the type of dummy points to use when "dummy" is missing. Currently available options are: "stratrand" (default), "binomial", "poisson", "grid" and "transgrid".
nd	Integer, or integer vector of length 2 controlling the intensity of dummy points when "dummy" is missing.
mark.repeat	Repeating the dummy points for each level of a marked data pattern when "dummy" is missing. (See details.)
...	Ignored.

Details

This is the primary method for producing a quadrature schemes for use by ppm when the logistic regression approximation (Baddeley et al. 2013) to the pseudolikelihood of the model is applied (i.e. when method="logi" in ppm).

The function ppm fits a point process model to an observed point pattern. When used with the option method="logi" it requires a quadrature scheme consisting of the original data point pattern and an

additional pattern of dummy points. Such quadrature schemes are represented by objects of class "logiquad".

Quadrature schemes are created by the function quadscheme.logi. The arguments data and dummy specify the data and dummy points, respectively. There is a sensible default for the dummy points. Alternatively the dummy points may be specified arbitrarily and given in any format recognised by as.ppp.

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 Window(dummy). If dummy is just a list of x, y coordinates then the quadrature region defaults to the observation window of the data pattern, Window(data).

If dummy is missing, then a pattern of dummy points will be generated, taking account of the optional arguments dummytype, nd, and mark.repeat.

The currently accepted values for dummytype are:

- "grid" where the frame of the window is divided into a nd * nd or nd[1] * nd[2] regular grid of tiles and the centers constitutes the dummy points.
- "transgrid" where a regular grid as above is translated by a random vector.
- "stratrand" where each point of a regular grid as above is randomly translated within its tile.
- "binomial" where nd * nd or nd[1] * nd[2] points are generated uniformly in the frame of the window. "poisson" where a homogeneous Poisson point process with intensity nd * nd or nd[1] * nd[2] is generated within the frame of observation window.

Then if the window is not rectangular, any dummy points lying outside it are deleted.

If data is a multitype point pattern the dummy points should also be marked (with the same levels of the marks as data). If dummy is missing and the dummy pattern is generated by quadscheme.logi the default behaviour is to attach a uniformly distributed mark (from the levels of the marks) to each dummy point. Alternatively, if mark.repeat=TRUE each dummy point is repeated as many times as there are levels of the marks with a distinct mark value attached to it.

Finally, each point (data and dummy) is assigned the weight 1. The weights are never used and only appear to be compatible with the class "quad" from which the "logiquad" object inherits.

Value

An object of class "logiquad" inheriting from "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@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

References

Baddeley, A., Coeurjolly, J.-F., Rubak, E. and Waagepetersen, R. (2014) Logistic regression for spatial Gibbs point processes. *Biometrika* **101** (2) 377–392.

See Also

ppm, as.ppp

Examples

```
Q <- quadscheme.logi(simdat)
```

---

quantess *Quantile Tessellation*

Description

Divide space into tiles which contain equal amounts of stuff.

Usage

```
quantess(M, Z, n, ...)

## S3 method for class 'owin'
quantess(M, Z, n, ..., type=2, origin=c(0,0), eps=NULL)

## S3 method for class 'ppp'
quantess(M, Z, n, ..., type=2, origin=c(0,0), eps=NULL)

## S3 method for class 'im'
quantess(M, Z, n, ..., type=2, origin=c(0,0))
```

Arguments

M	A spatial object (such as a window, point pattern or pixel image) determining the weight or amount of stuff at each location.
Z	A spatial covariate (a pixel image or a function(x,y)) or one of the strings "x" or "y" indicating the Cartesian coordinates x or y, or one of the strings "rad" or "ang" indicating polar coordinates. The range of values of Z will be broken into n bands containing equal amounts of stuff.
n	Number of bands. A positive integer.
type	Integer specifying the rule for calculating quantiles. Passed to quantile.default.
...	Additional arguments passed to quadrats or tess defining another tessellation which should be intersected with the quantile tessellation.

origin	Location of the origin of polar coordinates, if Z="rad" or Z="ang". Either a numeric vector of length 2 giving the location, or a point pattern containing only one point, or a list with two entries named x and y, or one of the character strings "centroid", "midpoint", "left", "right", "top", "bottom", "topleft", "bottomleft", "topright" or "bottomright" (partially matched).
eps	Optional. The size of pixels in the approximation which is used to compute the quantiles. A positive numeric value, or vector of two positive numeric values.

Details

A *quantile tessellation* is a division of space into pieces which contain equal amounts of stuff.

The function quantess computes a quantile tessellation and returns the tessellation itself. The function quantess is generic, with methods for windows (class "owin"), point patterns ("ppp") and pixel images ("im").

The first argument M (for mass) specifies the spatial distribution of stuff that is to be divided. If M is a window, the *area* of the window is to be divided into n equal pieces. If M is a point pattern, the *number of points* in the pattern is to be divided into n equal parts, as far as possible. If M is a pixel image, the pixel values are interpreted as weights, and the *total weight* is to be divided into n equal parts.

The second argument Z is a spatial covariate. The range of values of Z will be divided into n bands, each containing the same total weight. That is, we determine the quantiles of Z with weights given by M.

For convenience, additional arguments ... can be given, to further subdivide the tiles of the tessellation. These arguments should be recognised by one of the functions quadrats or tess. The tessellation determined by these arguments is intersected with the quantile tessellation.

The result of quantess is a tessellation of as.owin(M) determined by the quantiles of Z.

Value

A tessellation (object of class "tess").

Author(s)

Original idea by Ute Hahn. Implemented in spatstat by Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>, Rolf Turner <rolfturner@posteo.net> and Ege Rubak <rubak@math.aau.dk>.

See Also

tess, quadrats, quantile, tilenames

Examples

```
plot(quantess(letterR, "x", 5))

plot(quantess(bronzefilter, "x", 6))
points(unmark(bronzefilter))

plot(quantess(letterR, "rad", 7, origin=c(2.8, 1.5)))
```

```
plot(quantess(letterR, "ang", 7, origin=c(2.8, 1.5)))

opa <- par(mar=c(0,0,2,5))
A <- quantess(Window(bei), bei.extra$elev, 4)
plot(A, ribargs=list(las=1))

B <- quantess(bei, bei.extra$elev, 4)
tilenames(B) <- paste(spatstat.utils::ordinal(1:4), "quartile")
plot(B, ribargs=list(las=1))
points(bei, pch=".", cex=2, col="white")
par(opa)
```

`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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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)
```

---

quantilefun.im *Quantile Function for Images*

Description

Return the inverse function of the cumulative distribution function of pixel values in an image.

Usage

```
  ## S3 method for class 'im'
quantilefun(x, ..., type=1)
```

Arguments

x	Pixel image (object of class "im").
...	Other arguments passed to methods.
type	Integer specifying the type of quantiles, as explained in quantile.default. Only types 1 and 2 are currently implemented.

Details

Whereas the command quantile calculates the quantiles of a dataset corresponding to desired probabilities p, the command quantilefun returns a function which can be used to compute any quantiles of the dataset.

If f <- quantilefun(x) then f is a function such that f(p) is the quantile associated with any given probability p. For example f(0.5) is the median of the original data, and f(0.99) is the 99th percentile of the original data.

If x is a pixel image (object of class "im") then the pixel values of x will be extracted and the quantile function of the pixel values is constructed.

Value

A function in the R language.

Author(s)

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

See Also

quantilefun, ewcdf, quantile.ewcdf, ecdf, quantile

Examples

```
## image data: terrain elevation
Z <- bei.extra$elev
if(require(spatstat.explore)) {
  FE <- spatialcdf(Z, normalise=TRUE)
} else {
  FE <- ecdf(Z[])
}
QE <- quantilefun(FE)
QE(0.5) # median elevation
if(interactive()) plot(QE, xlim=c(0,1),
   xlab="probability", ylab="quantile of elevation")
```

---

quasirandom *Quasirandom Patterns*

Description

Generates quasirandom sequences of numbers and quasirandom spatial patterns of points in any dimension.

Usage

```
vdCorput(n, base)

Halton(n, bases = c(2, 3), raw = FALSE, simplify = TRUE)

Hammersley(n, bases = 2, raw = FALSE, simplify = TRUE)
```

Arguments

n	Number of points to generate.
base	A prime number giving the base of the sequence.
bases	Vector of prime numbers giving the bases of the sequences for each coordinate axis.
raw	Logical value indicating whether to return the coordinates as a matrix (raw=TRUE) or as a spatial point pattern (raw=FALSE, the default).
simplify	Argument passed to ppx indicating whether point patterns of dimension 2 or 3 should be returned as objects of class "ppp" or "pp3" respectively (simplify=TRUE, the default) or as objects of class "ppx" (simplify=FALSE).

Details

The function vdCorput generates the quasirandom sequence of Van der Corput (1935) of length n with the given base. These are numbers between 0 and 1 which are in some sense uniformly distributed over the interval.

The function Halton generates the Halton quasirandom sequence of points in d-dimensional space, where d = length(bases). The values of the i-th coordinate of the points are generated using the van der Corput sequence with base equal to bases[i].

The function Hammersley generates the Hammersley set of points in d+1-dimensional space, where d = length(bases). The first d coordinates of the points are generated using the van der Corput sequence with base equal to bases[i]. The d+1-th coordinate is the sequence 1/n, 2/n, ..., 1.

If raw=FALSE (the default) then the Halton and Hammersley sets are interpreted as spatial point patterns of the appropriate dimension. They are returned as objects of class "ppx" (multidimensional point patterns) unless simplify=TRUE and d=2 or d=3 when they are returned as objects of class "ppp" or "pp3". If raw=TRUE, the coordinates are returned as a matrix with n rows and D columns where D is the spatial dimension.

Value

For vdCorput, a numeric vector.

For Halton and Hammersley, an object of class "ppp", "pp3" or "ppx"; or if raw=TRUE, a numeric matrix.

Author(s)

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

, Rolf Turner <rolfturner@posteo.net>

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

References

Van der Corput, J. G. (1935) Verteilungsfunktionen. *Proc. Ned. Akad. v. Wetensch.* **38**: 813–821.

Kuipers, L. and Niederreiter, H. (2005) *Uniform distribution of sequences*, Dover Publications.

See Also

rQuasi

Examples

```
vdCorput(10, 2)

plot(Halton(256, c(2,3)))

plot(Hammersley(256, 3))
```

raster.x *Cartesian Coordinates for a Pixel Raster*

Description

Return the x and y coordinates of each pixel in a pixel image or binary mask.

Usage

```
raster.x(w, drop=FALSE)
raster.y(w, drop=FALSE)
raster.xy(w, drop=FALSE)
```

Arguments

w	A pixel image (object of class "im") or a mask window (object of class "owin" of type "mask").
drop	Logical. If TRUE, then coordinates of pixels that lie outside the window are removed. If FALSE (the default) then the coordinates of every pixel in the containing rectangle are retained.

Details

The argument w should be either a pixel image (object of class "im") or a mask window (an object of class "owin" of type "mask").

If drop=FALSE (the default), the functions raster.x and raster.y return a matrix of the same dimensions as the pixel image or mask itself, with entries giving the x coordinate (for raster.x) or y coordinate (for raster.y) of each pixel in the pixel grid.

If drop=TRUE, pixels that lie outside the window w (or outside the domain of the image w) are removed, and raster.x and raster.y return numeric vectors containing the coordinates of the pixels that are inside the window w.

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.xy returns a list with components x and y which are numeric vectors of equal length containing the pixel coordinates.

If drop=FALSE, 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.

If drop=TRUE, raster.x and raster.y return numeric vectors.

Author(s)

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

See Also

owin, as.mask, pixelcentres

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))
# plot(disc)
# approximation to the unit disc
```

---

rectdistmap *Distance Map Using Rectangular Distance Metric*

Description

Computes the distance map of a spatial region based on the rectangular distance metric.

Usage

```
rectdistmap(X, asp = 1, npasses=1, verbose=FALSE)
```

Arguments

X	A window (object of class "owin").
asp	Aspect ratio for the metric. See Details.
npasses	Experimental.
verbose	Logical value indicating whether to print trace information.

Details

This function computes the distance map of the spatial region X using the rectangular distance metric with aspect ratio asp. This metric is defined so that the set of all points lying at most 1 unit away from the origin (according to the metric) form a rectangle of width 1 and height asp.

Value

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

Author(s)

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

See Also

distmap

Examples

```
V <- letterR
Frame(V) <- grow.rectangle(Frame(V), 0.5)
plot(rectdistmap(V))
```

`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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

`affine`, `flipxy`

Examples

```
plot(reflect(as.im(letterR)))
plot(reflect(letterR), add=TRUE)
```

`regularpolygon` *Create A Regular Polygon*

Description

Create a window object representing a regular (equal-sided) polygon.

Usage

```
regularpolygon(n, edge = 1, centre = c(0, 0), ...,
               align = c("bottom", "top", "left", "right", "no"))

hexagon(edge = 1, centre = c(0,0), ...,
        align = c("bottom", "top", "left", "right", "no"))
```

Arguments

`n`	Number of edges in the polygon.
`edge`	Length of each edge in the polygon. A single positive number.
`centre`	Coordinates of the centre of the polygon. A numeric vector of length 2, or a `list(x,y)` giving the coordinates of exactly one point, or a point pattern (object of class `"ppp"`) containing exactly one point.
`align`	Character string specifying whether to align one of the edges with a vertical or horizontal boundary.
`...`	Ignored.

Details

The function `regularpolygon` creates a regular (equal-sided) polygon with `n` sides, centred at `centre`, with sides of equal length `edge`. The function `hexagon` is the special case `n=6`.

The orientation of the polygon is determined by the argument `align`. If `align="no"`, one vertex of the polygon is placed on the x-axis. Otherwise, an edge of the polygon is aligned with one side of the frame, specified by the value of `align`.

Value

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

Author(s)

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

See Also

disc, ellipse, owin.

hextess for hexagonal tessellations.

Examples

```
plot(hexagon())
plot(regularpolygon(7))
plot(regularpolygon(7, align="left"))
```

---

relevel.im *Reorder Levels of a Factor-Valued Image or Pattern*

Description

For a pixel image with factor values, or a point pattern with factor-valued marks, the levels of the factor are re-ordered so that the level ref is first and the others are moved down.

Usage

```
  ## S3 method for class 'im'
relevel(x, ref, ...)

  ## S3 method for class 'ppp'
relevel(x, ref, ...)

  ## S3 method for class 'ppx'
relevel(x, ref, ...)
```

Arguments

x	A pixel image (object of class "im") with factor values, or a point pattern (object of class "ppp", "ppx", "lpp" or "pp3") with factor-valued marks.
ref	The reference level.
...	Ignored.

Details

These functions are methods for the generic relevel.

If x is a pixel image (object of class "im") with factor values, or a point pattern (object of class "ppp", "ppx", "lpp" or "pp3") with factor-valued marks, the levels of the factor are changed so that the level specified by ref comes first.

Value

Object of the same kind as x.

Author(s)

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

Rolf Turner <rolfturner@posteo.net>

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

See Also

mergeLevels

Examples

```
amacrine
relevel(amacrine, "on")
```

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, j, ..., drop=TRUE] <- value
```

Arguments

x	A two-dimensional pixel image. An object of class "im".
i	Object defining the subregion or subset to be replaced. 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 any type of index that applies to a matrix, or something that can be converted to a point pattern by as.ppp (using the window of x).
j	An integer or logical vector serving as the column index if matrix indexing is being used. Ignored if i is appropriate to some sort of replacement *other than* matrix indexing.
...	Ignored.
drop	Logical value specifying what happens when i and j are both missing. See Details.
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 arguments `i,j` according to the following rules (which are checked in this order):

1. `i` is a spatial object such as a window, a pixel image with logical values, or a point pattern; or
2. `i,j` are indices for the matrix `as.matrix(x)`; or
3. `i` can be converted to a point pattern by `as.ppp(i, W=Window(x))`, and `i` is not a matrix.

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"`), 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 the algorithm tries to interpret `i,j` as indices for the matrix `as.matrix(x)`. Either `i` or `j` may be missing or blank.

If none of the conditions above are met, and if `i` is not a matrix, then `i` is converted into a point pattern by `as.ppp(i, W=Window(x))`. Again the values of the pixel image at the points of this pattern are changed.

If `i` and `j` are both missing, as in the call `x[] <- value`, then all pixel values in `x` are replaced by `value`:

- If `drop=TRUE` (the default), then this replacement applies only to pixels whose values are currently defined (i.e. where the current pixel value is not `NA`). If `value` is a vector, then its length must equal the number of pixels whose values are currently defined.
- If `drop=FALSE` then the replacement applies to all pixels inside the rectangle `Frame(x)`. If `value` is a vector, then its length must equal the number of pixels in the entire rectangle.

Value

The image `x` with the values replaced.

Warning

If you have a 2-column matrix containing the x, y coordinates of point locations, then to prevent this being interpreted as an array index, you should convert it to a `data.frame` or to a point pattern.

Author(s)

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

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
R <- affine(letterR, diag(c(1,1)/2), c(-2,-0.7))
X[R] <- 3
plot(X)

# a point pattern
X[cells] <- 10
plot(X)

# change pixel value at a specific location
X[list(x=0.1,y=0.2)] <- 7

# matrix indexing --- single vector index
X[1:2570] <- 10
plot(X)

# matrix indexing using double indices
X[1:257,1:10] <- 5
plot(X)

# matrix indexing using a matrix of indices
X[cbind(1:257,1:257)] <- 10
X[cbind(257:1,1:257)] <- 10
plot(X)

# Blank indices
Y <- as.im(letterR)
plot(Y)
Y[] <- 42  # replace values only inside the window 'R'
plot(Y)
Y[drop=FALSE] <- 7 # replace all values in the rectangle
plot(Y)

Z <- as.im(letterR)
Z[] <- raster.x(Z, drop=TRUE) # excludes NA
plot(Z)
Z[drop=FALSE] <- raster.y(Z, drop=FALSE) # includes NA
plot(Z)
```

`requireversion` *Require a Specific Version of a Package*

Description

Checks that the version number of a specified package is greater than or equal to the specified version number. For use in stand-alone R scripts.

Usage

```
requireversion(pkg, ver, fatal=TRUE)
```

Arguments

pkg	Package name.
ver	Character string containing version number.
fatal	Logical value indicating whether an error should occur when the package version is less than ver.

Details

This function checks whether the installed version of the package pkg is greater than or equal to ver. By default, an error occurs if this condition is not met.

It is useful in stand-alone R scripts, which often require a particular version of a package in order to work correctly.

This function should not be used inside a package: for that purpose, the dependence on packages and versions should be specified in the package description file.

Value

A logical value.

Author(s)

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

Examples

```
requireversion(spatstat.geom, "1.42-0")
requireversion(spatstat.data, "999.999-999", fatal=FALSE)
```

---

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, unitname)
```

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").
s	Conversion factor: the new units are s times the old units.
unitname	Optional. New name for the unit of length. See unitname.

Details

This is generic. Methods are provided for many spatial objects.

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. The name of the unit of length will also be adjusted. The result is an object of the same type, representing the same data, but expressed in the new units.

For example if X is a dataset giving coordinates in metres, then rescale(X,1000) will take the new unit of length to be 1000 metres. To do this, it will divide the old coordinate values by 1000 to obtain coordinates expressed in kilometres, and change the name of the unit of length from "metres" to "1000 metres".

If unitname is given, it will be taken as the new name of the unit of length. It should be a valid name for the unit of length, as described in the help for unitname. For example if X is a dataset giving coordinates in metres, rescale(X, 1000, "km") will divide the coordinate values by 1000 to obtain coordinates in kilometres, and the unit name will be changed to "km".

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

See Also

Available methods: rescale.im, rescale.layered, rescale.owin, rescale.ppp, rescale.psp and rescale.unitname.

Other generics: unitname, 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, unitname)
```

Arguments

X	Pixel image (object of class "im").
s	Conversion factor: the new units are s times the old units.
unitname	Optional. New name for the unit of length. See unitname.

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).

If s is missing, then the coordinates will be re-expressed in 'native' units; for example if the current unit is equal to 0.1 metres, then the coordinates will be re-expressed in metres.

The result is a pixel image representing the *same* data but re-expressed in a different unit.

Pixel values are unchanged. This may not be what you intended!

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

See Also

im, rescale, unitname, eval.im

Examples

```
# Bramble Canes data: 1 unit = 9 metres
  bramblecanes
# distance transform
  Z <- distmap(bramblecanes)
# convert to metres
# first alter the pixel values
  Zm <- eval.im(9 * Z)
# now rescale the pixel coordinates
  Z <- rescale(Zm, 1/9)
# or equivalently
  Z <- rescale(Zm)
```

`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, unitname)
```

Arguments

`X`	Window (object of class `"owin"`).
`s`	Conversion factor: the new units are `s` times the old units.
`unitname`	Optional. New name for the unit of length. See `unitname`.

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.

If `s` is missing, then the coordinates will be re-expressed in 'native' units; for example if the current unit is equal to 0.1 metres, then the coordinates will be re-expressed in metres.

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

See Also

unitname, rescale, rescale.owin, affine, rotate, shift

Examples

```
  W <- Window(swedishpines)
  W
# coordinates are in decimetres (0.1 metre)
# convert to metres:
  rescale(W, 10)
# or equivalently
  rescale(W)
```

---

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, unitname)
```

Arguments

X	Point pattern (object of class "ppp").
s	Conversion factor: the new units are s times the old units.
unitname	Optional. New name for the unit of length. See unitname.

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.

If s is missing, then the coordinates will be re-expressed in 'native' units; for example if the current unit is equal to 0.1 metres, then the coordinates will be re-expressed in metres.

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

See Also

unitname, rescale, rescale.owin, affine, rotate, shift

Examples

```
# Bramble Canes data: 1 unit = 9 metres
# convert to metres
  bram <- rescale(bramblecanes, 1/9)
# or equivalently
  bram <- rescale(bramblecanes)
```

---

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, unitname)
```

Arguments

X	Line segment pattern (object of class "psp").
s	Conversion factor: the new units are s times the old units.
unitname	Optional. New name for the unit of length. See unitname.

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.

If s is missing, then the coordinates will be re-expressed in 'native' units; for example if the current unit is equal to 0.1 metres, then the coordinates will be re-expressed in metres.

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

See Also

units, affine, rotate, shift

Examples

```
X <- copper$Lines
X
# data are in km
# convert to metres
rescale(X, 1/1000)

# convert data and rename unit
rescale(X, 1/1000, c("metre", "metres"))
```

`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@curtin.edu.au>`

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

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)
```

`restrict.colourmap` *Restrict a Colour Map to a Subset of Values*

Description

Given a colour map defined on a range of numerical values or a set of discrete inputs, the command restricts the range of values to a narrower range, or restricts the set of inputs to a subset, and returns the associated colour map.

Usage

```
restrict.colourmap(x, ..., range = NULL, breaks = NULL, inputs = NULL)
```

Arguments

`x`	Colour map (object of class `"colourmap"`).
`...`	Ignored.
`range`	New, restricted range of numerical values to which the colour map will apply. A numeric vector of length 2 giving the minimum and maximum values of the input. Incompatible with `breaks` and `inputs`.
`breaks`	Vector of breakpoints for the new colour map. A numeric vector with increasing entries. Incompatible with `range` and `inputs`.
`inputs`	Values accepted as inputs for the new colour map. A factor or vector. Incompatible with `breaks` and `range`.

Details

This command produces a new colour map `y` which is consistent with the original colour map `x`, except that `y` is defined on a narrower interval of numeric values, or a smaller set of discrete input values, than `x`.

Value

Colour map (object of class `"colourmap"`).

Author(s)

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

See Also

`colourmap`

Examples

```
plot(a <- colourmap(topo.colors(128), range=c(-1,1)))
plot(b <- restrict.colourmap(a, range=c(0,1)))
```

`rev.colourmap` *Reverse the Colours in a Colour Map*

Description

Reverse the sequence of colour values in a colour map.

Usage

```
  ## S3 method for class 'colourmap'
rev(x)
```

Arguments

`x`	A colour map (object of class `"colourmap"`).

Details

This is a method for the generic `rev` for the class of colour maps.

The sequence of colour values in the colour map will be reversed, without changing any other details.

Value

A colour map (object of class `"colourmap"`).

Author(s)

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

See Also

`colourmap`, `colouroutputs`

Examples

```
co <- colourmap(rainbow(100), range=c(-1,1))
opa <- par(mfrow=c(1,2))
plot(co, vertical=TRUE)
plot(rev(co), vertical=TRUE)
par(opa)
```

rexplode *Explode a Point Pattern by Displacing Duplicated Points*

Description

Given a point pattern which contains duplicated points, separate the duplicated points from each other by slightly perturbing their positions.

Usage

```
rexplode(X, ...)

## S3 method for class 'ppp'
rexplode(X, radius, ..., nsim = 1, drop = TRUE)
```

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. There is a sensible default. Alternatively, radius may be a numeric vector of length equal to the number of points in X, giving a different displacement radius for each data point. Radii will be restricted to be less than or equal to the distance to the boundary of the window.
...	Ignored.
nsim	Number of simulated realisations to be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a point pattern.

Details

Duplicated points in the point pattern X are identified. Each group of duplicated points is then 'exploded' by randomly displacing the point locations to form a circular arrangement around the original position.

This function is an alternative to rjitter.ppp. Whereas rjitter.ppp applies independent random displacements to each data point, rexplode.ppp applies displacements only to the points that are duplicated, and the displacements are mutually dependent within each group of duplicates, to ensure that the displaced points are visually separated from each other.

First the code ensures that the displacement radius for each data point is less than or equal to the distance to the boundary of the window. Then each group of duplicated points (or data points with the same location but possibly different mark values) is taken in turn. The first element of the group is randomly displaced by a vector uniformly distributed in a circle of radius radius. The remaining elements of the group are then positioned around the original location, at the same distance from the orginal location, with equal angular spacing from the first point. The result is that each group of duplicated points becomes a circular pattern centred around the original location.

Value

A point pattern (an object of class "ppp") or a list of point patterns.

Author(s)

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

See Also

rjitter.ppp

Examples

```
## create a pattern containing duplicated points
X <- runifrect(5) %mark% letters[1:5]
X <- X[rep(1:5, 1 + rpois(5, 2))]
## explode it
Y <- rexplode(X, 0.05)
## display
if(interactive()) {
    plot(solist(X=X, 'explode(X)'=Y),
    main="", cols=2:6, cex=1.25, leg.side="bottom")
}
```

---

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, A, maxColorValue=255, autoscale=FALSE)
hsvim(H, S, V, A, autoscale=FALSE)
```

Arguments

R, G, B	Pixel images (objects of class "im") or constants giving the red, green, and blue components of a colour, respectively.
A	Optional. Pixel image or constant value giving the alpha (transparency) component of a colour.
maxColorValue	Maximum colour channel value for R,G,B,A.
H, S, V	Pixel images (objects of class "im") or constants giving the hue, saturation, and value components of a colour, respectively.
autoscale	Logical. If TRUE, input values are automatically rescaled to fit the permitted range. RGB values are scaled to lie between 0 and maxColorValue. HSV values are scaled to lie between 0 and 1.

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

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))
M <- Window(X)
Y <- as.im(function(x,y){(x+1)/2}, W=M)
Z <- as.im(function(x,y){(y+1)/2}, W=M)
# convert
RGB <- rgbim(X, Y, Z, maxColorValue=1)
HSV <- hsvim(X, Y, Z)
opa <- par(mfrow=c(1,2))
plot(RGB, valuesAreColours=TRUE)
plot(HSV, valuesAreColours=TRUE)
par(opa)
```

---

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=c("convex", "rectangle"), f)
```

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.
shape	String (partially matched) indicating the type of window to be estimated: either "convex" or "rectangle".
f	(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 $(n+1)/(n-1)$ 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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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="ripras(x,y)")
plot(w, add=TRUE)
points(x,y)

X <- runifrect(15)
plot(X, main="ripras(X)")
plot(ripras(X), add=TRUE)

# two points insufficient
ripras(c(0,1),c(0,0))
# triangle
ripras(c(0,1,0.5), c(0,0,1))
# three collinear points
ripras(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, ...)

## S3 method for class 'ppp'
rjitter(X, radius, retry=TRUE, giveup = 10000, trim=FALSE,
        ..., nsim=1, drop=TRUE, adjust=1)
```

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. There is a sensible default. Alternatively, radius may be a numeric vector of length equal to the number of points in X, giving a different displacement radius for each data point.
retry	What to do when a perturbed point lies outside the window of the original point pattern. If retry=FALSE, the point will be lost; if retry=TRUE, the algorithm will try again.
giveup	Maximum number of unsuccessful attempts.
trim	Logical value. If TRUE, the displacement radius for each data point will be constrained to be less than or equal to the distance from the data point to the window boundary. This ensures that all displaced points will fall inside the window.

...	Ignored.
nsim	Number of simulated realisations to be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a point pattern.
adjust	Adjustment factor applied to the radius. A numeric value or numeric vector.

Details

The function rjitter is generic, with methods for point patterns (described here) and for some other types of geometrical objects.

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 the perturbed point 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.

If trim=TRUE, then the displacement radius for each data point will be constrained to be less than or equal to the distance from the data point to the window boundary. This ensures that the randomly displaced points will always fall inside the window; no displaced points will be lost and no retrying will be required. However, it implies that a point lying exactly on the boundary will never be perturbed.

If adjust is given, the jittering radius will be multiplied by adjust. This allows the user to specify that the radius should be a multiple of the default radius.

The resulting point pattern has an attribute "radius" giving the value of radius used. If retry=TRUE, the resulting point pattern also has an attribute "tries" reporting the maximum number of trials needed to ensure that all jittered points were inside the window.

Value

The result of rjitter.ppp is a point pattern (an object of class "ppp") or a list of point patterns.

Each point pattern has attributes "radius" and (if retry=TRUE) "tries".

Author(s)

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

See Also

rexplode, rjitter.pp3

Examples

```
X <- rsyst(owin(), 10, 10)
Y <- rjitter(X, 0.02)
plot(Y)
Z <- rjitter(X)
U <- rjitter(X, 0.025, trim=TRUE)
```

---

rjitter.pp3 *Random Perturbation of a 3D Point Pattern*

Description

Applies independent random displacements to each point in a three-dimensional point pattern.

Usage

```
## S3 method for class 'pp3'
rjitter(X, radius, retry=TRUE, giveup = 10000, trim=FALSE,
        ..., nsim=1, drop=TRUE, adjust=1)
```

Arguments

X	A point pattern (object of class "ppp").
radius	Scale of perturbations. A positive numerical value. The displacement vectors will be uniformly distributed inside a sphere of this radius. There is a sensible default. Alternatively, radius may be a numeric vector of length equal to the number of points in X, giving a different displacement radius for each data point.
retry	What to do when a perturbed point lies outside the window of the original point pattern. If retry=FALSE, the point will be lost; if retry=TRUE, the algorithm will try again.
giveup	Maximum number of unsuccessful attempts.
trim	Logical value. If TRUE, the displacement radius for each data point will be constrained to be less than or equal to the distance from the data point to the window boundary. This ensures that all displaced points will fall inside the window.
...	Ignored.
nsim	Number of simulated realisations to be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a point pattern.
adjust	Adjustment factor applied to the radius. A numeric value or numeric vector.

Details

The function rjitter is generic, with methods for point patterns and for some other types of geometrical objects. This help page describes the method for three-dimensional point patterns of class "pp3".

Each of the points in the point pattern X is subjected to an independent random displacement. The displacement vectors are uniformly distributed in a sphere of radius radius.

If a displaced point lies outside the domain of the pattern, 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 domain, the algorithm will reject the perturbed point and generate another proposed perturbation of the original point, until one lies inside the domain, 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.

If trim=TRUE, then the displacement radius for each data point will be constrained to be less than or equal to the distance from the data point to the domain boundary. This ensures that the randomly displaced points will always fall inside the domain; no displaced points will be lost and no retrying will be required. However, it implies that a point lying exactly on the boundary will never be perturbed.

If adjust is given, the jittering radius will be multiplied by adjust. This allows the user to specify that the radius should be a multiple of the default radius.

The resulting point pattern has an attribute "radius" giving the value of radius used. If retry=TRUE, the resulting point pattern also has an attribute "tries" reporting the maximum number of trials needed to ensure that all jittered points were inside the domain.

Value

The result of rjitter.pp3 is a three-dimensional point pattern (an object of class "pp3") or a list of 3D point patterns.

Each point pattern has attributes "radius" and (if retry=TRUE) "tries".

Author(s)

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

See Also

rjitter

Examples

```
X <- osteo$pts[[1]]
Y <- rjitter(X, 5)
plot(Y)
Z <- rjitter(X)
U <- rjitter(X, 5, trim=TRUE)
```

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

psp, rpoisline

Examples

```
plot(rlinegrid(30, 0.05))
```

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

rotate.ppp, rotate.owin

rotate.im *Rotate a Pixel Image*

Description

Rotates a pixel image

Usage

```
 ## S3 method for class 'im'
rotate(X, angle=pi/2, ..., centre=NULL)
```

Arguments

X	A pixel image (object of class "im").
angle	Angle of rotation, in radians.
...	Ignored.
centre	Centre of rotation. Either a vector of length 2, or a character string (partially matched to "centroid", "midpoint" or "bottomleft"). The default is the coordinate origin c(0,0).

Details

The image is rotated by the angle specified. Angles are measured in radians, anticlockwise. The default is to rotate the image 90 degrees anticlockwise.

Value

Another object of class "im" representing the rotated pixel image.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

affine.im, shift.im, rotate

Examples

```
Z <- distmap(letterR)
X <- rotate(Z)
# plot(X)
Y <- rotate(X, centre="midpoint")
```

---

rotate.infline *Rotate or Shift Infinite Lines*

Description

Given the coordinates of one or more infinite straight lines in the plane, apply a rotation or shift.

Usage

```
## S3 method for class 'infline'
rotate(X, angle = pi/2, ...)

## S3 method for class 'infline'
shift(X, vec = c(0,0), ...)

## S3 method for class 'infline'
reflect(X)

## S3 method for class 'infline'
flipxy(X)
```

Arguments

X	Object of class "infline" representing one or more infinite straight lines in the plane.
angle	Angle of rotation, in radians.
vec	Translation (shift) vector: a numeric vector of length 2, or a list(x,y), or a point pattern containing one point.
...	Ignored.

Details

These functions are methods for the generic shift, rotate, reflect and flipxy for the class "infline".

An object of class "infline" represents one or more infinite lines in the plane.

Value

Another "infline" object representing the result of the transformation.

Author(s)

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

See Also

infline

Examples

```
L <- infline(v=0.5)

plot(square(c(-1,1)), main="rotate lines", type="n")
points(0, 0, pch=3)
plot(L, col="green")
plot(rotate(L, pi/12), col="red")
plot(rotate(L, pi/6), col="red")
```

```
plot(rotate(L, pi/4), col="red")

L <- infline(p=c(0.4, 0.9), theta=pi* c(0.2, 0.6))

plot(square(c(-1,1)), main="shift lines", type="n")
L <- infline(p=c(0.7, 0.8), theta=pi* c(0.2, 0.6))
plot(L, col="green")
plot(shift(L, c(-0.5, -0.4)), col="red")

plot(square(c(-1,1)), main="reflect lines", type="n")
points(0, 0, pch=3)
L <- infline(p=c(0.7, 0.8), theta=pi* c(0.2, 0.6))
plot(L, col="green")
plot(reflect(L), col="red")
```

`rotate.owin` *Rotate a Window*

Description

Rotates a window

Usage

```
 ## S3 method for class 'owin'
rotate(X, angle=pi/2, ..., rescue=TRUE, centre=NULL)
```

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.
`centre`	Centre of rotation. Either a vector of length 2, or a character string (partially matched to `"centroid"`, `"midpoint"` or `"bottomleft"`). The default is the coordinate origin `c(0,0)`.

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, by default, unless `centre` is specified.

Value

Another object of class `"owin"` representing the rotated window.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

owin.object

Examples

```
w <- owin(c(0,1),c(0,1))
v <- rotate(w, pi/3)
e <- rotate(w, pi/2, centre="midpoint")
# plot(v)
w <- as.mask(letterR)
v <- rotate(w, pi/5)
```

---

rotate.ppp *Rotate a Point Pattern*

Description

Rotates a point pattern

Usage

```
 ## S3 method for class 'ppp'
rotate(X, angle=pi/2, ..., centre=NULL)
```

Arguments

X	A point pattern (object of class "ppp").
angle	Angle of rotation.
...	Arguments passed to rotate.owin affecting the handling of the observation window, if it is a binary pixel mask.
centre	Centre of rotation. Either a vector of length 2, or a character string (partially matched to "centroid", "midpoint" or "bottomleft"). The default is the coordinate origin c(0,0).

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

See Also

ppp.object, rotate.owin

Examples

```
X <- rotate(cells, pi/3)
# plot(X)
```

---

rotate.psp *Rotate a Line Segment Pattern*

Description

Rotates a line segment pattern

Usage

```
 ## S3 method for class 'psp'
rotate(X, angle=pi/2, ..., centre=NULL)
```

Arguments

X	A line segment pattern (object of class "psp").
angle	Angle of rotation.
...	Arguments passed to rotate.owin affecting the handling of the observation window, if it is a binary pixel mask.
centre	Centre of rotation. Either a vector of length 2, or a character string (partially matched to "centroid", "midpoint" or "bottomleft"). The default is the coordinate origin c(0,0).

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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)
```

---

round.ppp *Apply Numerical Rounding to Spatial Coordinates*

Description

Apply numerical rounding to the spatial coordinates of a point pattern.

Usage

```
## S3 method for class 'ppp'
round(x, digits = 0, ...)

## S3 method for class 'pp3'
round(x, digits = 0, ...)

## S3 method for class 'ppx'
round(x, digits = 0, ...)
```

Arguments

x	A spatial point pattern in any dimension (object of class "ppp", "pp3" or "ppx").
digits	integer indicating the number of decimal places.
...	Additional arguments passed to the default method.

Details

These functions are methods for the generic function round. They apply numerical rounding to the spatial coordinates of the point pattern x.

Value

A point pattern object, of the same class as x.

Author(s)

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

See Also

rounding.ppp to determine whether numbers have been rounded.

round in the Base package.

Examples

```
round(cells, 1)
```

---

rounding.ppp *Detect Numerical Rounding*

Description

Given a numeric vector, or an object containing numeric spatial coordinates, determine whether the values have been rounded to a certain number of decimal places.

Usage

```
## S3 method for class 'ppp'
rounding(x)

## S3 method for class 'pp3'
rounding(x)

## S3 method for class 'ppx'
rounding(x)
```

Arguments

x	A point pattern (object of class ppp, pp3 or ppx).

Details

The functions documented here are methods for the generic rounding. They determine whether the coordinates of a spatial object have been rounded to a certain number of decimal places.

- If the coordinates of the points in x are not all integers, then rounding(x) returns the smallest number of digits d after the decimal point such that round(coords(x), digits=d) is identical to coords(x). For example if rounding(x) = 2 then the coordinates of the points in x appear to have been rounded to 2 decimal places, and are multiples of 0.01.

- If all the coordinates of the points in x are integers, then rounding(x) returns -d, where d is the smallest number of digits *before* the decimal point such that round(coords(x), digits=-d) is identical to coords(x). For example if rounding(x) = -3 then the coordinates of all points in x are multiples of 1000. If rounding(x) = 0 then the entries of x are integers but not multiples of 10.
- If all coordinates of points in x are equal to 0, a value of 0 is returned.

Value

An integer.

Author(s)

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

See Also

round.ppp, rounding

Examples

```
rounding(cells)
```

---

rQuasi *Generate Quasirandom Point Pattern in Given Window*

Description

Generates a quasirandom pattern of points in any two-dimensional window.

Usage

```
rQuasi(n, W, type = c("Halton", "Hammersley"), ...)
```

Arguments

n	Maximum number of points to be generated.
W	Window (object of class "owin") in which to generate the points.
type	String identifying the quasirandom generator.
...	Arguments passed to the quasirandom generator.

Details

This function generates a quasirandom point pattern, using the quasirandom sequence generator Halton or Hammersley as specified.

If W is a rectangle, exactly n points will be generated.

If W is not a rectangle, n points will be generated in the containing rectangle as.rectangle(W), and only the points lying inside W will be retained.

Value

Point pattern (object of class "ppp") inside the window W.

Author(s)

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

, Rolf Turner <rolfturner@posteo.net>

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

See Also

Halton

Examples

```
plot(rQuasi(256, letterR))
```

---

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=NULL, ny=nx, ..., dx=NULL, dy=dx,
      nsim=1, drop=TRUE)
```

Arguments

win	A window. An object of class owin, or data in any format acceptable to as.owin().
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.
...	Ignored.
dx	Spacing of grid points in x direction. Incompatible with nx.
dy	Spacing of grid points in y direction. Incompatible with ny.
nsim	Number of simulated realisations to be generated.
drop	Logical. If nsim=1 and drop=TRUE (the default), the result will be a point pattern, rather than a list containing a point pattern.

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 in the x direction is determined either by the number of columns `nx` or by the horizontal spacing `dx`. The grid spacing in the y direction is determined either by the number of rows `ny` or by the vertical spacing `dy`.

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 (an object of class `"ppp"`) if `nsim=1`, or a list of point patterns if `nsim > 1`.

Author(s)

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

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

See Also

`rstrat`, `runifpoint`, `quadscheme`

Examples

```
X <- rsyst(nx=10)
plot(X)

# polygonal boundary
X <- rsyst(letterR, 5, 10)
plot(X)
```

`run.simplepanel` *Run Point-and-Click Interface*

Description

Execute various operations in a simple point-and-click user interface.

Usage

```
run.simplepanel(P, popup=TRUE, verbose = FALSE)
clear.simplepanel(P)
redraw.simplepanel(P, verbose = FALSE)
```

Arguments

P	An interaction panel (object of class "simplepanel", created by simplepanel or grow.simplepanel).
popup	Logical. If popup=TRUE (the default), the panel will be displayed in a new popup window. If popup=FALSE, the panel will be displayed on the current graphics window if it already exists, and on a new window otherwise.
verbose	Logical. If TRUE, debugging information will be printed.

Details

These commands enable the user to run a simple, robust, point-and-click interface to any R code. The interface is implemented using only the basic graphics package in R.

The argument P is an object of class "simplepanel", created by simplepanel or grow.simplepanel, which specifies the graphics to be displayed and the actions to be performed when the user interacts with the panel.

The command run.simplepanel(P) activates the panel: the display is initialised and the graphics system waits for the user to click the panel. While the panel is active, the user can only interact with the panel; the R command line interface and the R GUI cannot be used. When the panel terminates (typically because the user clicked a button labelled Exit), control returns to the R command line interface and the R GUI.

The command clear.simplepanel(P) clears all the display elements in the panel, resulting in a blank display except for the title of the panel.

The command redraw.simplepanel(P) redraws all the buttons of the panel, according to the redraw functions contained in the panel.

If popup=TRUE (the default), run.simplepanel begins by calling dev.new so that a new popup window is created; this window is closed using dev.off when run.simplepanel terminates. If popup=FALSE, the panel will be displayed on the current graphics window if it already exists, and on a new window otherwise; this window is not closed when run.simplepanel terminates.

For more sophisticated control of the graphics focus (for example, to use the panel to control the display on another window), initialise the graphics devices yourself using dev.new or similar commands; save these devices in the shared environment env of the panel P; and write the click/redraw functions of P in such a way that they access these devices using dev.set. Then use run.simplepanel with popup=FALSE.

Value

The return value of run.simplepanel(P) is the value returned by the exit function of P. See simplepanel.

The functions clear.simplepanel and redraw.simplepanel return NULL.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

simplepanel

Examples

```
if(interactive()) {
  # make boxes (alternatively use layout.boxes())
  Bminus <- square(1)
  Bvalue <- shift(Bminus, c(1.2, 0))
  Bplus <- shift(Bvalue, c(1.2, 0))
  Bdone <- shift(Bplus, c(1.2, 0))
  myboxes <- list(Bminus, Bvalue, Bplus, Bdone)
  myB <- do.call(boundingbox,myboxes)

  # make environment containing an integer count
  myenv <- new.env()
  assign("answer", 0, envir=myenv)

  # what to do when finished: return the count.
  myexit <- function(e) { return(get("answer", envir=e)) }

  # button clicks
  # decrement the count
  Cminus <- function(e, xy) {
   ans <- get("answer", envir=e)
   assign("answer", ans - 1, envir=e)
   return(TRUE)
 }
 # display the count (clicking does nothing)
 Cvalue <- function(...) { TRUE }
 # increment the count
 Cplus <- function(e, xy) {
  ans <- get("answer", envir=e)
  assign("answer", ans + 1, envir=e)
  return(TRUE)
 }
 # quit button
 Cdone <- function(e, xy) { return(FALSE) }

 myclicks <- list("-"=Cminus,
                  value=Cvalue,
                  "+"=Cplus,
                  done=Cdone)

 # redraw the button that displays the current value of the count
 Rvalue <- function(button, nam, e) {
   plot(button, add=TRUE)
   ans <- get("answer", envir=e)
   text(centroid.owin(button), labels=ans)
   return(TRUE)
 }
```

```
# make the panel
P <- simplepanel("Counter",
                 B=myB, boxes=myboxes,
                 clicks=myclicks,
                 redraws = list(NULL, Rvalue, NULL, NULL),
                 exit=myexit, env=myenv)
P

run.simplepanel(P)
}
```

`runifrect` *Generate N Uniform Random Points in a Rectangle*

Description

Generate a random point pattern, containing n independent uniform random points, inside a specified rectangle.

Usage

```
runifrect(n, win = owin(c(0, 1), c(0, 1)), nsim = 1, drop = TRUE)
```

Arguments

`n`	Number of points.
`win`	Rectangular window in which to simulate the pattern. An object of class `"owin"` or something acceptable to `as.owin`, which must specify a rectangle.
`nsim`	Number of simulated realisations to be generated.
`drop`	Logical. If `nsim=1` and `drop=TRUE` (the default), the result will be a point pattern, rather than a list containing a point pattern.

Details

This function is a slightly faster version of `runifpoint` for the special case where the window is a rectangle.

The function generates `n` independent random points, uniformly distributed in the window `win`, by assigning uniform random values to the cartesian coordinates.

For normal usage we recommend `runifpoint` because it is more flexible. However, `runifrect` is slightly faster (when the window is a rectangle), and may be preferable in very computationally-demanding tasks.

Value

A point pattern (an object of class `"ppp"`) if `nsim=1` and `drop=TRUE`, otherwise a list of point patterns.

Author(s)

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

See Also

ppp.object, owin.object, runifpoint, rpoispp, rpoint

Examples

```
# 42 random points in the unit square
pp <- runifrect(42)
```

---

scalardilate *Apply Scalar Dilation*

Description

Applies 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 'im'
scalardilate(X, f, ..., origin=NULL)

  ## S3 method for class 'owin'
scalardilate(X, f, ..., origin=NULL)

  ## S3 method for class 'ppp'
scalardilate(X, f, ..., origin=NULL)

  ## S3 method for class 'psp'
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"), a window (object of class "owin"), a pixel image (class "im") and so on.
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", "left", "right", "top", "bottom", "topleft", "bottomleft", "topright" or "bottomright" (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"), point patterns (class "ppp"), pixel images (class "im"), line segment patterns (class "psp") 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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

affine, shift

Examples

```
plot(letterR)
plot(scalardilate(letterR, 0.7, origin="left"), 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, xrange=range(x))
  ## Default S3 method:
scaletointerval(x, from=0, to=1, xrange=range(x))
  ## S3 method for class 'im'
scaletointerval(x, from=0, to=1, xrange=range(x))
```

Arguments

x	Data to be rescaled.
from, to	Lower and upper endpoints of the interval to which the values of x should be rescaled.
xrange	Optional range of values of x that should be mapped to the new interval.

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

scale

Examples

```
X <- as.im(function(x,y) {x+y+3}, unit.square())
summary(X)
Y <- scaletointerval(X)
summary(Y)
```

---

scanpp *Read Point Pattern From Data File*

Description

Reads a point pattern dataset from a text file.

Usage

```
scanpp(filename, window, header=TRUE, dir="", factor.marks=NULL, ...)
```

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.
factor.marks	Logical vector (or NULL) indicating whether marks are to be interpreted as factors. Defaults to NULL which means that strings will be interpreted as factors while numeric variables will not. See details.
...	Ignored.

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 more 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. Remaining columns, if present, will be interpreted as containing the marks for these points.

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 columns 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.

If a logical vector is provided for factor.marks the length should equal the number of mark columns (a shorter factor.marks is recycled to this length). This vector is then used to determine which mark columns should be interpreted as factors. Note: Strings will not be interpreted as factors if the corresponding entry in factor.marks is FALSE.

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

See Also

ppp.object, ppp, as.ppp, ripras

Examples

```
##  files installed with spatstat, for demonstration
d <- system.file("rawdata", "finpines", package="spatstat.data")
if(nzchar(d)) {
  W <- owin(c(-5,5), c(-8,2))
  X <- scanpp("finpines.txt", dir=d, window=W)
  print(X)
}
d <- system.file("rawdata", "amacrine", package="spatstat.data")
if(nzchar(d)) {
  W <- owin(c(0, 1060/662), c(0, 1))
  Y <- scanpp("amacrine.txt", dir=d, window=W, factor.marks=TRUE)
  print(Y)
}
```

---

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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")
```

`selfcut.psp` *Cut Line Segments Where They Intersect*

Description

Finds any crossing points between the line segments in a line segment pattern, and cuts the segments into pieces at these crossing-points.

Usage

```
selfcut.psp(A, ..., eps)
```

Arguments

`A`	Line segment pattern (object of class `"psp"`).
`eps`	Optional. Smallest permissible length of the resulting line segments. There is a sensible default.
`...`	Ignored.

Details

This function finds any crossing points between different line segments in the line segment pattern `A`, and cuts the line segments into pieces at these intersection points.

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

Another line segment pattern (object of class `"psp"`) in the same window as `A` with the same kind of marks as `A`.

The result also has an attribute `"camefrom"` indicating the provenance of each segment in the result. For example `camefrom[3]=2` means that the third segment in the result is a piece of the second segment of `A`.

Author(s)

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

See Also

selfcrossing.psp

Examples

```
X <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
Y <- selfcut.psp(X)
n <- nsegments(Y)
plot(Y %mark% factor(sample(seq_len(n), n, replace=TRUE)))
```

---

sessionLibs *Print Names and Version Numbers of Libraries Loaded*

Description

Prints the names and version numbers of libraries currently loaded by the user.

Usage

```
sessionLibs()
```

Details

This function prints a list of the libraries loaded by the user in the current session, giving just their name and version number. It obtains this information from sessionInfo.

This function is not needed in an interactive R session because the package startup messages will usually provide this information.

Its main use is in an Sweave script, where it is needed because the package startup messages are not printed.

Value

Null.

Author(s)

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

See Also

sessionInfo

Examples

```
sessionLibs()
```

setcov *Set Covariance of a Window*

Description

Computes the set covariance function of a window.

Usage

```
setcov(W, V=W, ...)
```

Arguments

W	A window (object of class "owin".
V	Optional. Another window.
...	Optional arguments passed to as.mask 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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

imcov, owin, as.owin, erosion

Examples

```
w <- owin(c(0,1),c(0,1))
v <- setcov(w)
plot(v)
```

`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

`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"`).
`...`	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@curtin.edu.au>`

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

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

X	Pixel image (object of class "im").
vec	Vector of length 2 representing a translation.
...	Ignored
origin	Location that will be shifted to the origin. Either a numeric vector of length 2 giving the location, or a point pattern containing only one point, or a list with two entries named x and y, or one of the character strings "centroid", "midpoint", "left", "right", "top", "bottom", "topleft", "bottomleft", "topright" or "bottomright" (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, the argument vec will be ignored; instead the shift will be performed so that the specified geometric location is shifted to the coordinate origin $(0, 0)$. The argument origin should be either a numeric vector of length 2 giving the spatial coordinates of a location, or one of the character strings "centroid", "midpoint", "left", "right", "top", "bottom", "topleft", "bottomleft", "topright" or "bottomright" (partially matched). 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, and so on.

Value

Another pixel image (of class "im") representing the result of applying the vector shift.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

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="c")
```

---

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	Location that will be shifted to the origin. Either a numeric vector of length 2 giving the location, or a point pattern containing only one point, or a list with two entries named x and y, or one of the character strings "centroid", "midpoint", "left", "right", "top", "bottom", "topleft", "bottomleft", "topright" or "bottomright" (partially matched).

Details

The window is translated by the vector vec. This is a method for the generic function shift.

If origin is given, the argument vec will be ignored; instead the shift will be performed so that the specified geometric location is shifted to the coordinate origin $(0, 0)$. The argument origin should be either a numeric vector of length 2 giving the spatial coordinates of a location, or one of the character strings "centroid", "midpoint", "left", "right", "top", "bottom", "topleft", "bottomleft", "topright" or "bottomright" (partially matched). 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, and so on.

Value

Another window (of class "owin") representing the result of applying the vector shift.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

shift, shift.ppp, periodify, rotate, affine, centroid.owin

Examples

```
W <- owin(c(0,1),c(0,1))
X <- shift(W, c(2,3))
# plot(W)
# no discernible difference except coordinates are different
shift(W, origin="top")
```

---

shift.ppp *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	Location that will be shifted to the origin. Either a numeric vector of length 2 giving the location, or a point pattern containing only one point, or a list with two entries named x and y, or one of the character strings "centroid", "midpoint", "left", "right", "top", "bottom", "topleft", "bottomleft", "topright" or "bottomright" (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, the argument `vec` will be ignored; instead the shift will be performed so that the specified geometric location is shifted to the coordinate origin $(0, 0)$. The argument `origin` should be either a numeric vector of length 2 giving the spatial coordinates of a location, or one of the character strings `"centroid"`, `"midpoint"`, `"left"`, `"right"`, `"top"`, `"bottom"`, `"topleft"`, `"bottomleft"`, `"topright"` or `"bottomright"` (partially matched). 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, and so on.

Value

Another point pattern (of class `"ppp"`) representing the result of applying the vector shift.

Author(s)

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

See Also

`shift`, `shift.owin`, `periodify`, `rotate`, `affine`

Examples

```
X <- shift(cells, c(2,3))
# plot(X)
# no discernible difference except coordinates are different
plot(cells, pch=16)
plot(shift(cells, c(0.03,0.03)), add=TRUE)

shift(cells, origin="mid")
```

`shift.ppx` *Apply Vector Translation To Box Or Point Pattern In Arbitrary Dimension*

Description

Applies a vector shift to a box or point pattern in arbitrary dimension (object of class `"boxx"` or `"ppx"`).

Usage

```
 ## S3 method for class 'boxx'
shift(X, vec= 0, ...)
 ## S3 method for class 'ppx'
shift(X, vec = 0, ..., spatial = TRUE, temporal = TRUE, local = TRUE)
```

Arguments

X	Box or point pattern in arbitrary dimension (object of class "boxx" or "ppx").
vec	Either a single numeric or a vector of the same length as the dimension of the spatial and/or temporal and/or local domain.
...	Ignored
spatial, temporal, local	Logical to indicate whether or not to shift this type of coordinates for the ppx method.

Details

This is a method for the generic function shift.

Value

For shift.boxx, another "boxx" object and for shift.ppx another "ppx" object. In both cases the new object represents the result of applying the vector shift.

Author(s)

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

See Also

shift, boxx, ppx

Examples

```
vec <- c(2,3)
dom <- boxx(c(0,1), c(0,1))
X <- ppx(coords(cells), domain = dom)
shift(dom, vec)
Xs <- shift(X, vec)
Xs
head(coords(X), n = 3)
head(coords(Xs), n = 3)
```

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	Location that will be shifted to the origin. Either a numeric vector of length 2 giving the location, or a point pattern containing only one point, or a list with two entries named x and y, or one of the character strings "centroid", "midpoint", "left", "right", "top", "bottom", "topleft", "bottomleft", "topright" or "bottomright" (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, the argument vec will be ignored; instead the shift will be performed so that the specified geometric location is shifted to the coordinate origin $(0, 0)$. The argument origin should be either a numeric vector of length 2 giving the spatial coordinates of a location, or one of the character strings "centroid", "midpoint", "left", "right", "top", "bottom", "topleft", "bottomleft", "topright" or "bottomright" (partially matched). 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, and so on.

Value

Another line segment pattern (of class "psp") representing the result of applying the vector shift.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

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")
```

---

sidelengths.owin *Side Lengths of Enclosing Rectangle of a Window*

Description

Computes the side lengths of the (enclosing rectangle of) a window.

Usage

```
 ## S3 method for class 'owin'
sidelengths(x)

 ## S3 method for class 'owin'
shortside(x)
```

Arguments

x	A window whose side lengths will be computed. Object of class "owin".

Details

The functions shortside and sidelengths are generic. The functions documented here are the methods for the class "owin".

sidelengths.owin computes the side-lengths of the enclosing rectangle of the window x.

For safety, both functions give a warning if the window is not a rectangle. To suppress the warning, first convert the window to a rectangle using as.rectangle.

shortside.owin computes the minimum of the two side-lengths.

Value

For sidelengths.owin, a numeric vector of length 2 giving the side-lengths (x then y) of the enclosing rectangle. For shortside.owin, a numeric value.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

shortside, sidelengths for the generic functions.

area.owin, diameter.owin, perimeter for other geometric calculations on "owin" objects.

owin, as.owin.

Examples

```
w <- owin(c(0,2),c(-1,3))
sidelengths(w)
shortside(as.rectangle(letterR))
```

---

simplepanel *Simple Point-and-Click Interface Panels*

Description

These functions enable the user to create a simple, robust, point-and-click interface to any R code.

Usage

```
simplepanel(title, B, boxes, clicks,
   redraws=NULL, exit = NULL, env)

grow.simplepanel(P, side = c("right", "left", "top", "bottom"),
   len = NULL, new.clicks, new.redraws=NULL, ..., aspect)
```

Arguments

title	Character string giving the title of the interface panel.
B	Bounding box of the panel coordinates. A rectangular window (object of class "owin")
boxes	A list of rectangular windows (objects of class "owin") specifying the placement of the buttons and other interactive components of the panel.
clicks	A list of R functions, of the same length as boxes, specifying the operations to be performed when each button is clicked. Entries can also be NULL indicating that no action should occur. See Details.
redraws	Optional list of R functions, of the same length as boxes, specifying how to redraw each button. Entries can also be NULL indicating a simple default. See Details.

exit	An R function specifying actions to be taken when the interactive panel terminates.
env	An environment that will be passed as an argument to all the functions in clicks, redraws and exit.
P	An existing interaction panel (object of class "simplepanel").
side	Character string identifying which side of the panel P should be grown to accommodate the new buttons.
len	Optional. Thickness of the new panel area that should be grown to accommodate the new buttons. A single number in the same units as the coordinate system of P.
new.clicks	List of R functions defining the operations to be performed when each of the new buttons is clicked.
new.redraws	Optional. List of R functions, of the same length as new.clicks, defining how to redraw each of the new buttons.
...	Arguments passed to layout.boxes to determine the layout of the new buttons.
aspect	Optional. Aspect ratio (height/width) of the new buttons.

Details

These functions enable the user to create a simple, robust, point-and-click interface to any R code.

The functions simplepanel and grow.simplepanel create an object of class "simplepanel". Such an object defines the graphics to be displayed and the actions to be performed when the user interacts with the panel.

The panel is activated by calling run.simplepanel.

The function simplepanel creates a panel object from basic data. The function grow.simplepanel modifies an existing panel object P by growing an additional row or column of buttons.

For simplepanel,

- The spatial layout of the panel is determined by the rectangles B and boxes.
- The argument clicks must be a list of functions specifying the action to be taken when each button is clicked (or NULL to indicate that no action should be taken). The list entries should have names (but there are sensible defaults). Each function should be of the form function(env, xy) where env is an environment that may contain shared data, and xy gives the coordinates of the mouse click, in the format list(x, y). The function returns TRUE if the panel should continue running, and FALSE if the panel should terminate.
- The argument redraws, if given, must be a list of functions specifying the action to be taken when each button is to be redrawn. Each function should be of the form function(button, name, env) where button is a rectangle specifying the location of the button in the current coordinate system; name is a character string giving the name of the button; and env is the environment that may contain shared data. The function returns TRUE if the panel should continue running, and FALSE if the panel should terminate. If redraws is not given (or if one of the entries in redraws is NULL), the default action is to draw a pink rectangle showing the button position, draw the name of the button in the middle of this rectangle, and return TRUE.

- The argument exit, if given, must be a function specifying the action to be taken when the panel terminates. (Termination occurs when one of the clicks functions returns FALSE). The exit function should be of the form function(env) where env is the environment that may contain shared data. Its return value will be used as the return value of run.simplepanel.
- The argument env should be an R environment. The panel buttons will have access to this environment, and will be able to read and write data in it. This mechanism is used to exchange data between the panel and other R code.

For grow.simplepanel,

- the spatial layout of the new boxes is determined by the arguments side, len, aspect and by the additional ... arguments passed to layout.boxes.
- the argument new.clicks should have the same format as clicks. It implicitly specifies the number of new buttons to be added, and the actions to be performed when they are clicked.
- the optional argument new.redraws, if given, should have the same format as redraws. It specifies the actions to be performed when the new buttons are clicked.

Value

An object of class "simplepanel".

Author(s)

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

See Also

run.simplepanel, layout.boxes

Examples

```
# make boxes (alternatively use layout.boxes())
Bminus <- square(1)
Bvalue <- shift(Bminus, c(1.2, 0))
Bplus <- shift(Bvalue, c(1.2, 0))
Bdone <- shift(Bplus, c(1.2, 0))
myboxes <- list(Bminus, Bvalue, Bplus, Bdone)
myB <- do.call(boundingbox,myboxes)

# make environment containing an integer count
myenv <- new.env()
assign("answer", 0, envir=myenv)

# what to do when finished: return the count.
myexit <- function(e) { return(get("answer", envir=e)) }

# button clicks
# decrement the count
Cminus <- function(e, xy) {
  ans <- get("answer", envir=e)
```

```
  assign("answer", ans - 1, envir=e)
  return(TRUE)
}
# display the count (clicking does nothing)
Cvalue <- function(...) { TRUE }
# increment the count
Cplus <- function(e, xy) {
  ans <- get("answer", envir=e)
  assign("answer", ans + 1, envir=e)
  return(TRUE)
}
# 'Clear' button
Cclear <- function(e, xy) {
  assign("answer", 0, envir=e)
  return(TRUE)
}
# quit button
Cdone <- function(e, xy) { return(FALSE) }

myclicks <- list("-"=Cminus,
                 value=Cvalue,
                 "+"=Cplus,
                 done=Cdone)

# redraw the button that displays the current value of the count
Rvalue <- function(button, nam, e) {
   plot(button, add=TRUE)
   ans <- get("answer", envir=e)
   text(centroid.owin(button), labels=ans)
   return(TRUE)
}

# make the panel
P <- simplepanel("Counter",
                 B=myB, boxes=myboxes,
                 clicks=myclicks,
                 redraws = list(NULL, Rvalue, NULL, NULL),
                 exit=myexit, env=myenv)
# print it
P
# show what it looks like
redraw.simplepanel(P)

# ( type run.simplepanel(P) to run the panel interactively )

# add another button to right
Pplus <- grow.simplepanel(P, "right", new.clicks=list(clear=Cclear))
```

---

simplify.owin *Approximate a Window by a Simpler Polygon*

Description

Given a window, this function finds a simpler polygon that approximates it.

Usage

```
simplify.owin(W, dmin)
```

Arguments

W	The window which is to be simplied. An object of class "owin".
dmin	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"). If W is a rectangle, it is returned without alteration. If W is a binary mask, it is first converted to a polygonal window (with many small vertical and horizontal edges), then simplified.

Value

Another window (object of class "owin") of type "polygonal".

Author(s)

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

See Also

fillholes.owin, owin

Examples

```
plot(letterR, col="red")
plot(simplify.owin(letterR, 0.3), col="blue", add=TRUE)

W <- Window(chorley)
plot(W)
WS <- simplify.owin(W, 2)
plot(WS, add=TRUE, border="green")
points(vertices(WS))
```

solapply *Apply a Function Over a List and Obtain a List of Objects*

Description

Applies the function FUN to each element of the list X, and returns the result as a list of class "solist" or "anylist" as appropriate.

Usage

```
anylapply(X, FUN, ...)

solapply(X, FUN, ..., check = TRUE, promote = TRUE, demote = FALSE)
```

Arguments

X	A list.
FUN	Function to be applied to each element of X.
...	Additional arguments to FUN.
check, promote, demote	Arguments passed to solist which determine how to handle different classes of objects.

Details

These convenience functions are similar to lapply except that they return a list of class "solist" or "anylist".

In both functions, the result is computed by lapply(X, FUN, ...).

In anylapply the result is converted to a list of class "anylist" and returned.

In solapply the result is converted to a list of class "solist" **if possible**, using as.solist. If this is not possible, then the behaviour depends on the argument demote. If demote=TRUE the result will be returned as a list of class "anylist". If demote=FALSE (the default), an error occurs.

Value

A list, usually of class "solist".

Author(s)

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

See Also

solist, anylist.

Examples

```
solapply(waterstriders, distmap)
```

`solist` *List of Two-Dimensional Spatial Objects*

Description

Make a list of two-dimensional spatial objects.

Usage

```
solist(..., check=TRUE, promote=TRUE, demote=FALSE, .NameBase)
```

Arguments

`...`	Any number of objects, each representing a two-dimensional spatial dataset.
`check`	Logical value. If `TRUE`, check that each of the objects is a 2D spatial object.
`promote`	Logical value. If `TRUE`, test whether all objects belong to the *same* class, and if so, promote the list of objects to the appropriate class of list.
`demote`	Logical value determining what should happen if any of the objects is not a 2D spatial object: if `demote=FALSE` (the default), a fatal error occurs; if `demote=TRUE`, a list of class `"anylist"` is returned.
`.NameBase`	Optional. Character string. If the `...` arguments have no names, then the entries of the resulting list will be given names that start with `.NameBase`.

Details

This command creates an object of class `"solist"` (spatial object list) which represents a list of two-dimensional spatial datasets. The datasets do not necessarily belong to the same class.

Typically the intention is that the datasets in the list should be treated in the same way, for example, they should be plotted side-by-side. The **spatstat** package provides a plotting function, `plot.solist`, and many other functions for this class.

In the **spatstat** package, various functions produce an object of class `"solist"`. For example, when a point pattern is split into several point patterns by `split.ppp`, or an image is split into several images by `split.im`, the result is of class `"solist"`.

If `check=TRUE` then the code will check whether all objects in `...` belong to the classes of two-dimensional spatial objects defined in the **spatstat** package. They do not have to belong to the *same* class. Set `check=FALSE` for efficiency, but only if you are sure that all the objects are valid.

If some of the objects in `...` are not two-dimensional spatial objects, the action taken depends on the argument `demote`. If `demote=TRUE`, the result will belong to the more general class `"anylist"` instead of `"solist"`. If `demote=FALSE` (the default), an error occurs.

If `promote=TRUE` then the code will check whether all the objects `...` belong to the same class. If they are all point patterns (class `"ppp"`), the result will also belong to the class `"ppplist"`. If they are all pixel images (class `"im"`), the result will also belong to the class `"imlist"`.

Use `as.solist` to convert a list to a `"solist"`.

Value

A list, usually belonging to the class "solist".

Author(s)

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

See Also

as.solist, anylist, solapply

Examples

```
solist(cells, distmap(cells), quadratcount(cells))
solist(cells, japanesepines, redwood, .NameBase="Pattern")
```

---

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 solutionset(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.

If X and Y are two pixel images, solutionset(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, ... can be any logical expression involving pixel images.

The code first tries to evaluate the expression using eval.im. This is successful if the expression involves only (a) the *names* of pixel images, (b) scalar constants, and (c) functions which are vectorised. There must be at least one pixel image in the expression. The expression expr must be vectorised. See the Examples.

If this is unsuccessful, the code then tries to evaluate the expression using pixel arithmetic. This is successful if all the arithmetic operations in the expression are listed in Math.im.

Value

A spatial window (object of class "owin", see owin.object).

Author(s)

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

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(solutionset(X < Y))

solutionset(distmap(cells) < 0.05)
```

---

spatdim *Spatial Dimension of a Dataset*

Description

Extracts the spatial dimension of an object in the **spatstat** package.

Usage

```
spatdim(X, intrinsic=FALSE)
```

Arguments

X	Object belonging to any class defined in the **spatstat** package.
intrinsic	Logical value indicating whether to return the number of intrinsic dimensions. See Details.

Details

This function returns the number of spatial coordinate dimensions of the dataset X. The results for some of the more common types of objects are as follows:

object class	dimension
"ppp"	2
"lpp"	2
"pp3"	3
"ppx"	number of *spatial* dimensions
"owin"	2
"psp"	2
"ppm"	2

Note that time dimensions are not counted.

Some spatial objects are lower-dimensional subsets of the space in which they live. This lower number of dimensions is returned if intrinsic=TRUE. For example, a dataset on a linear network (an object X of class "linnet", "lpp", "linim", "linfun" or "lintess") returns spatdim(X) = 2 but spatdim(X, intrinsic=TRUE) = 1.

If X is not a recognised spatial object, the result is NA.

Value

An integer, or NA.

Author(s)

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

Examples

```
spatdim(lansing)
A <- osteo$pts[[1]]
spatdim(A)
spatdim(domain(A))
```

---

spatstat.options *Internal Options in Spatstat Package*

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 of interest to the user are:

checkpolygons Logical flag indicating whether the functions `owin` and `as.owin` should apply very strict checks on the validity of polygon data. These strict checks are no longer necessary, and the default is `checkpolygons=FALSE`. See also `fixpolygons` below.

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`.

dpp.maxmatrix Integer specifying the maximum size of matrices generated by `dppeigen`. Defaults to `2^24`.

eroded.intensity Logical flag affecting the behaviour of the score and pseudo-score residual functions `Gcom`, `Gres` `Kcom`, `Kres`, `psstA`, `psstG`, `psst`. The flag indicates whether to compute intensity estimates on an eroded window (`eroded.intensity=TRUE`) or on the original data window (`eroded.intensity=FALSE`, the default).

expand The default expansion factor (area inflation factor) for expansion of the simulation window in `rmh` (see `rmhcontrol`). Initialised to `2`.

expand.polynom Logical. Whether expressions involving `polynom` in a model formula should be expanded, so that `polynom(x,2)` is replaced by `x + I(x^2)` and so on. Initialised to `TRUE`.

fastpois Logical. Whether to use a fast algorithm (introduced in **spatstat 1.42-3**) for simulating the Poisson point process in `rpoispp` when the argument `lambda` is a pixel image. Initialised to `TRUE`. Should be set to `FALSE` if needed to guarantee repeatability of results computed using earlier versions of **spatstat**.

fastthin Logical. Whether to use a fast C language algorithm (introduced in **spatstat 1.42-3**) for random thinning in `rthin` when the argument `P` is a single number. Initialised to `TRUE`. Should be set to `FALSE` if needed to guarantee repeatability of results computed using earlier versions of **spatstat**.

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`.

fixpolygons Logical flag indicating whether the functions `owin` and `as.owin` should repair errors in polygon data. For example, self-intersecting polygons and overlapping polygons will be repaired. The default is `fixpolygons=TRUE`.

fftw Logical value indicating whether the two-dimensional Fast Fourier Transform should be computed using the package **fftwtools**, instead of the `fft` function in the **stats** package. This affects the speed of `density.ppp`, `density.psp`, `blur` `setcov` and `Smooth.ppp`.

gpclib Defunct. This parameter was used to permit or forbid the use of the package **gpclib**, because of its restricted software licence. This package is no longer needed.

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`.

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.

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`.

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.

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`.

monochrome Logical flag indicating whether graphics should be plotted in grey scale (`monochrome=TRUE`) or in colour (`monochrome=FALSE`, the default).

n.bandwidth Integer. Number of trial values of smoothing bandwidth to use for cross-validation in `bw.relrisk` and similar functions.

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.

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.

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`.

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.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.

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.pp3 List of arguments controlling the plotting of three-dimensional point patterns by `plot.pp3`.

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.

progress Character string determining the style of progress reports printed by `progressreport`. Either `"tty"`, `"tk"` or `"txtbar"`. For explanation of these options, see `progressreport`.

project.fast Logical. If `TRUE`, the algorithm of `project.ppm` will be accelerated using a shorcut. Initialised to `FALSE`.

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.

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`.

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`.

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.

terse Integer between 0 and 4. The level of terseness (brevity) in printed output from many functions in **spatstat**. Higher values mean shorter output. A rough guide is the following:

0	Full output
1	Avoid wasteful output
2	Remove space between paragraphs
3	Suppress extras such as standard errors
4	Compress text, suppress internal warnings

The value of `terse` is initialised to 0.

transparent Logical value indicating whether default colour maps are allowed to include semi-transparent colours, where possible. Default is `TRUE`. Currently this only affects `plot.ppp`.

units.paren The kind of parenthesis which encloses the text that explains a `unitname`. This text is seen in the text output of functions like `print.ppp` and in the graphics generated by `plot.fv`. The value should be one of the character strings `'('`, `'['`, `'{'` or `''`. The default is `'('`.

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.

Internal parameters

The following parameters may also be specified to `spatstat.options` but are intended for software development or testing purposes.

closepairs.newcode Logical. Whether to use new version of the code for `closepairs`. Initialised to `TRUE`.

crossing.psp.useCall Logical. Whether to use new version of the code for `crossing.psp`. Initialised to `TRUE`.

crosspairs.newcode Logical. Whether to use new version of the code for `crosspairs`. Initialised to `TRUE`.

densityC Logical. Indicates whether to use accelerated C code (`densityC=TRUE`) or interpreted R code (`densityC=FALSE`) to evaluate `density.ppp(X, at="points")`. Initialised to `TRUE`.

exactdt.checks.data Logical. Do not change this value, unless you are Adrian Baddeley `<Adrian.Baddeley@curtin.edu.a`

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'`.

old.morpho.psp Logical. Whether to use old R code for morphological operations. Initialise to `FALSE`.

selfcrossing.psp.useCall Logical. Whether to use new version of the code for `selfcrossing.psp`. Initialised to `TRUE`.

use.Krect Logical. Whether to use specialised code for the K-function in a rectangular window. Initialised to `TRUE`.

Author(s)

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

See Also

`options`

Examples

```
# save current values whatever they are
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 to the state at the beginning of these examples
spatstat.options(oldopt)
```

```
# revert to 'factory defaults'
reset.spatstat.options()
```

`split.hyperframe` *Divide Hyperframe Into Subsets and Reassemble*

Description

`split` divides the data `x` into subsets defined by `f`. The replacement form replaces values corresponding to such a division.

Usage

```
  ## S3 method for class 'hyperframe'
split(x, f, drop = FALSE, ...)

  ## S3 replacement method for class 'hyperframe'
split(x, f, drop = FALSE, ...) <- value
```

Arguments

`x`	Hyperframe (object of class `"hyperframe"`).
`f`	a `factor` in the sense that `as.factor(f)` defines the grouping, or a list of such factors in which case their interaction is used for the grouping.
`drop`	logical value, indicating whether levels that do not occur should be dropped from the result.
`value`	a list of hyperframes which arose (or could have arisen) from the command `split(x,f,drop=drop)`.
`...`	Ignored.

Details

These are methods for the generic functions `split` and `split<-` for hyperframes (objects of class `"hyperframe"`).

A hyperframe is like a data frame, except that its entries can be objects of any kind. The behaviour of these methods is analogous to the corresponding methods for data frames.

Value

The value returned from `split.hyperframe` is a list of hyperframe containing the values for the groups. The components of the list are named by the levels of `f` (after converting to a factor, or if already a factor and `drop = TRUE`, dropping unused levels).

The replacement method `split<-.hyperframe` returns a new hyperframe `x` for which `split(x,f)` equals `value`.

Author(s)

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

, Rolf Turner <rolfturner@posteo.net>

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

See Also

hyperframe, [.hyperframe

Examples

```
split(pyramidal, pyramidal$group)
```

---

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

x	Pixel image (object of class "im").
f	Splitting criterion. Either a tessellation (object of class "tess") or a pixel image with factor values.
...	Ignored.
drop	Logical value determining whether each subset should be returned as a pixel images (drop=FALSE) or as a one-dimensional vector of pixel values (drop=TRUE).

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 "solist" so that it can be plotted immediately.

If drop=TRUE, a list of numeric vectors.

Author(s)

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

See Also

by.im, tess, im

Examples

```
W <- square(1)
X <- as.im(function(x,y){sqrt(x^2+y^2)}, W)
Y <- dirichlet(runifrect(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, reduce=FALSE, ...)
  ## S3 replacement method for class 'ppp'
split(x, f = marks(x), drop=FALSE, un=NULL, ...) <- value
```

Arguments

x	A two-dimensional point pattern. An object of class "ppp".
f	Data determining the grouping. Either a factor, a logical vector, a pixel image with factor values, a tessellation, a window, or the name of one of the columns of marks.
drop	Logical. Determines whether empty groups will be deleted.

un	Logical. Determines whether the resulting subpatterns will be unmarked (i.e. whether marks will be removed from the points in each subpattern).
reduce	Logical. Determines whether to delete the column of marks used to split the pattern, when the marks are a data frame.
...	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 ith 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 window (object of class "owin"). Each point of x will be classified according to whether it falls inside or outside this window.
- the character string "marks", if marks(x) is a factor.
- 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.

If the marks of x are a data frame, then split(x, reduce=TRUE) will discard only the column of marks that was used to split the pattern. This applies only when the argument f is missing.

The result of split.ppp has class "splitppp" and can be plotted using plot.splitppp.

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, but this typically changes the ordering of the data.

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 "splitppp".

The assignment form split<-.ppp returns the updated point pattern x.

Author(s)

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

See Also

cut.ppp, plot.splitppp, superimpose, im, tess, ppp.object

Examples

```
# (1) Splitting by marks

# Multitype point pattern: separate into types
 u <- split(amacrine)

# plot them
 plot(split(amacrine))

# the following are equivalent:
 amon <- split(amacrine)$on
 amon <- unmark(amacrine[amacrine$marks == "on"])
 amon <- subset(amacrine, marks == "on", -marks)

# the following are equivalent:
 amon <- split(amacrine, un=FALSE)$on
 amon <- amacrine[amacrine$marks == "on"]

# Scramble the locations of the 'on' cells
 X <- amacrine
 u <- split(X)
 u$on <- runifrect(npoints(amon), Window(amon))
 split(X) <- u

# Point pattern with continuous marks
 trees <- longleaf

 # cut the range of tree diameters into three intervals
```

```
 # using cut.ppp
 long3 <- cut(trees, breaks=3)
 # now split them
 long3split <- split(long3)

# (2) Splitting by a factor

# Unmarked point pattern
  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)


# (4) how to apply an operation to selected points:
#  split into components, transform desired component, then un-split
#  e.g. apply random jitter to 'on' points only
  X <- amacrine
  Y <- split(X)
  Y$on <- rjitter(Y$on, 0.1)
  split(X) <- Y
```

---

split.ppx *Divide Multidimensional Point Pattern into Sub-patterns*

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

x	A multi-dimensional point pattern. An object of class "ppx".
f	Data determining the grouping. Either a factor, a logical vector, or the name of one of the columns of marks.
drop	Logical. Determines whether empty groups will be deleted.
un	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.

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 ith point of x will be placed in the sub-pattern split.ppx(x)$l where l = f[i].
- the character string "marks", if marks(x) is a factor.
- a character string, matching the name of one of the columns of marks, if marks(x) is a data frame or hyperframe. 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 "anylist". 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 "anylist".

Author(s)

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

See Also

ppx, plot.anylist

Examples

```
df <- data.frame(x=runif(4),y=runif(4),t=runif(4),
                 age=factor(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

x	Vector of x coordinates of data points, or a list with components x and y, or a point pattern (an object of class ppp).
y	Vector of y coordinates of data points. Ignored unless x is a vector.
nrad	Number of radii emanating from each data point.
nper	Number of dummy points per radius.
fctr	Scale factor. Length of largest spoke radius is fctr * M where M is the mean nearest neighbour distance for the data points.
Mdefault	Value of M to be used if x 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 * 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 * 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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

quad.object, quadscheme, inside.owin, gridcentres, stratrand

Examples

```
dat <- runifrect(10)
dum <- spokes(dat$x, dat$y, 5, 3, 0.7)
plot(dum)
Q <- quadscheme(dat, dum, method="dirichlet")
plot(Q, tiles=TRUE)
```

---

square *Square Window*

Description

Creates a square window

Usage

```
square(r=1, unitname=NULL)
unit.square()
```

Arguments

r	Numeric. The side length of the square, or a vector giving the minimum and maximum coordinate values.
unitname	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

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 for convenience, and 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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

owin.object, owin

Examples

```
W <- square(10)
W <- square(c(-1,1))
```

---

stratrand *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 in each tile.

Usage

```
stratrand(window, nx, ny, k = 1)
```

Arguments

window	A window. An object of class owin, or data in any format acceptable to as.owin().
nx	Number of tiles in each row.
ny	Number of tiles in each column.
k	Number of random points to generate in each tile.

Details

The bounding rectangle of window 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.

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) as well as in simulating random point patterns.

Value

A list with two components x and y, which are numeric vectors giving the coordinates of the random points.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

quad.object, quadscheme, inside.owin, gridcentres

Examples

```
w <- unit.square()
xy <- stratrand(w, 10, 10)
# plot(w)
# points(xy)

# polygonal boundary
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 <- stratrand(w, 10, 10, 3)
# plot(w)
# points(xy)

# determine which grid points are inside polygon
ok <- inside.owin(xy$x, xy$y, w)
# plot(w)
# points(xy$x[ok], xy$y[ok])
```

subset.hyperframe *Subset of Hyperframe Satisfying A Condition*

Description

Given a hyperframe, return the subset specified by imposing a condition on each row, and optionally by choosing only some of the columns.

Usage

```
## S3 method for class 'hyperframe'
subset(x, subset, select, ...)
```

Arguments

x	A hyperframe pattern (object of class "hyperframe".
subset	Logical expression indicating which points are to be kept. The expression may involve the names of columns of x and will be evaluated by with.hyperframe.
select	Expression indicating which columns of marks should be kept.
...	Arguments passed to [.hyperframe such as drop and strip.

Details

This is a method for the generic function subset. It extracts the subset of rows of x that satisfy the logical expression subset, and retains only the columns of x that are specified by the expression select. The result is always a hyperframe.

The argument subset determines the subset of rows that will be extracted. It should be a logical expression. It may involve the names of columns of x. The default is to keep all points.

The argument select determines which columns of x will be retained. It should be an expression involving the names of columns (which will be interpreted as integers representing the positions of these columns). For example if there are columns named A to Z, then select=D:F is a valid expression and means that columns D, E and F will be retained. Similarly select=-(A:C) is valid and means that columns A to C will be deleted. The default is to retain all columns.

Setting subset=FALSE will remove all the rows. Setting select=FALSE will remove all the columns.

The result is always a hyperframe.

Value

A hyperframe.

Author(s)

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

, Rolf Turner <rolfturner@posteo.net>

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

See Also

subset, [.hyperframe

Examples

```
a <- subset(flu, virustype=="wt")

aa <- subset(flu, minnndist(pattern) > 10)

aaa <- subset(flu, virustype=="wt", select = -pattern)
```

---

subset.ppp *Subset of Point Pattern Satisfying A Condition*

Description

Given a point pattern, return the subset of points which satisfy a specified condition.

Usage

```
## S3 method for class 'ppp'
subset(x, subset, select, drop=FALSE, ...)

## S3 method for class 'pp3'
subset(x, subset, select, drop=FALSE, ...)


## S3 method for class 'ppx'
subset(x, subset, select, drop=FALSE, ...)
```

Arguments

x	A point pattern (object of class "ppp", "lpp", "pp3" or "ppx").
subset	Logical expression indicating which points are to be kept. The expression may involve the names of spatial coordinates (x, y, etc), the marks, and (if there is more than one column of marks) the names of individual columns of marks. Missing values are taken as false. See Details.
select	Expression indicating which columns of marks should be kept. The *names* of columns of marks can be used in this expression, and will be treated as if they were column indices. See Details.
drop	Logical value indicating whether to remove unused levels of the marks, if the marks are a factor.
...	Ignored.

Details

This is a method for the generic function subset. It extracts the subset of points of x that satisfy the logical expression subset, and retains only the columns of marks that are specified by the expression select. The result is always a point pattern, with the same window as x.

The argument subset determines the subset of points that will be extracted. It should be a logical expression. It may involve the variable names x and y representing the Cartesian coordinates; the names of other spatial coordinates or local coordinates; the name marks representing the marks; and (if there is more than one column of marks) the names of individual columns of marks. The default is to keep all points.

The argument select determines which columns of marks will be retained (if there are several columns of marks). It should be an expression involving the names of columns of marks (which will be interpreted as integers representing the positions of these columns). For example if there are columns of marks named A to Z, then select=D:F is a valid expression and means that columns D, E and F will be retained. Similarly select=-(A:C) is valid and means that columns A to C will be deleted. The default is to retain all columns.

Setting subset=FALSE will produce an empty point pattern (i.e. containing zero points) in the same window as x. Setting select=FALSE or select= -marks will remove all the marks from x.

The argument drop determines whether to remove unused levels of a factor, if the resulting point pattern is multitype (i.e. the marks are a factor) or if the marks are a data frame in which some of the columns are factors.

The result is always a point pattern, of the same class as x. Spatial coordinates (and local coordinates) are always retained. To extract only some columns of marks or coordinates as a data frame, use subset(as.data.frame(x), ...)

Value

A point pattern of the same class as x, in the same spatial window as x. The result is a subset of x, possibly with some columns of marks removed.

Other kinds of subset arguments

Alternatively the argument subset can be any kind of subset index acceptable to [.ppp, [.pp3, [.ppx. This argument selects which points of x will be retained.

Warning: if the argument subset is a window, this is interpreted as specifying the subset of points that fall inside that window, but the resulting point pattern has the same window as the original pattern x.

Author(s)

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

See Also

subset,

[.ppp, [.pp3, [.ppx

Examples

```
plot(subset(cells, x > 0.5))

subset(amacrine, marks == "on")

subset(amacrine, marks == "on", drop=TRUE)

subset(redwood, nndist(redwood) > 0.04)

subset(finpines, select=height)

subset(finpines, diameter > 2, height)

subset(nbfires, year==1999 & ign.src == "campfire",
               select=cause:fnl.size)


if(require(spatstat.random)) {
 a <- subset(rpoispp3(40), z > 0.5)
}
```

`subset.psp` *Subset of Line Segment Satisfying A Condition*

Description

Given a line segment pattern, return the subset of segments which satisfy a specified condition.

Usage

```
## S3 method for class 'psp'
subset(x, subset, select, drop=FALSE, ...)
```

Arguments

`x`	A line segment pattern (object of class `"psp"`).
`subset`	Logical expression indicating which points are to be kept. The expression may involve the names of spatial coordinates of the segment endpoints (`x0`, `y0`, `x1`, `y1`), the `marks`, and (if there is more than one column of marks) the names of individual columns of marks. Missing values are taken as false. See Details.
`select`	Expression indicating which columns of marks should be kept. The *names* of columns of marks can be used in this expression, and will be treated as if they were column indices. See Details.
`drop`	Logical value indicating whether to remove unused levels of the marks, if the marks are a factor.
`...`	Ignored.

Details

This is a method for the generic function subset. It extracts the subset of x consisting of those segments that satisfy the logical expression subset, and retains only the columns of marks that are specified by the expression select. The result is always a line segment pattern, with the same window as x.

The argument subset determines the subset that will be extracted. It should be a logical expression. It may involve the variable names x0, y0, x1, y1 representing the Cartesian coordinates of the segment endpoints; the name marks representing the marks; and (if there is more than one column of marks) the names of individual columns of marks. The default is to keep all segments.

The argument select determines which columns of marks will be retained (if there are several columns of marks). It should be an expression involving the names of columns of marks (which will be interpreted as integers representing the positions of these columns). For example if there are columns of marks named A to Z, then select=D:F is a valid expression and means that columns D, E and F will be retained. Similarly select=-(A:C) is valid and means that columns A to C will be deleted. The default is to retain all columns.

Setting subset=FALSE will produce an empty point pattern (i.e. containing zero points) in the same window as x. Setting select=FALSE or select= -marks will remove all the marks from x.

The argument drop determines whether to remove unused levels of a factor, if the resulting point pattern is multitype (i.e. the marks are a factor) or if the marks are a data frame in which some of the columns are factors.

The result is always a line segment pattern. To extract only some columns of marks as a data frame, use subset(as.data.frame(x), ...)

Value

A line segment pattern (object of class "psp") in the same spatial window as x. The result is a subset of x, possibly with some columns of marks removed.

Author(s)

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

See Also

subset, [.psp.

Examples

```
plot(nbw.seg)
plot(subset(nbw.seg, x0 < 500 & y0 < 800), add=TRUE, lwd=6)
subset(nbw.seg, type == "island")
subset(nbw.seg, type == "coast", select= -type)
subset(nbw.seg, type %in% c("island", "coast"), select= FALSE)
```

summary.anylist *Summary of a List of Things*

Description

Prints a useful summary of each item in a list of things.

Usage

```
 ## S3 method for class 'anylist'
summary(object, ...)
```

Arguments

object	An object of class "anylist".
...	Ignored.

Details

This is a method for the generic function summary.

An object of the class "anylist" is effectively a list of things which are intended to be treated in a similar way. See anylist.

This function extracts a useful summary of each of the items in the list.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

See Also

anylist, summary, plot.anylist

Examples

```
x <- anylist(A=runif(10), B=runif(10), C=runif(10))
summary(x)
```

summary.distfun *Summarizing a Function of Spatial Location*

Description

Prints a useful summary of a function of spatial location.

Usage

```
  ## S3 method for class 'distfun'
summary(object, ...)

  ## S3 method for class 'funxy'
summary(object, ...)
```

Arguments

object	An object of class "distfun" or "funxy" representing a function of spatial coordinates.
...	Arguments passed to as.mask controlling the pixel resolution used to compute the summary.

Details

These are the summary methods for the classes "funxy" and "distfun".

An object of class "funxy" represents a function of spatial location, defined in a particular region of space. This includes objects of the special class "distfun" which represent distance functions.

The summary method computes a summary of the function values. The function is evaluated on a grid of locations using as.im and numerical values at these locations are summarised using summary.im. The pixel resolution for the grid of locations is determined by the arguments ... which are passed to as.mask.

Value

For summary.funxy the result is an object of class "summary.funxy". For summary.distfun the result is an object of class "summary.distfun". There are print methods for these classes.

Author(s)

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

See Also

distfun, funxy

Examples

```
f <- function(x,y) { x^2 + y^2 - 1}
g <- funxy(f, square(2))
summary(g)

summary(distfun(cells))
summary(distfun(cells), dimyx=256)
```

---

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

See Also

mean.im, integral.im, anyNA.im

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

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

See Also

summary, summary.ppp, print.owin

Examples

```
summary(owin())  # the unit square

W <- Window(demopat)  # weird polygonal window
summary(W)           # describes it

summary(as.mask(W))  # demonstrates current pixel resolution
```

---

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

See Also

summary, summary.owin, print.ppp

Examples

```
summary(cells)  # plain vanilla point pattern

# multitype point pattern
woods <- lansing

summary(woods) # tabulates frequencies of each mark

# numeric marks
trees <- longleaf

summary(trees) # prints summary.default(marks(trees))

# weird polygonal window
summary(demopat)  # describes it
```

`summary.psp` *Summary of a Line Segment Pattern Dataset*

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@curtin.edu.au>`

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

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 summary.quad.

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

Examples

```
# make a quadrature scheme
Q <- quadscheme(runifrect(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.solist *Summary of a List of Spatial Objects*

Description

Prints a useful summary of each entry in a list of two-dimensional spatial objects.

Usage

```
 ## S3 method for class 'solist'
summary(object, ...)
```

Arguments

object	An object of class "solist".
...	Ignored.

Details

This is a method for the generic function summary.

An object of the class "solist" is effectively a list of two-dimensional spatial datasets. See solist.

This function extracts a useful summary of each of the datasets.

Author(s)

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

Rolf Turner <rolfturner@posteo.net>

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

See Also

solist, summary, plot.solist

Examples

```
x <- solist(cells, japanesepines, redwood)
summary(x)
```

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

See Also

`summary`, `split`, `split.ppp`

Examples

```
summary(split(amacrine))
```

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)

  ## S3 method for class 'splitppp'
superimpose(..., W=NULL, check=TRUE)

  ## S3 method for class 'ppplist'
superimpose(..., W=NULL, check=TRUE)

  ## Default S3 method:
superimpose(...)
```

Arguments

`...`	Any number of arguments, each of which represents either a point pattern or a line segment pattern or a list of point patterns.
`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. There is also a method for `lpp`, described separately in `superimpose.lpp`.

The dispatch to a method is initially 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. This checks the class of all arguments, and dispatches to the appropriate method. Arguments of class ppplist can be handled.
- 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@curtin.edu.au> and Rolf Turner <rolfturner@posteo.net>.

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 = runifrect(10), b = runifrect(20))

# how to make a 2-type point pattern with types 1 and 2
u <- superimpose("1" = runifrect(10), "2" = runifrect(20))

# superimposing line segment patterns
X <- as.psp(matrix(runif(20), 5, 4), window=owin())
Y <- as.psp(matrix(runif(40), 10, 4), window=owin())
Z <- superimpose(X, Y)

# being unreasonable
## Not run:
 if(FALSE) {
  crud <- try(superimpose(p1,p2,X,Y)) # Gives an error, of course!
 }

## End(Not run)
```

SupportFun *Support Function*

Description

Create the support function of a two-dimensional domain.

Usage

```
SupportFun(W, origin = c(0, 0))
```

Arguments

W	Window (object of class "owin") representing the domain.
origin	The location of the origin for the support function. Either a numeric vector of length 2 giving the location, or a point pattern containing only one point, or a list with two entries named x and y, or one of the character strings "centroid", "midpoint", "left", "right", "top", "bottom", "topleft", "bottomleft", "topright" or "bottomright" (partially matched).

Details

The support function $h_W(\theta)$ of a set W is defined by

$$h_W(\theta) = \sup\{x\cos\theta + y\sin\theta : (x,y) \in W\}$$

defined for all angles $theta$ in $(0, 2\pi)$.

For any direction theta (in radians), $h_W(\theta)$ is the signed distance from the origin to a line which is tangent to W and perpendicular to theta.

The result of SupportFun(W) is a function in the R language with one argument theta. This function can be applied to any vector of values of θ to compute the values of the support function.

Value

A function in the R language with one argument theta.

Author(s)

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

See Also

FeretDiamFun, polarbody

Examples

```
W <- letterR
Frame(W) <- grow.rectangle(Frame(W), 0.2)
plot(W, main="")

f <- SupportFun(W)
p <- f(pi/6)
plot(infline(p=p, theta=pi/6), col="red", lwd=2)
```

```
## as a function
curve(f, to=2*pi)
```

`symbolmap` *Graphics Symbol Map*

Description

Create a graphics symbol map that associates data values with graphical symbols.

Usage

```
symbolmap(..., range = NULL, inputs = NULL,
                transform = NULL, compress = transform, decompress = NULL)
```

Arguments

`...`	Named arguments specifying the graphical parameters. See Details.
`range`	Optional. Range of numbers that are mapped. A numeric vector of length 2 giving the minimum and maximum values that will be mapped. Incompatible with `inputs`.
`inputs`	Optional. A vector containing all the data values that will be mapped to symbols. Incompatible with `range`.
`transform`	Optional. Experimental. An R function which should applied to input data for the symbol map before the graphics parameters `...` are determined. See section on *Nonlinear transformation*.
`compress`	Optional. Experimental. An R function determining a nonlinear transformation of the domain of the symbol map, to be used when the symbol map is plotted. See section on *Nonlinear transformation*.
`decompress`	Experimental. An R function giving the inverse function of `compress`, if `compress` is specified.

Details

A graphical symbol map is an association between data values and graphical symbols. The command `symbolmap` creates an object of class `"symbolmap"` that represents a graphical symbol map.

Once a symbol map has been created, it can be applied to any suitable data to generate a plot of those data. This makes it easy to ensure that the *same* symbol map is used in two different plots. The symbol map can be plotted as a legend to the plots, and can also be plotted in its own right.

The possible values of data that will be mapped are specified by `range` or `inputs`.

- if `range` is given, it should be a numeric vector of length 2 giving the minimum and maximum values of the range of numbers that will be mapped. These limits must be finite.
- if `inputs` is given, it should be a vector of any atomic type (e.g. numeric, character, logical, factor). This vector contains all the possible data values that will be mapped.

- If neither range nor inputs is given, it is assumed that the possible values are real numbers.

The association of data values with graphical symbols is specified by the other arguments ... which are given in name=value form. These arguments specify the kinds of symbols that will be used, the sizes of the symbols, and graphics parameters for drawing the symbols.

Each graphics parameter can be either a single value, for example shape="circles", or a function(x) which determines the value of the graphics parameter as a function of the data x, for example shape=function(x) ifelse(x > 0, "circles", "squares"). Colourmaps (see colourmap) are also acceptable because they are functions.

Currently recognised graphics parameters, and their allowed values, are:

shape The shape of the symbol: currently either "circles", "squares", "arrows", "crossticks" or NA. This parameter takes precedence over pch. (Crossticks are used only for point patterns on a linear network).

size The size of the symbol: a positive number or zero.

pch Graphics character code: a positive integer, or a single character. See par.

cex Graphics character expansion factor.

cols Colour of plotting characters.

fg,bg Colour of foreground (or symbol border) and background (or symbol interior).

col,lwd,lty Colour, width and style of lines.

etch Logical. If TRUE, each symbol is surrounded by a border drawn in the opposite colour, which improves its visibility against the background. Default is FALSE.

direction,headlength,headangle,arrowtype Numeric parameters of arrow symbols, applicable when shape="arrows". Here direction is the direction of the arrow in degrees anticlockwise from the x axis; headlength is the length of the head of the arrow in coordinate units; headangle is the angle subtended by the point of the arrow; and arrowtype is an integer code specifying which ends of the shaft have arrowheads attached (0 means no arrowheads, 1 is an arrowhead at the start of the shaft, 2 is an arrowhead at the end of the shaft, and 3 is arrowheads at both ends).

A vector of colour values is also acceptable for the arguments col,cols,fg,bg if range is specified.

Value

An object of class "symbolmap".

Nonlinear transformation

- The argument transform defines a transformation that will be applied to the input data for the symbol map.

  If transform is given, it should be an R function. All input data for the symbol map will first be transformed by this function, before the graphical parameters specified by ... are determined.

  A typical example would be a logarithmic symbol map defined by transform = log10 together with something like size = function(x) { 3 * x }. This would mean that a numerical value z will be represented on the plot by a circle of diameter size(transform(z)) = 3 * log10(z) on the physical scale of the plot.

- The arguments compress and decompress define a transformation of the range of numbers that is used only when the symbol map is plotted.

  A typical example would be a logarithmic symbol map defined by compress = log10 and decompress = function(x) { 10^x }.

  The arguments compress and decompress have no effect on the interpretation of the other arguments. They only affect the way in which the symbol map is plotted by plot.symbolmap. For a continuous symbol map, the range of input values is plotted on the compressed scale, but annotated on the original scale. See the Examples.

If transform is given, then the default value of compress is the same function transform. This reflects the fact that, when the user has specified that the input data should be transformed to another scale in order to determine their graphical representation, it would usually be appropriate to display the symbol map legend on the same transformed scale. However this can be overridden by specifying another value for compress, including NULL.

The arguments transform, compress and decompress should be functions which are vectorised (i.e. if x is a vector then compress(x) and decompress(x) are also vectors of the same length as x) and increasing (if x < y then compress(x) < compress(y) and decompress(x) < decompress(y).

The argument decompress is not needed in the following cases:

- If compress is the function log10, then decompress is taken to be its inverse function(x) { 10^x }.
- If compress is a cumulative distribution function (of class "ecdf", "ewcdf" or "interpolatedCDF") then decompress is taken to be its inverse function decompress = quantilefun(compress).

Author(s)

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

See Also

plot.symbolmap to plot the symbol map itself.

invoke.symbolmap to apply the symbol map to some data and plot the resulting symbols.

update.symbolmap to change the symbol map.

There are methods for print and summary for symbol maps.

Examples

```
g <- symbolmap(inputs=letters[1:10], pch=11:20)

g1 <- symbolmap(range=c(0,100), size=function(x) x/50)

## squares and circles with area proportional to |x|
## For 'squares', size is side length; size = sqrt(area)
## For 'circles', size is diameter;    size = sqrt(area * 4/pi)
g2 <- symbolmap(shape=function(x) ifelse(x > 0, "circles", "squares"),
                size=function(x) sqrt(ifelse(x > 0,
                                             abs(x)*4/pi,
                                             abs(x))),
```

```
                bg = function(x) ifelse(abs(x) < 1, "red", "black"))

colmap <- colourmap(topo.colors(20), range=c(0,10))
g3 <- symbolmap(pch=21, bg=colmap, range=c(0,10))
plot(g3)

## logarithmic symbol map
gm <- symbolmap(range=c(1,1000), pch=21, transform=log10,
                size=function(x) { x + 1 })
## input value x is plotted as a circle of diameter = log10(x) + 1.
gm(100)
plot(gm, nsymbols=4)
## note logarithmic scale of legend, because compress=transform by default.

## logarithmic display scale only
gl <- symbolmap(range=c(1,1000), pch=21, compress=log10)
gl(10)
plot(gl, nsymbols=4)
gu <- symbolmap(range=c(1,1000), pch=21)
gu(10)
plot(gu, nsymbols=4)
## log transformation 'compress' does not affect symbol map itself
```

---

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, marks=NULL, keepempty=FALSE, unitname=NULL, check=TRUE)
```

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 (object of class "im") 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".
marks	Optional vector, data frame or hyperframe of marks associated with the tiles.

keepempty	Logical flag indicating whether empty tiles should be retained or deleted.
unitname	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. If this argument is missing or NULL, information about the unitname will be extracted from the other arguments. If this argument is given, it overrides any other information about the unitname.
check	Logical value indicating whether to check the validity of the input data. It is strongly recommended to use the default value check=TRUE.

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, summary, [and [<- for tessellations. Use tiles to extract the list of tiles in a tessellation, tilenames to extract the names of the tiles, and tile.areas to compute their areas.

The tiles may have marks, which can be extracted by marks.tess and changed by marks<-.tess.

Tessellations can be used to classify the points of a point pattern, in split.ppp, cut.ppp and by.ppp.

To construct particular tessellations, see quadrats, hextess, dirichlet, delaunay, venn.tess, polartess, quantess, bufftess and rpoislinetess.

Value

An object of class "tess" representing the tessellation.

Author(s)

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

See Also

marks.tess, plot.tess, [.tess, as.tess, tiles, intersect.tess, split.ppp, cut.ppp, by.ppp, bdist.tiles, tile.areas, as.function.tess.

To construct particular tessellations, see quadrats, hextess, venn.tess, polartess, dirichlet, delaunay, quantess and rpoislinetess.

To divide space into pieces containing equal amounts of stuff, use quantess.

To convert a tessellation to a function, for use as a spatial covariate (associating a numerical value with each tile of the tessellation) use as.function.tess.

Examples

```
A <- tess(xgrid=0:4,ygrid=0:4)
A
plot(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)
plot(E)
G <- tess(image=v, marks=toupper(levels(v)), unitname="km")
G
```

---

test.crossing.psp *Check Whether Segments Cross*

Description

Determine whether there is a crossing (intersection) between each pair of line segments.

Usage

```
test.crossing.psp(A, B)
test.selfcrossing.psp(A)
```

Arguments

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

Details

These functions decide whether the given line segments intersect each other.

If A and B are two spatial patterns of line segments, test.crossing.psp(A, B) returns a logical matrix in which the entry on row i, column j is equal to TRUE if segment A[i] has an intersection with segment B[j].

If A is a pattern of line segments, test.selfcross.psp(A) returns a symmetric logical matrix in which the entry on row i, column j is equal to TRUE if segment A[i] has an intersection with segment A[j].

Value

A logical matrix.

Author(s)

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

See Also

psp

Examples

```
  B <- edges(letterR)
if(require(spatstat.random)) {
  A <- rpoisline(5, Frame(B))
} else {
  FB <- Frame(B)
  A <- as.psp(from=runifrect(5, FB), to=runifrect(5, FB))
}
  MA <- test.selfcrossing.psp(A)
  MAB <- test.crossing.psp(A, B)
```

---

text.ppp *Add Text Labels to Spatial Pattern*

Description

Plots a text label at the location of each point in a spatial point pattern, or each object in a spatial pattern of objects.

Usage

```
## S3 method for class 'ppp'
text(x, ...)


## S3 method for class 'psp'
text(x, ...)
```

Arguments

x	A spatial point pattern (object of class "ppp"), or a spatial pattern of line segments (class "psp").
...	Additional arguments passed to text.default.

Details

These functions are methods for the generic text. A text label is added to the existing plot, at the location of each point in the point pattern x, or near the location of the midpoint of each segment in the segment pattern x.

Additional arguments ... are passed to text.default and may be used to control the placement of the labels relative to the point locations, and the size and colour of the labels.

By default, the labels are the serial numbers 1 to n, where n is the number of points or segments in x. This can be changed by specifying the argument labels, which should be a vector of length n.

Value

Null.

Author(s)

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

See Also

text.default

Examples

```
plot(cells)
text(cells, pos=2)

plot(Frame(cells))
text(cells, cex=1.5)
```

texturemap *Texture Map*

Description

Create a map that associates data values with graphical textures.

Usage

```
texturemap(inputs, textures, ...)
```

Arguments

inputs	A vector containing all the data values that will be mapped to textures.
textures	Optional. A vector of integer codes specifying the textures to which the inputs will be mapped.
...	Other graphics parameters such as col, lwd, lty.

Details

A texture map is an association between data values and graphical textures. The command texturemap creates an object of class "texturemap" that represents a texture map.

Once a texture map has been created, it can be applied to any suitable data to generate a texture plot of those data using textureplot. This makes it easy to ensure that the *same* texture map is used in two different plots. The texture map can also be plotted in its own right.

The argument inputs should be a vector containing all the possible data values (such as the levels of a factor) that are to be mapped.

The textures should be integer values between 1 and 8, representing the eight possible textures described in the help for add.texture. The default is textures = 1:n where n is the length of inputs.

Value

An object of class "texturemap" representing the texture map.

Author(s)

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

See Also

textureplot

Examples

```
texturemap(letters[1:4], 2:5, col=1:4, lwd=2)
```

textureplot *Plot Image or Tessellation Using Texture Fill*

Description

For a factor-valued pixel image, this command plots each level of the factor using a different texture. For a tessellation, each tile is plotted using a different texture.

Usage

```
textureplot(x, ...,
            main, add=FALSE, clipwin=NULL, do.plot = TRUE,
            border=NULL, col = NULL, lwd = NULL, lty = NULL, spacing = NULL,
            textures=1:8,
            legend=TRUE,
            leg.side=c("right", "left", "bottom", "top"),
            legsep=0.1, legwid=0.2)
```

Arguments

x	A tessellation (object of class "tess" or something acceptable to as.tess) with at most 8 tiles, or a pixel image (object of class "im" or something acceptable to as.im) whose pixel values are a factor with at most 8 levels.
...	Other arguments passed to add.texture.
main	Character string giving a main title for the plot.
add	Logical value indicating whether to draw on the current plot (add=TRUE) or to initialise a new plot (add=FALSE).
clipwin	Optional. A window (object of class "owin"). Only this subset of the image will be displayed.
do.plot	Logical. Whether to actually do the plot.
border	Colour for drawing the boundaries between the different regions. The default (border=NULL) means to use par("fg"). Use border=NA to omit borders.
col	Numeric value or vector giving the colour or colours in which the textures should be plotted.
lwd	Numeric value or vector giving the line width or widths to be used.
lty	Numeric value or vector giving the line type or types to be used.
spacing	Numeric value or vector giving the spacing parameter for the textures.
textures	Textures to be used for each level. Either a texture map (object of class "texturemap") or a vector of integer codes (to be interpreted by add.texture).
legend	Logical. Whether to display an explanatory legend.
leg.side	Position of legend relative to main plot.
legsep	Separation between legend and main plot, as a fraction of the shortest side length of the main plot.
legwid	Width (if vertical) or height (if horizontal) of the legend as a fraction of the shortest side length of the main plot.

Details

If x is a tessellation, then each tile of the tessellation is plotted and filled with a texture using add.texture.

If x is a factor-valued pixel image, then for each level of the factor, the algorithm finds the region where the image takes this value, and fills the region with a texture using add.texture.

Value

(Invisible) A texture map (object of class "texturemap") associating a texture with each level of the factor.

Author(s)

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

See Also

im, plot.im, add.texture.

Examples

```
nd <- if(interactive()) 128 else 32
Z <- setcov(owin(), dimyx=nd)
Zcut <- cut(Z, 3, labels=c("Lo", "Med", "Hi"))
textureplot(Zcut)
textureplot(dirichlet(runifrect(6)))
```

---

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

tess, tiles, tilenames, tiles.empty

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)
```

---

tile.centroids *Compute Centroids of Tiles in a Tessellation*

Description

Finds the centroid of each tile in a tessellation and returns them as a point pattern.

Usage

```
tile.centroids(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 determines the centroid of each of the tiles that make up the tessellation x using centroid.owin. The result is a point pattern containing the centroids, listed in the same order as the tiles would be listed by tiles(x).

Value

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

Author(s)

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

See Also

tess, tile.areas

Examples

```
A <- tess(xgrid=0:2,ygrid=0:2)
plot(A, main="")
plot(tile.centroids(A), add=TRUE)
D <- dirichlet(runifrect(6))
plot(D)
plot(tile.centroids(D), add=TRUE)
```

---

tileindex *Determine Which Tile Contains Each Given Point*

Description

Given a tessellation and a list of spatial points, determine which tile of the tessellation contains each of the given points.

Usage

```
tileindex(x, y, Z, close.gaps=TRUE, all.inside=FALSE)
```

Arguments

x, y	Spatial coordinates. Numeric vectors of equal length. (Alternatively y may be missing and x may be an object containing spatial coordinates).
Z	A tessellation (object of class "tess").
close.gaps	Logical value specifying whether all points x, y lying inside the window of Z must be classified as belonging to a tile. This avoids the effect of numerical errors in the tile geometry. See Details.
all.inside	Logical value specifying whether all points x, y should be classified as lying inside a tile. This implies that points lying outside the window will be assigned to the closest tile. See Details.

Details

This function determines which tile of the tessellation Z contains each of the spatial points with coordinates (x[i],y[i]).

The result is a factor, of the same length as x and y, indicating which tile contains each point. The levels of the factor are the names of the tiles of Z.

A point lying outside the window containing the tessellation is assigned the value NA by default. However if all.inside=TRUE is specified, then every point will be assigned to the closest tile, and no NA values are returned.

It is possible that, due to numerical error, a point lying *inside* the window may not be classified as belonging to any of the tiles of Z. If this occurs, the default behaviour is to assign the point to the closest tile. This can be suppressed by setting close.gaps=FALSE (and all.inside=FALSE); in that case, NA values are returned for such points.

Value

A factor, of the same length as x and y, whose levels are the names of the tiles of Z. The factor values may include NA unless all.inside=TRUE was specified.

Author(s)

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

See Also

cut.ppp and split.ppp to divide up the points of a point pattern according to a tessellation.

as.function.tess to create a function whose value is the tile index.

Examples

```
X <- runifrect(7)
V <- dirichlet(X)
tileindex(0.1, 0.4, V)
tileindex(list(x=0.1, y=0.4), Z=V)
tileindex(X, Z=V)
```

---

tilenames *Names of Tiles in a Tessellation*

Description

Extract or Change the Names of the Tiles in a Tessellation.

Usage

```
tilenames(x)
tilenames(x) <- value

## S3 method for class 'tess'
tilenames(x)

## S3 replacement method for class 'tess'
tilenames(x) <- value
```

Arguments

x	A tessellation (object of class "tess").
value	Character vector giving new names for the tiles.

Details

These functions extract or change the names of the tiles that make up the tessellation x.

If the tessellation is a regular grid, the tile names cannot be changed.

Value

tilenames returns a character vector.

Author(s)

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

See Also

tess, tiles

Examples

```
D <- dirichlet(runifrect(10))
tilenames(D)
tilenames(D) <- paste("Cell", 1:10)
tilenames(D)
```

tiles *Extract List of Tiles in a Tessellation*

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

tess, tilenames, tile.areas, tiles.empty

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)
```

`tiles.empty` *Check For Empty Tiles in a Tessellation*

Description

Checks whether each tile in a tessellation is empty or non-empty.

Usage

```
tiles.empty(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`.

It is possible for some tiles of a tessellation to be empty. For example, this can happen when the tessellation `x` is obtained by restricting another tessellation `y` to a smaller spatial domain `w`.

The function `tiles.empty` checks whether each tile is empty or non-empty. The result is a logical vector, with entries equal to `TRUE` when the corresponding tile is empty. Results are given in the same order as the tiles would be listed by `tiles(x)`.

Value

A logical vector.

Author(s)

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

Rolf Turner <rolfturner@posteo.net>

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

See Also

`tess`, `tiles`, `tilenames`, `tile.areas`

Examples

```
A <- tess(xgrid=0:2,ygrid=0:2)
tiles.empty(A)
v <- as.im(function(x,y){factor(round(x^2 + y^2))}, W=owin())
E <- tess(image=v)
tiles.empty(E)
```

timed *Record the Computation Time*

Description

Saves the result of a calculation as an object of class "timed" which includes information about the time taken to compute the result. The computation time is printed when the object is printed.

Usage

```
timed(x, ..., starttime = NULL, timetaken = NULL)
```

Arguments

x	An expression to be evaluated, or an object that has already been evaluated.
starttime	The time at which the computation is defined to have started. The default is the current time. Ignored if timetaken is given.
timetaken	The length of time taken to perform the computation. The default is the time taken to evaluate x.
...	Ignored.

Details

This is a simple mechanism for recording how long it takes to perform complicated calculations (usually for the purposes of reporting in a publication).

If x is an expression to be evaluated, timed(x) evaluates the expression and measures the time taken to evaluate it. The result is saved as an object of the class "timed". Printing this object displays the computation time.

If x is an object which has already been computed, then the time taken to compute the object can be specified either directly by the argument timetaken, or indirectly by the argument starttime.

- timetaken is the duration of time taken to perform the computation. It should be the difference of two clock times returned by proc.time. Typically the user sets begin <- proc.time() before commencing the calculations, then end <- proc.time() after completing the calculations, and then sets timetaken <- end - begin.
- starttime is the clock time at which the computation started. It should be a value that was returned by proc.time at some earlier time when the calculations commenced. When timed is called, the computation time will be taken as the difference between the current clock time and starttime. Typically the user sets begin <- proc.time() before commencing the calculations, and when the calculations are completed, the user calls result <- timed(result, starttime=begin).

If the result of evaluating x belongs to other S3 classes, then the result of timed(x, ...) also inherits these classes, and printing the object will display the appropriate information for these classes as well.

Value

An object inheriting the class "timed".

Author(s)

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

See Also

timeTaken to extract the time taken.

Examples

```
timed(minnndist(cells))

answer <- timed(42, timetaken=4.1e17)
answer
```

---

timeTaken *Extract the Total Computation Time*

Description

Given an object or objects that contain timing information (reporting the amount of computer time taken to compute each object), this function extracts the timing data and evaluates the total time taken.

Usage

```
timeTaken(..., warn=TRUE)
```

Arguments

...	One or more objects of class "timed" containing timing data.
warn	Logical value indicating whether a warning should be issued if some of the arguments do not contain timing information.

Details

An object of class "timed" contains information on the amount of computer time that was taken to compute the object. See timed.

This function extracts the timing information from one or more such objects, and calculates the total time.

Value

An object inheriting the class "timed".

Author(s)

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

See Also

timed

Examples

```
A <- timed(minnndist(bei))
B <- timed(minnndist(redwood))
A
B
timeTaken(A,B)
```

---

transmat *Convert Pixel Array Between Different Conventions*

Description

This function provides a simple way to convert arrays of pixel data between different display conventions.

Usage

```
transmat(m, from, to)
```

Arguments

m	A matrix.
from, to	Specifications of the spatial arrangement of the pixels. See Details.

Details

Pixel images are handled by many different software packages. In virtually all of these, the pixel values are stored in a matrix, and are accessed using the row and column indices of the matrix. However, different pieces of software use different conventions for mapping the matrix indices $[i, j]$ to the spatial coordinates (x, y).

- In the *Cartesian* convention, the first matrix index i is associated with the first Cartesian coordinate x, and j is associated with y. This convention is used in image.default.
- In the *European reading order* convention, a matrix is displayed in the spatial coordinate system as it would be printed in a page of text: i is effectively associated with the negative y coordinate, and j is associated with x. This convention is used in some image file formats.
- In the spatstat convention, i is associated with the increasing y coordinate, and j is associated with x. This is also used in some image file formats.

To convert between these conventions, use the function `transmat`. If a matrix `m` contains pixel image data that is correctly displayed by software that uses the Cartesian convention, and we wish to convert it to the European reading convention, we can type `mm <- transmat(m, from="Cartesian", to="European")`. The transformed matrix `mm` will then be correctly displayed by software that uses the European convention.

Each of the arguments `from` and `to` can be one of the names `"Cartesian"`, `"European"` or `"spatstat"` (partially matched) or it can be a list specifying another convention. For example `to=list(x="-i", y="-j")!` specifies that rows of the output matrix are expected to be displayed as vertical columns in the plot, starting at the right side of the plot, as in the traditional Chinese, Japanese and Korean writing order.

Value

Another matrix obtained by rearranging the entries of `m`.

Author(s)

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

Rolf Turner <rolfturner@posteo.net>

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

Examples

```
opa <- par(mfrow=c(1,2))
# image in spatstat format
Z <- bei.extra$elev
plot(Z, main="plot.im", ribbon=FALSE)
m <- as.matrix(Z)
# convert matrix to format suitable for display by image.default
Y <- transmat(m, from="spatstat", to="Cartesian")
image(Y, asp=0.5, main="image.default", axes=FALSE)
par(opa)
```

---

triangulate.owin *Decompose Window into Triangles*

Description

Given a spatial window, this function decomposes the window into disjoint triangles. The result is a tessellation of the window in which each tile is a triangle.

Usage

```
triangulate.owin(W)
```

Arguments

`W`	Window (object of class `"owin"`).

Details

The window W will be decomposed into disjoint triangles. The result is a tessellation of W in which each tile is a triangle. All triangle vertices lie on the boundary of the original polygon.

The window is first converted to a polygonal window using as.polygonal. The vertices of the polygonal window are extracted, and the Delaunay triangulation of these vertices is computed using delaunay. Each Delaunay triangle is intersected with the window: if the result is not a triangle, the triangulation procedure is applied recursively to this smaller polygon.

Value

Tessellation (object of class "tess").

Author(s)

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

See Also

tess, delaunay, as.polygonal

Examples

```
plot(triangulate.owin(letterR))
```

---

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

See Also

grow.rectangle, erosion, 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))
```

---

tweak.colourmap *Change Colour Values in a Colour Map*

Description

Assign new colour values to some of the entries in a colour map.

Usage

```
tweak.colourmap(m, col, ..., inputs=NULL, range=NULL)
```

Arguments

m	A colour map (object of class "colourmap").
inputs	Input values to the colour map, to be assigned new colours. Incompatible with range.
range	Numeric vector of length 2 specifying a range of numerical values which should be assigned a new colour. Incompatible with inputs.
col	Replacement colours for the specified inputs or the specified range of values.
...	Other arguments are ignored.

Details

This function changes the colour map m by assigning new colours to each of the input values specified by inputs, or by assigning a single new colour to the range of input values specified by range.

The modified colour map is returned.

Value

Another colour map (object of class "colourmap").

Author(s)

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

See Also

colourmap, interp.colourmap, colouroutputs, colourtools.

Examples

```
co <- colourmap(rainbow(32), range=c(0,1))
plot(tweak.colourmap(co, inputs=c(0.5, 0.6), "white"))
plot(tweak.colourmap(co, range=c(0.5,0.6), "white"))
```

---

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

See Also

quad.object, as.ppp

Examples

```
Q <- quadscheme(simdat, default.dummy(simdat))
U <- union.quad(Q)
# 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, ..., warn=FALSE)

 ## S3 method for class 'ppx'
unique(x, ..., warn=FALSE)
```

Arguments

x	A spatial point pattern (object of class "ppp" or "ppx").
...	Arguments passed to duplicated.ppp or duplicated.data.frame.
warn	Logical. If TRUE, issue a warning message if any duplicated points were found.

Details

These are methods for the generic function unique for point pattern datasets (of class "ppp", see ppp.object, or class "ppx").

This function removes duplicate points in x, and returns a point pattern.

Two points in a point pattern are deemed to be identical if their x, y coordinates are the same, *and* their marks are the same (if they carry marks). This is the default rule: see duplicated.ppp for other options.

Value

Another point pattern object.

Author(s)

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

and Rolf Turner <rolfturner@posteo.net>

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)
unique(X, rule="deldir")
```

---

uniquemap.ppp *Map Duplicate Entries to Unique Entries*

Description

Determine whether points in a point pattern are duplicated, choose a unique representative for each set of duplicates, and map the duplicates to the unique representative.

Usage

```
## S3 method for class 'ppp'
uniquemap(x)

## S3 method for class 'lpp'
uniquemap(x)

## S3 method for class 'ppx'
uniquemap(x)
```

Arguments

x	A point pattern (object of class "ppp", "lpp", "pp3" or "ppx").

Details

The function uniquemap is generic, with methods for point patterns, and a default method.

This function determines whether any points of x are duplicated, and constructs a mapping of the indices of x so that all duplicates are mapped to a unique representative index.

The result is an integer vector u such that u[j] = i if the points x[i] and x[j] are identical and point i has been chosen as the unique representative. The entry u[i] = i means either that point i is unique, or that it has been chosen as the unique representative of its equivalence class.

Value

An integer vector.

Author(s)

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

See Also

unique.ppp, duplicated.ppp, uniquemap.default

Examples

```
Y <- runifrect(4)
X <- Y[c(1,2,3,4,2,1)]
uniquemap(X)
```

---

unitname *Name for Unit of Length*

Description

Inspect or change the name of the unit of length in a spatial dataset.

Usage

```
unitname(x)
unitname(x) <- value
## S3 method for class 'im'
unitname(x)
## S3 method for class 'owin'
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 'tess'
unitname(x)
## S3 replacement method for class 'im'
unitname(x) <- value
## S3 replacement method for class 'owin'
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 'tess'
unitname(x) <- value
```

Arguments

x	A spatial dataset. Either a point pattern (object of class "ppp"), a line segment pattern (object of class "psp"), a window (object of class "owin"), a pixel image (object of class "im"), a tessellation (object of class "tess"), a quadrature scheme (object of class "quad"), or a fitted point process model (object of class "ppm" or "kppm" or "slrm" or "dppm" or "minconfit").
value	Name of the unit of length. See Details.

Details

Spatial datasets in the **spatstat** package may include the name of the unit of length. This name is used when printing or plotting the dataset, and in some other applications.

`unitname(x)` extracts this name, and `unitname(x) <- value` sets the name to `value`.

A valid name is either

- a single character string
- a vector of two character strings giving the singular and plural forms of the unit name
- a list of length 3, containing two character strings giving the singular and plural forms of the basic unit, and a number specifying the multiple of this unit.

Note that re-setting the name of the unit of length *does not* affect the numerical values in `x`. It changes only the string containing the name of the unit of length. To rescale the numerical values, use `rescale`.

Value

The return value of `unitname` is an object of class "unitname" containing the name of the unit of length in `x`. There are methods for `print`, `summary`, `as.character`, `rescale` and `compatible`.

Author(s)

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

See Also

`rescale`, `owin`, `ppp`

Examples

```
X <- runifrect(20)

# if the unit of length is 1 metre:
unitname(X) <- c("metre", "metres")

# if the unit of length is 6 inches:
unitname(X) <- list("inch", "inches", 6)
```

---

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

See Also

ppp.object, psp.object

Examples

```
hicks <- lansing[lansing$marks == "hickory", ]

# plot(hicks)  # still a marked point pattern, but only 1 value of marks
# plot(unmark(hicks)) # unmarked
```

---

unstack.ppp *Separate Multiple Columns of Marks*

Description

Given a spatial pattern with several columns of marks, take one column at a time, and return a list of spatial patterns each having only one column of marks.

Usage

```
## S3 method for class 'ppp'
unstack(x, ...)

## S3 method for class 'psp'
unstack(x, ...)

## S3 method for class 'tess'
unstack(x, ...)
```

Arguments

x	A spatial point pattern (object of class "ppp") or a spatial pattern of line segments (object of class "psp") or a spatial tessellation (object of class "tess").
...	Ignored.

Details

The functions defined here are methods for the generic unstack. The functions expect a spatial object x which has several columns of marks; they separate the columns, and return a list of spatial objects, each having only one column of marks.

If x has several columns of marks (i.e. marks(x) is a matrix, data frame or hyperframe with several columns), then y <- unstack(x) is a list of spatial objects, each of the same kind as x. The jth entry y[[j]] is equivalent to x except that it only includes the jth column of marks(x).

If x has no marks, or has only a single column of marks, the result is a list consisting of one entry, which is x.

Value

A list, of class "solist", whose entries are objects of the same type as x.

Author(s)

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

See Also

unstack

unstack.msr

See also methods for the generic split such as split.ppp.

Examples

```
finpines
unstack(finpines)
```

---

unstack.solist *Unstack Each Spatial Object in a List of Objects*

Description

Given a list of two-dimensional spatial objects, apply

Usage

```
## S3 method for class 'solist'
unstack(x, ...)

## S3 method for class 'layered'
unstack(x, ...)
```

Arguments

x	An object of class "solist" or "layered" representing a list of two-dimensional spatial objects.
...	Ignored.

Details

The functions defined here are methods for the generic unstack. They expect the argument x to be a list of spatial objects, of class "solist" or "layered".

Each spatial object in the list x will be unstacked by applying the relevant method for unstack. This means that

- a marked point pattern with several columns of marks will be separated into several point patterns, each having a single column of marks
- a measure with k-dimensional vector values will be separated into k measures with scalar values

The resulting unstacked objects will be collected into a list of the same kind as x. Typically the length of unstack(x) is greater than the length of x.

Value

A list belonging to the same class as x.

Author(s)

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

See Also

unstack

unstack.msr, unstack.ppp, unstack.psp

Examples

```
A <- solist(finpines=finpines, cells=cells)
A
unstack(A)
B <- layered(fin=finpines, loc=unmark(finpines),
             plotargs=list(list(), list(pch=16)))
B
plot(B)
unstack(B)
plot(unstack(B))
```

update.symbolmap *Update a Graphics Symbol Map.*

Description

This command updates the object using the arguments given.

Usage

```
## S3 method for class 'symbolmap'
update(object, ...)
```

Arguments

object	Graphics symbol map (object of class "symbolmap").
...	Additional or replacement arguments to symbolmap.

Details

This is a method for the generic function update for the class "symbolmap" of graphics symbol maps. It updates the object using the parameters given in the extra arguments ....

The extra arguments must be given in the form name=value and must be recognisable to symbolmap. They override any parameters of the same name in object.

Value

Another object of class "symbolmap".

Author(s)

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

See Also

symbolmap to create a graphics symbol map.

Examples

```
g <- symbolmap(size=function(x) x/50)
g
update(g, range=c(0,1))
update(g, size=42)
update(g, shape="squares", range=c(0,1))
```

venn.tess *Tessellation Delimited by Several Sets*

Description

Given a list of windows, construct the tessellation formed by all combinations of inclusion/exclusion of these windows.

Usage

```
venn.tess(..., window = NULL, labels=FALSE)
```

Arguments

...	Sets which delimit the tessellation. Any number of windows (objects of class "owin") or tessellations (objects of class "tess").
window	Optional. The bounding window of the resulting tessellation. If not specified, the default is the union of all the arguments ....
labels	Logical value, specifying whether to attach marks to each tile that reveal how it was formed.

Details

The arguments ... may be any number of windows. This function constructs a tessellation, like a Venn diagram, whose boundaries are made up of the boundaries of these sets. Each tile of the tessellation is defined by one of the possible combinations in which each set is either included or excluded.

If the arguments ... are named, then the resulting tiles will also have tile names, which identify the inclusion/exclusion combinations defining each tile. See the Examples.

If labels=TRUE then the tiles have marks which indicate the inclusion/exclusion combinations defining each tile. See the Examples.

Value

A tessellation (object of class "tess").

Author(s)

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

See Also

intersect.tess.

To construct other kinds of tessellations, see tess, quadrats, hextess, polartess, dirichlet, delaunay, quantess and rpoislinetess.

Examples

```
A <- square(1)
B <- square(c(-0.5,0.5))
W <- square(c(-1, 1.5))
V <- venn.tess(A=A, B=B, window=W)
V
plot(V, do.labels=TRUE)
Vlab <- venn.tess(A=A, B=B, window=W, labels=TRUE)
marks(Vlab)
```

`vertices` *Vertices of a Window*

Description

Finds the vertices of a window, or similar object.

Usage

```
vertices(w)

## S3 method for class 'owin'
vertices(w)
```

Arguments

`w`	A window (object of class `"owin"`) or similar object.

Details

This function computes the vertices ('corners') of a spatial window or other object.

For `vertices.owin`, 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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

owin.object.

Examples

```
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, and a method for the class "linnet" of linear networks, which returns the length of the network.

Value

The numerical value of the volume of the object.

Author(s)

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

See Also

area.owin, volume.box3, volume.boxx.

---

where.max *Find Location of Maximum in a Pixel Image*

Description

Finds the spatial location(s) where a given pixel image attains its maximum or minimum value.

Usage

```
where.max(x, first = TRUE)
where.min(x, first = TRUE)
```

Arguments

x	A pixel image (object of class "im") or data that can be converted to a pixel image by as.im.
first	Logical value. If TRUE (the default), then only one location will be returned. If FALSE, then all locations where the maximum is achieved will be returned.

Details

This function finds the spatial location or locations where the pixel image x attains its maximum or minimum value. The result is a point pattern giving the locations.

If first=TRUE (the default), then only one location will be returned, namely the location with the smallest y coordinate value which attains the maximum or minimum. This behaviour is analogous to the functions which.min and which.max.

If first=FALSE, then the function returns the locations of all pixels where the maximum (or minimum) value is attained. This could be a large number of points.

Value

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

Author(s)

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

See Also

Summary.im for computing the minimum and maximum of pixel values; eval.im and Math.im for mathematical expressions involving images; solutionset for finding the set of pixels where a statement is true.

Examples

```
D <- distmap(letterR, invert=TRUE)
plot(D)
plot(where.max(D), add=TRUE, pch=16, cols="green")
```

---

whichhalfplane *Test Which Side of Infinite Line a Point Falls On*

Description

Given an infinite line and a spatial point location, determine which side of the line the point falls on.

Usage

```
whichhalfplane(L, x, y = NULL)
```

Arguments

L	Object of class "infline" specifying one or more infinite straight lines in two dimensions.
x, y	Arguments acceptable to xy.coords specifying the locations of the points.

Details

An infinite line L divides the two-dimensional plane into two half-planes. This function returns a matrix M of logical values in which M[i,j] = TRUE if the jth spatial point lies below or to the left of the ith line.

Value

A logical matrix.

Author(s)

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

See Also

infline

Examples

```
L <- infline(p=runif(3), theta=runif(3, max=2*pi))
X <- runifrect(4)
whichhalfplane(L, X)
```

`Window` *Extract or Change the Window of a Spatial Object*

Description

Given a spatial object (such as a point pattern or pixel image) in two dimensions, these functions extract or change the window in which the object is defined.

Usage

```
   Window(X, ...)

   Window(X, ...) <- value

   ## S3 method for class 'ppp'
Window(X, ...)

   ## S3 replacement method for class 'ppp'
Window(X, ...) <- value

   ## S3 method for class 'quad'
Window(X, ...)

   ## S3 replacement method for class 'quad'
Window(X, ...) <- value

   ## S3 method for class 'psp'
Window(X, ...)

   ## S3 replacement method for class 'psp'
Window(X, ...) <- value

   ## S3 method for class 'im'
Window(X, ...)

   ## S3 replacement method for class 'im'
Window(X, ...) <- value
```

Arguments

`X`	A spatial object such as a point pattern, line segment pattern or pixel image.
`...`	Extra arguments. They are ignored by all the methods listed here.
`value`	Another window (object of class `"owin"`) to be used as the window for `X`.

Details

The functions Window and Window<- are generic.

Window(X) extracts the spatial window in which X is defined.

Window(X) <- W changes the window in which X is defined to the new window W, and *discards any data outside* W. In particular:

- If X is a point pattern (object of class "ppp") then Window(X) <- W discards any points of X which fall outside W.
- If X is a quadrature scheme (object of class "quad") then Window(X) <- W discards any points of X which fall outside W, and discards the corresponding quadrature weights.
- If X is a line segment pattern (object of class "psp") then Window(X) <- W clips the segments of X to the boundaries of W.
- If X is a pixel image (object of class "im") then Window(X) <- W has the effect that pixels lying outside W are retained but their pixel values are set to NA.

Many other classes of spatial object have a method for Window, but not Window<-. See Window.tess.

Value

The result of Window is a window (object of class "owin").

The result of Window<- is the updated object X, of the same class as X.

Author(s)

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

See Also

Window.ppm

Examples

```
## point patterns
Window(cells)
X <- demopat
Window(X)
Window(X) <- as.rectangle(Window(X))

## line segment patterns
X <- psp(runif(10), runif(10), runif(10), runif(10), window=owin())
Window(X)
Window(X) <- square(0.5)

## images
Z <- setcov(owin())
Window(Z)
Window(Z) <- square(0.5)
```

Window.tess *Extract Window of Spatial Object*

Description

Given a spatial object (such as a point pattern or pixel image) in two dimensions, these functions extract the window in which the object is defined.

Usage

```
 ## S3 method for class 'quadratcount'
Window(X, ...)

 ## S3 method for class 'tess'
Window(X, ...)

 ## S3 method for class 'layered'
Window(X, ...)

 ## S3 method for class 'distfun'
Window(X, ...)

 ## S3 method for class 'nnfun'
Window(X, ...)

 ## S3 method for class 'funxy'
Window(X, ...)
```

Arguments

X	A spatial object.
...	Ignored.

Details

These are methods for the generic function Window which extract the spatial window in which the object X is defined.

Value

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

Author(s)

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

See Also

Window, Window.ppp, Window.psp.

owin.object

Examples

```
A <- quadratcount(cells, 4)
Window(A)
```

---

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 parent.frame().

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@curtin.edu.au>

and Rolf Turner <rolfturner@posteo.net>

See Also

`hyperframe`, `plot.hyperframe`

Examples

```
  # generate Poisson point patterns with intensities 10 to 100
   H <- hyperframe(L=seq(10,100, by=10))
if(require(spatstat.random)) {
   X <- with(H, rpoispp(L))
} else {
   X <- with(H, runifrect(rpois(1, L)))
}
```

`yardstick` *Text, Arrow or Scale Bar in a Diagram*

Description

Create spatial objects that represent a text string, an arrow, or a yardstick (scale bar).

Usage

```
textstring(x, y, txt = NULL, ...)

onearrow(x0, y0, x1, y1, txt = NULL, ...)

yardstick(x0, y0, x1, y1, txt = NULL, ...)
```

Arguments

x, y	Coordinates where the text should be placed.
x0, y0, x1, y1	Spatial coordinates of both ends of the arrow or yardstick. Alternatively x0 can be a point pattern (class "ppp") containing exactly two points, or a line segment pattern (class "psp") consisting of exactly one line segment.
txt	The text to be displayed beside the line segment. Either a character string or an expression.
...	Additional named arguments for plotting the object.

Details

These commands create objects that represent components of a diagram:

- textstring creates an object that represents a string of text at a particular spatial location.
- onearrow creates an object that represents an arrow between two locations.
- yardstick creates an object that represents a scale bar: a line segment indicating the scale of the plot.

To display the relevant object, it should be plotted, using plot. See the help files for the plot methods plot.textstring, plot.onearrow and plot.yardstick.

These objects are designed to be included as components in a layered object or a solist. This makes it possible to build up a diagram consisting of many spatial objects, and to annotate the diagram with arrows, text and so on, so that ultimately the entire diagram is plotted using plot.

Value

An object of class "diagramobj" which also belongs to one of the special classes "textstring", "onearrow" or "yardstick". There are methods for plot, print, "[" and shift.

Author(s)

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

See Also

plot.textstring, plot.onearrow, plot.yardstick.

Examples

```
X <- rescale(swedishpines)
plot(X, pch=16, main="")
yd <- yardstick(0,0,1,1, "diagonal")
yy <- yardstick(X[1:2])
ys <- yardstick(as.psp(list(xmid=4, ymid=0.5, length=1, angle=0),
                       window=Window(X)),
                txt="1 m")
ys
plot(ys, angle=90)
scalardilate(ys, 2)
```

`zapsmall.im` *Rounding of Pixel Values*

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 `zapsmall` 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 <rubak@math.aau.dk> and Adrian Baddeley <Adrian.Baddeley@curtin.edu.au>

See Also

`zapsmall`

Examples

```
Z <- as.im(function(x,y) { exp(-40*(x+y)) }, square(1), dimyx=32)
zapsmall.im(Z)
```

Index

∗ **Dirichlet tessellation**
dirichlet, 173
∗ **Geometrical transformations**
affine, 24
affine.im, 25
affine.owin, 26
affine.ppp, 27
affine.psp, 28
affine.tess, 29
∗ **IO**
scanpp, 572
∗ **Linear network**
is.linim, 298
is.linnet, 298
is.lpp, 299
∗ **Tessellation**
as.data.frame.tess, 50
as.function.tess, 53
as.tess, 81
bufftess, 94
chop.tess, 101
connected.tess, 129
Extract.tess, 230
hextess, 254
integral.tessfun, 276
intersect.tess, 288
marks.tess, 322
plot.tess, 475
polartess, 484
quantess, 523
tess, 633
venn.tess, 663
∗ **Three-dimensional**
as.box3, 42
box3, 92
closepairs.pp3, 109
crossdist.pp3, 145
diameter.box3, 167
methods.box3, 336
methods.pp3, 342
nncross.pp3, 359
nndist.pp3, 367
nnwhich.pp3, 381
pairdist.pp3, 400
plot.pp3, 452
pp3, 485
∗ **attribute**
im.object, 267
metric.object, 346
owin.object, 393
ppp.object, 490
pppmatching.object, 496
psp.object, 510
quad.object, 513
∗ **color**
as.colourmap, 44
beachcolours, 86
colourmap, 114
colouroutputs, 116
colourtools, 118
default.image.colours, 157
interp.colourmap, 283
pHcolourmap, 416
plot.colourmap, 429
restrict.colourmap, 545
rev.colourmap, 546
tweak.colourmap, 652
∗ **datagen**
box3, 92
boxx, 93
default.dummy, 156
disc, 178
discs, 182
ellipse, 202
gridcentres, 242
gridweights, 244
hextess, 254
im, 263

infline, 271
owin, 390
pixelquad, 424
pp3, 485
ppp, 486
pppmatching, 495
ppx, 498
psp, 509
quadrats, 517
quadscheme, 518
quadscheme.logi, 521
quasirandom, 527
regularpolygon, 532
rexplode, 547
rgbim, 548
rjitter, 551
rjitter.pp3, 553
rlinegrid, 555
rQuasi, 564
rsyst, 565
runifrect, 569
spokes, 607
square, 608
stratrand, 609
tess, 633

∗ **data**
sessionLibs, 576

∗ **environment**
requireversion, 536

∗ **hplot**
add.texture, 23
contour.im, 130
contour.imlist, 132
default.colourmap, 154
default.image.colours, 157
default.symbolmap, 158
default.symbolmap.ppp, 159
invoke.symbolmap, 291
layered, 310
layerplotargs, 311
persp.im, 408
persp.ppp, 411
perspPoints, 414
plot.anylist, 426
plot.colourmap, 429
plot.hyperframe, 431
plot.im, 433
plot.imlist, 440
plot.layered, 442
plot.listof, 443
plot.onearrow, 447
plot.owin, 448
plot.pp3, 452
plot.ppp, 453
plot.pppmatching, 461
plot.psp, 462
plot.quad, 465
plot.quadratcount, 467
plot.solist, 468
plot.splitppp, 471
plot.symbolmap, 473
plot.tess, 475
plot.textstring, 477
plot.texturemap, 478
plot.yardstick, 480
symbolmap, 630
text.ppp, 636
texturemap, 638
textureplot, 639
transmat, 649
update.symbolmap, 662
yardstick, 672

∗ **iplot**
clickbox, 102
clickdist, 103
clickpoly, 104
clickppp, 105
identify.ppp, 260
identify.psp, 261
identify.tess, 262
run.simplepanel, 566
simplepanel, 586

∗ **iteration**
applynbd, 35

∗ **list**
anylist, 32
as.solist, 80
Extract.anylist, 209
Extract.solist, 227
solapply, 591
solist, 592

∗ **manip**
anylist, 32
append.psp, 34
as.box3, 42
as.data.frame.hyperframe, 45

as.data.frame.ppp, 48
as.data.frame.psp, 49
as.function.im, 51
as.function.owin, 52
as.function.tess, 53
as.hyperframe, 54
as.hyperframe.ppx, 55
as.im, 57
as.layered, 63
as.mask, 64
as.owin, 69
as.polygonal, 72
as.ppp, 74
as.psp, 76
as.rectangle, 79
as.solist, 80
as.tess, 81
bufftess, 94
by.im, 96
by.ppp, 97
cbind.hyperframe, 99
commonGrid, 120
compatible, 121
compatible.im, 122
concatxy, 124
coords, 139
crossing.psp, 150
delaunay, 163
delaunayDistance, 164
dirichlet, 173
dirichletAreas, 174
dirichletVertices, 175
discretise, 181
domain, 193
edges, 196
edges2triangles, 197
edges2vees, 199
edit.hyperframe, 200
edit.ppp, 201
endpoints.psp, 203
eval.im, 208
Extract.anylist, 209
Extract.hyperframe, 210
Extract.im, 213
Extract.layered, 216
Extract.listof, 217
Extract.owin, 219
Extract.ppp, 220
Extract.ppx, 223
Extract.psp, 225
Extract.quad, 226
Extract.solist, 227
Extract.splitppp, 229
Extract.tess, 230
FeretBox, 233
FeretDiamFun, 234
Frame, 239
grow.boxx, 245
grow.rectangle, 246
harmonise, 247
harmonise.im, 248
harmonise.owin, 249
harmoniseLevels, 250
headtail, 252
hyperframe, 258
im, 263
im.apply, 265
interp.im, 284
is.convex, 295
is.empty, 296
is.im, 297
is.linim, 298
is.linnet, 298
is.lpp, 299
is.marked, 300
is.marked.ppp, 301
is.multitype, 302
is.multitype.ppp, 303
is.na.hyperframe, 304
is.na.solist, 305
is.owin, 307
is.ppp, 307
is.rectangle, 308
levelset, 314
lut, 317
marks, 319
marks.psp, 321
marks.tess, 322
mergeLevels, 333
mergeTiles, 335
nearest.raster.point, 352
nearestValue, 354
nestsplit, 355
nobjects, 384
npoints, 385
nsegments, 386

nvertices, 387
owin2mask, 394
padimage, 397
periodify, 407
pixelcentres, 417
pixellate, 418
pixellate.owin, 419
pixellate.ppp, 420
pixellate.psp, 423
pointsOnLines, 482
polarbody, 483
polartess, 484
psp2mask, 512
quantess, 523
raster.x, 529
relevel.im, 533
Replace.im, 534
rescue.rectangle, 544
rexplode, 547
rgbim, 548
rjitter, 551
rjitter.pp3, 553
rotate.infline, 557
round.ppp, 562
selfcrossing.psp, 574
selfcut.psp, 575
shift, 578
shift.im, 579
shift.owin, 580
shift.ppp, 581
shift.ppx, 582
shift.psp, 584
solapply, 591
solist, 592
solutionset, 593
split.hyperframe, 600
split.im, 601
split.ppp, 602
split.ppx, 605
subset.hyperframe, 611
subset.ppp, 612
subset.psp, 614
superimpose, 626
SupportFun, 628
tile.areas, 640
tile.centroids, 641
tileindex, 642
tilenames, 643
tiles, 645
tiles.empty, 646
transmat, 649
triangulate.owin, 650
trim.rectangle, 651
union.quad, 653
unitname, 656
unmark, 658
unstack.ppp, 659
unstack.solist, 660
whichhalfplane, 667
Window, 668
Window.tess, 670
with.hyperframe, 671

∗ **math**

affine, 24
affine.im, 25
affine.owin, 26
affine.ppp, 27
affine.psp, 28
affine.tess, 29
angles.psp, 31
area.owin, 38
areaGain, 39
areaLoss, 40
bdist.pixels, 82
bdist.points, 84
bdist.tiles, 85
border, 87
boundingcircle, 91
centroid.owin, 100
chop.tess, 101
clip.infline, 106
closepairs, 107
closepairs.pp3, 109
closetriples, 111
closing, 112
complement.owin, 123
connected, 125
connected.ppp, 128
connected.tess, 129
convexmetric, 136
convolve.im, 138
covering, 142
crossdist, 143
crossdist.default, 144
crossdist.pp3, 145
crossdist.ppp, 146

crossdist.ppx, 147
crossdist.psp, 148
deltametric, 165
diameter, 166
diameter.box3, 167
diameter.boxx, 168
diameter.owin, 170
dilated.areas, 171
dilation, 172
dirichletAreas, 174
dirichletVertices, 175
discpartarea, 180
distbdry, 184
distfun, 185
distmap, 187
distmap.owin, 188
distmap.ppp, 190
distmap.psp, 192
eroded.areas, 204
erosion, 205
erosionAny, 207
extrapolate.psp, 231
fardist, 232
FeretBox, 233
FeretDiamFun, 234
fillholes.owin, 235
flipxy, 236
framedist.pixels, 240
funxy, 241
has.close, 251
imcov, 268
incircle, 269
inside.boxx, 272
inside.owin, 273
integral.im, 275
integral.tessfun, 276
intersect.boxx, 285
intersect.owin, 286
intersect.tess, 288
invoke.metric, 290
is.connected, 293
is.connected.ppp, 294
is.subset.owin, 309
lengths_psp, 313
matchingdist, 325
maxnndist, 331
midpoints.psp, 347
MinkowskiSum, 348
nearestsegment, 353
nncross, 356
nncross.pp3, 359
nncross.ppx, 362
nndist, 364
nndist.pp3, 367
nndist.ppx, 369
nndist.psp, 371
nnfun, 373
nnmap, 374
nnwhich, 378
nnwhich.pp3, 381
nnwhich.ppx, 382
opening, 388
overlap.owin, 389
pairdist, 398
pairdist.default, 399
pairdist.pp3, 400
pairdist.ppp, 402
pairdist.ppx, 403
pairdist.psp, 404
perimeter, 406
polarbody, 483
pppdist, 492
project2segment, 506
project2set, 507
quadratcount, 514
rectdistmap, 530
reflect, 531
rescale, 537
rescale.im, 539
rescale.owin, 540
rescale.ppp, 541
rescale.psp, 542
rotate, 556
rotate.im, 556
rotate.owin, 559
rotate.ppp, 560
rotate.psp, 561
rounding.ppp, 563
scalardilate, 570
setcov, 577
sidelengths.owin, 585
simplify.owin, 589
SupportFun, 628
test.crossing.psp, 635
venn.tess, 663
vertices, 664

volume, 665
where.max, 666
* **methods**
anyNA.im, 33
as.data.frame.im, 46
as.data.frame.owin, 47
as.data.frame.tess, 50
as.matrix.im, 67
as.matrix.owin, 68
by.im, 96
by.ppp, 97
cut.im, 151
cut.ppp, 152
duplicated.ppp, 195
hist.funxy, 255
hist.im, 256
is.boxx, 292
is.NAobject, 306
levelset.distfun, 316
Math.im, 327
Math.imlist, 329
mean.im, 332
methods.box3, 336
methods.boxx, 337
methods.distfun, 338
methods.funxy, 340
methods.layered, 341
methods.pp3, 342
methods.unitname, 345
NAobject, 351
nnmark, 376
quantile.im, 525
scaletointerval, 571
split.im, 601
split.ppp, 602
split.ppx, 605
summary.anylist, 616
summary.distfun, 617
summary.im, 618
summary.listof, 619
summary.owin, 620
summary.ppp, 621
summary.psp, 622
summary.quad, 623
summary.solist, 624
summary.splitppp, 625
unique.ppp, 654
uniquemap.ppp, 655
zapsmall.im, 674
* **metric**
invoke.metric, 290
* **models**
intensity, 277
* **nonparametric**
intensity.ppp, 278
intensity.psp, 281
intensity.quadratcount, 282
quantilefun.im, 526
* **package**
spatstat.geom-package, 12
* **print**
print.im, 499
print.owin, 500
print.ppp, 501
print.psp, 502
print.quad, 503
progressreport, 504
* **programming**
applynbd, 35
eval.im, 208
im.apply, 265
levelset, 314
markstat, 323
solutionset, 593
with.hyperframe, 671
* **smooth**
nnmark, 376
* **spatial**
add.texture, 23
affine, 24
affine.im, 25
affine.owin, 26
affine.ppp, 27
affine.psp, 28
affine.tess, 29
angles.psp, 31
anyNA.im, 33
append.psp, 34
applynbd, 35
area.owin, 38
areaGain, 39
areaLoss, 40
as.box3, 42
as.colourmap, 44
as.data.frame.hyperframe, 45
as.data.frame.im, 46

as.data.frame.owin, 47
as.data.frame.ppp, 48
as.data.frame.psp, 49
as.data.frame.tess, 50
as.function.im, 51
as.function.owin, 52
as.function.tess, 53
as.hyperframe, 54
as.hyperframe.ppx, 55
as.im, 57
as.layered, 63
as.mask, 64
as.matrix.im, 67
as.matrix.owin, 68
as.owin, 69
as.polygonal, 72
as.ppp, 74
as.psp, 76
as.rectangle, 79
as.solist, 80
as.tess, 81
bdist.pixels, 82
bdist.points, 84
bdist.tiles, 85
beachcolours, 86
border, 87
bounding.box.xy, 88
boundingbox, 89
boundingcircle, 91
box3, 92
boxx, 93
bufftess, 94
by.im, 96
by.ppp, 97
cbind.hyperframe, 99
centroid.owin, 100
chop.tess, 101
clickbox, 102
clickdist, 103
clickpoly, 104
clickppp, 105
clip.infline, 106
closepairs, 107
closepairs.pp3, 109
closetriples, 111
closing, 112
colourmap, 114
colouroutputs, 116
commonGrid, 120
compatible, 121
compatible.im, 122
complement.owin, 123
concatxy, 124
connected, 125
connected.ppp, 128
connected.tess, 129
contour.im, 130
contour.imlist, 132
convexhull, 133
convexhull.xy, 134
convexify, 135
convexmetric, 136
convolve.im, 138
coords, 139
corners, 141
covering, 142
crossdist, 143
crossdist.default, 144
crossdist.pp3, 145
crossdist.ppp, 146
crossdist.ppx, 147
crossdist.psp, 148
crossing.psp, 150
cut.im, 151
cut.ppp, 152
default.colourmap, 154
default.dummy, 156
default.symbolmap, 158
default.symbolmap.ppp, 159
delaunay, 163
delaunayDistance, 164
deltametric, 165
diameter, 166
diameter.box3, 167
diameter.boxx, 168
diameter.owin, 170
dilated.areas, 171
dilation, 172
dirichlet, 173
dirichletAreas, 174
dirichletVertices, 175
dirichletWeights, 176
disc, 178
discpartarea, 180
discretise, 181
discs, 182

distbdry, 184
distfun, 185
distmap, 187
distmap.owin, 188
distmap.ppp, 190
distmap.psp, 192
domain, 193
duplicated.ppp, 195
edges, 196
edges2triangles, 197
edges2vees, 199
edit.hyperframe, 200
edit.ppp, 201
ellipse, 202
endpoints.psp, 203
eroded.areas, 204
erosion, 205
erosionAny, 207
eval.im, 208
Extract.anylist, 209
Extract.hyperframe, 210
Extract.im, 213
Extract.layered, 216
Extract.listof, 217
Extract.owin, 219
Extract.ppp, 220
Extract.ppx, 223
Extract.psp, 225
Extract.quad, 226
Extract.solist, 227
Extract.splitppp, 229
Extract.tess, 230
extrapolate.psp, 231
fardist, 232
FeretBox, 233
FeretDiamFun, 234
fillholes.owin, 235
flipxy, 236
Frame, 239
framedist.pixels, 240
funxy, 241
gridcentres, 242
gridweights, 244
grow.boxx, 245
grow.rectangle, 246
harmonise, 247
harmonise.im, 248
harmonise.owin, 249
has.close, 251
headtail, 252
hextess, 254
hist.funxy, 255
hist.im, 256
hyperframe, 258
identify.ppp, 260
identify.psp, 261
identify.tess, 262
im, 263
im.apply, 265
im.object, 267
imcov, 268
incircle, 269
infline, 271
inside.boxx, 272
inside.owin, 273
integral.im, 275
integral.tessfun, 276
intensity, 277
intensity.ppp, 278
intensity.psp, 281
intensity.quadratcount, 282
interp.colourmap, 283
interp.im, 284
intersect.boxx, 285
intersect.owin, 286
intersect.tess, 288
invoke.metric, 290
invoke.symbolmap, 291
is.boxx, 292
is.connected, 293
is.connected.ppp, 294
is.convex, 295
is.empty, 296
is.im, 297
is.linim, 298
is.linnet, 298
is.lpp, 299
is.marked, 300
is.marked.ppp, 301
is.multitype, 302
is.multitype.ppp, 303
is.na.hyperframe, 304
is.na.solist, 305
is.owin, 307
is.ppp, 307
is.rectangle, 308

is.subset.owin, 309
layered, 310
layerplotargs, 311
lengths_psp, 313
levelset, 314
levelset.distfun, 316
lut, 317
marks, 319
marks.psp, 321
marks.tess, 322
markstat, 323
matchingdist, 325
Math.im, 327
Math.imlist, 329
maxnndist, 331
mean.im, 332
mergeLevels, 333
mergeTiles, 335
methods.box3, 336
methods.boxx, 337
methods.distfun, 338
methods.funxy, 340
methods.layered, 341
methods.pp3, 342
methods.ppx, 344
methods.unitname, 345
metric.object, 346
midpoints.psp, 347
MinkowskiSum, 348
multiplicity.ppp, 350
nearest.raster.point, 352
nearestsegment, 353
nearestValue, 354
nestsplit, 355
nncross, 356
nncross.pp3, 359
nncross.ppx, 362
nndist, 364
nndist.pp3, 367
nndist.ppx, 369
nndist.psp, 371
nnfun, 373
nnmap, 374
nnmark, 376
nnwhich, 378
nnwhich.pp3, 381
nnwhich.ppx, 382
nobjects, 384
npoints, 385
nsegments, 386
nvertices, 387
opening, 388
overlap.owin, 389
owin, 390
owin.object, 393
owin2mask, 394
padimage, 397
pairdist, 398
pairdist.default, 399
pairdist.pp3, 400
pairdist.ppp, 402
pairdist.ppx, 403
pairdist.psp, 404
perimeter, 406
periodify, 407
persp.im, 408
persp.ppp, 411
perspPoints, 414
pHcolourmap, 416
pixelcentres, 417
pixellate, 418
pixellate.owin, 419
pixellate.ppp, 420
pixellate.psp, 423
pixelquad, 424
plot.anylist, 426
plot.colourmap, 429
plot.hyperframe, 431
plot.im, 433
plot.imlist, 440
plot.layered, 442
plot.listof, 443
plot.onearrow, 447
plot.owin, 448
plot.pp3, 452
plot.ppp, 453
plot.pppmatching, 461
plot.psp, 462
plot.quad, 465
plot.quadratcount, 467
plot.solist, 468
plot.splitppp, 471
plot.symbolmap, 473
plot.tess, 475
plot.textstring, 477
plot.texturemap, 478

plot.yardstick, 480
pointsOnLines, 482
polarbody, 483
polartess, 484
pp3, 485
ppp, 486
ppp.object, 490
pppdist, 492
pppmatching, 495
pppmatching.object, 496
ppx, 498
print.im, 499
print.owin, 500
print.ppp, 501
print.psp, 502
print.quad, 503
project2segment, 506
project2set, 507
psp, 509
psp.object, 510
psp2mask, 512
quad.object, 513
quadratcount, 514
quadscheme, 518
quadscheme.logi, 521
quantess, 523
quantile.im, 525
quantilefun.im, 526
quasirandom, 527
raster.x, 529
rectdistmap, 530
reflect, 531
regularpolygon, 532
relevel.im, 533
Replace.im, 534
rescale, 537
rescale.im, 539
rescale.owin, 540
rescale.ppp, 541
rescale.psp, 542
rescue.rectangle, 544
restrict.colourmap, 545
rev.colourmap, 546
rexplode, 547
rgbim, 548
ripras, 549
rjitter, 551
rjitter.pp3, 553
rlinegrid, 555
rotate, 556
rotate.im, 556
rotate.infline, 557
rotate.owin, 559
rotate.ppp, 560
rotate.psp, 561
round.ppp, 562
rounding.ppp, 563
rQuasi, 564
rsyst, 565
runifrect, 569
scalardilate, 570
scaletointerval, 571
scanpp, 572
selfcrossing.psp, 574
selfcut.psp, 575
setcov, 577
shift, 578
shift.im, 579
shift.owin, 580
shift.ppp, 581
shift.ppx, 582
shift.psp, 584
sidelengths.owin, 585
simplify.owin, 589
solapply, 591
solist, 592
solutionset, 593
spatstat.geom-package, 12
spatstat.options, 595
split.hyperframe, 600
split.im, 601
split.ppp, 602
split.ppx, 605
spokes, 607
square, 608
stratrand, 609
subset.hyperframe, 611
subset.ppp, 612
subset.psp, 614
summary.anylist, 616
summary.distfun, 617
summary.im, 618
summary.listof, 619
summary.owin, 620
summary.ppp, 621
summary.psp, 622

summary.quad, 623
summary.solist, 624
summary.splitppp, 625
superimpose, 626
SupportFun, 628
symbolmap, 630
tess, 633
test.crossing.psp, 635
text.ppp, 636
texturemap, 638
textureplot, 639
tile.areas, 640
tile.centroids, 641
tileindex, 642
tilenames, 643
tiles, 645
tiles.empty, 646
transmat, 649
triangulate.owin, 650
trim.rectangle, 651
tweak.colourmap, 652
union.quad, 653
unique.ppp, 654
uniquemap.ppp, 655
unitname, 656
unmark, 658
unstack.ppp, 659
unstack.solist, 660
update.symbolmap, 662
venn.tess, 663
vertices, 664
volume, 665
where.max, 666
whichhalfplane, 667
Window, 668
Window.tess, 670
with.hyperframe, 671
yardstick, 672
zapsmall.im, 674

∗ **univar**

mean.im, 332
quantile.im, 525
scaletointerval, 571
zapsmall.im, 674

∗ **utilities**

bounding.box.xy, 88
boundingbox, 89
convexhull, 133
convexhull.xy, 134
convexify, 135
corners, 141
dirichletWeights, 176
layout.boxes, 312
multiplicity.ppp, 350
quadrats, 517
ripras, 549
run.simplepanel, 566
simplepanel, 586
timed, 647
timeTaken, 648

[, *211*, *217*, *220*, *224*, *225*, *227*, *535*
[.anylist *(*Extract.anylist*)*, 209
[.hyperframe, *259*, *601*, *611*, *612*
[.hyperframe *(*Extract.hyperframe*)*, 210
[.im, *17*, *51*, *151*, *264*, *267*, *268*, *276*, *535*
[.im *(*Extract.im*)*, 213
[.layered, *20*, *311*, *312*, *443*
[.layered *(*Extract.layered*)*, 216
[.owin *(*Extract.owin*)*, 219
[.pp3, *613*
[.ppp, *14*, *153*, *226*, *227*, *490*, *491*, *613*
[.ppp *(*Extract.ppp*)*, 220
[.ppx, *613*
[.ppx *(*Extract.ppx*)*, 223
[.psp, *17*, *511*, *615*
[.psp *(*Extract.psp*)*, 225
[.quad *(*Extract.quad*)*, 226
[.solist *(*Extract.solist*)*, 227
[.splitppp *(*Extract.splitppp*)*, 229
[.tess, *18*, *635*
[.tess *(*Extract.tess*)*, 230
[<-.im, *17*
[<-.tess, *18*
[<-.anylist *(*Extract.anylist*)*, 209
[<-.hyperframe *(*Extract.hyperframe*)*, 210
[<-.im *(*Replace.im*)*, 534
[<-.layered *(*Extract.layered*)*, 216
[<-.listof *(*Extract.listof*)*, 217
[<-.ppp *(*Extract.ppp*)*, 220
[<-.solist *(*Extract.solist*)*, 227
[<-.splitppp *(*Extract.splitppp*)*, 229
[<-.tess *(*Extract.tess*)*, 230
[[, *212*
[[.data.frame, *212*
[[.hyperframe *(*Extract.hyperframe*)*, 210
[[<-.hyperframe *(*Extract.hyperframe*)*,

210
[[<-.layered (Extract.layered), 216
[[<-.solist (Extract.solist), 227
$, *212*
$.hyperframe (Extract.hyperframe), 210
$<-.hyperframe (Extract.hyperframe), 210
%mark% (marks), 319

abline, *271*
add.texture, 23, *449*, *478, 479*, *638–640*
affine, *14, 15*, 24, *25–30*, *237*, *339*, *342*, *393*, *531*, *538*, *541–543*, *571*, *578*, *581, 582*, *585*
affine.distfun (methods.distfun), 338
affine.im, *17*, *24*, 25, *26, 27*, *29, 30*, *557*
affine.layered (methods.layered), 341
affine.owin, *24, 25*, 26, *27–30*, *394*
affine.ppp, *24–26*, 27, *29*
affine.psp, *18*, *24–27*, 28
affine.tess, *19*, 29
aggregate, *366*
amacrine, *300–303*, *490*
angles.psp, *18*, 31, *204*, *231*, *314*, *348*, *510, 511*
anyDuplicated, *196*
anyDuplicated.ppp (duplicated.ppp), 195
anyDuplicated.ppx (duplicated.ppp), 195
anylapply, *32*
anylapply (solapply), 591
anylist, 32, *210*, *591*, *593*, *616*
anyNA, *33*
anyNA.im, 33, *333*, *619*
append.psp, 34
apply, *35–37*, *209*, *324, 325*
applynbd, *21*, 35, *109*, *324, 325*
area (area.owin), 38
area.owin, *16*, 38, *170*, *390*, *393, 394*, *406*, *586*, *665, 666*
areaGain, 39, *41*, *597*
AreaInter, *40, 41*
areaLoss, *40*, 40, *597*
array, *67*
as.anylist, *80*
as.anylist (anylist), 32
as.array.im (as.matrix.im), 67
as.box3, *19*, 42, *93*, *168*, *486*
as.boxx, 43, *272, 273*
as.colourmap, 44
as.data.frame, *45*, *48, 49*, *55, 56*
as.data.frame.default, *46, 47*, *50*
as.data.frame.hyperframe, *20*, 45, *259*
as.data.frame.im, *16*, 46, *48*, *50*
as.data.frame.owin, *15*, 47, *50*
as.data.frame.ppp, 48, *254*
as.data.frame.ppx, *254*
as.data.frame.ppx (as.hyperframe.ppx), 55
as.data.frame.psp, *18*, 49, *254*
as.data.frame.tess, 50
as.function.im, *17*, 51
as.function.owin, 52
as.function.tess, *19*, 53, *276, 277*, *635*, *643*
as.hyperframe, *19, 20*, 54, *56*, *99*, *259*
as.hyperframe.ppx, *55*, 55, *259*
as.im, *16*, 57, *68, 69*, *121*, *186*, *209*, *248, 249*, *255, 256*, *264*, *267, 268*, *340*, *374*, *418–422*, *617*, *639*, *666*
as.im.function, *340*
as.im.owin, *15*, *52*
as.im.ppp, *14*, *60*, *62*
as.im.ppp (pixellate.ppp), 420
as.layered, 63, *311*, *438*, *450*, *458*, *465*, *488*
as.layered.msr, *64*
as.mask, *15, 16*, *25, 26*, *58*, 64, *68*, *73*, *82, 83*, *91*, *95, 96*, *113*, *120, 121*, *125*, *129*, *142*, *165*, *172*, *178, 179*, *181–183*, *187–190*, *192*, *202, 203*, *206*, *232*, *240*, *244*, *282*, *286, 287*, *289*, *309*, *316*, *353*, *375*, *377*, *388*, *393–396*, *417*, *419*, *421*, *423–425*, *507, 508*, *512*, *530*, *559*, *577*, *617*
as.mask.psp, *18*
as.mask.psp (psp2mask), 512
as.matrix.im, *16*, 67, *69*, *264*, *267, 268*
as.matrix.owin, *68*, 68
as.matrix.ppx (as.hyperframe.ppx), 55
as.owin, *15*, *38, 39*, *65*, 69, *73*, *75, 76*, *78–80*, *87*, *89, 90*, *100, 101*, *113*, *133, 134*, *141*, *170, 171*, *173*, *197*, *205, 206*, *243*, *249, 250*, *269*, *274*, *287*, *309*, *315*, *389*, *391–394*, *406*, *424, 425*, *449*, *476*, *484*, *487*, *489*, *507*, *509, 510*, *517*, *544*, *550*, *555*, *565*, *569*, *577*, *586*, *596*, *609*, *626, 627*
as.owin.data.frame, *48*
as.owin.lpp, *72*
as.owin.ppm, *72*

as.owin.rmhmodel, *72*
as.polygonal, *16*, *66*, 72, *406*, *508*, *651*
as.ppp, *13*, *35*, 74, *156*, *180*, *182*, *213–215*, *454*, *488–491*, *518–523*, *534, 535*, *573*, *653, 654*
as.psp, *17*, *34*, 76, *463*, *510, 511*, *596*
as.rectangle, *66*, 79, *90*, *100*, *393, 394*, *585*
as.solist, *32*, 80, *591–593*
as.tess, *18*, 81, *289*, *514, 515*, *635*, *639*
as.tess.tessfun, *19*
atan2, *31*
axis, *429, 430*, *434, 435*, *457*, *463*, *473*, *478*
axisTicks, *434, 435*

barplot, *257*
bdist.pixels, *16*, 82, *84, 85*, *241*, *393, 394*
bdist.points, *16*, *83*, 84, *85*, *184*, *393, 394*
bdist.tiles, *16*, *19*, *83, 84*, 85, *635*
beachcolourmap, *20*, *409*, *436*
beachcolourmap *(*beachcolours*)*, 86
beachcolours, 86, *409*, *436*
betacells, *490*
blur, *355*, *596*
border, *15*, 87
bounding.box.xy, 88, *134*, *550*, *627*
boundingbox, *15*, *79, 80*, 89, *287*, *393, 394*
boundingcentre *(*boundingcircle*)*, 91
boundingcircle, 91
boundingradius *(*boundingcircle*)*, 91
box3, *19*, *42*, 92, *246*, *336*, *346*, *486*
boxx, *19*, *43*, 93, *169*, *246*, *272, 273*, *286*, *293*, *338*, *583*
bramblecanes, *490*
bufftess, 94, *485*, *518*, *634*
bw.relrisk, *597*
by, *96*, *98*
by.im, 96, *602*
by.ppp, *14*, 97, *634, 635*

c, *310*, *342*
c.layered *(*methods.layered*)*, 341
cbind, *99*
cbind.hyperframe, *20*, 99, *259*
cells, *490*
centroid.owin, *16*, 100, *270*, *581*, *641*
chop.tess, *18*, 101, *107*, *272*
chull, *295*
clear.simplepanel *(*run.simplepanel*)*, 566
clickbox, *15*, 102, *104–106*
clickdist, *16*, *103*, 103, *105, 106*
clickpoly, *15*, *103, 104*, 104, *106*
clickppp, *13*, *103–105*, 105, *260*
clip.infline, *102*, 106, *272*
clip.psp, *511*
closepairs, 107, *111, 112*, *399, 400*, *599*
closepairs.pp3, *109*, 109
closetriples, 111
closing, *15*, 112, *389*, *393, 394*
cm.colors, *409*, *437*
col2hex *(*colourtools*)*, 118
col2rgb, *118*, *120*
colourmap, *20*, *44*, *87*, 114, *117*, *120*, *131*, *156*, *161*, *284*, *319*, *409*, *417*, *430, 431*, *434*, *436*, *439*, *464*, *474*, *476*, *545, 546*, *631*, *653*
colouroutputs, *116*, 116, *546*, *653*
colouroutputs<- *(*colouroutputs*)*, 116
colours, *116*
colourtools, *87*, *116, 117*, 118, *284*, *549*, *653*
commonGrid, *15*, *17*, 120, *123*, *248–250*
compatible, 121, *247, 248*, *345*, *657*
compatible.fasp, *122*
compatible.fv, *122*
compatible.im, *17*, *121, 122*, 122, *209*, *249*
compatible.unitname, *122*
compatible.unitname *(*methods.unitname*)*, 345
complement.owin, *15*, 123, *296*, *392–395*
complementarycolour *(*colourtools*)*, 118
Complex.im, *333*
Complex.im *(*Math.im*)*, 327
Complex.imlist *(*Math.imlist*)*, 329
concatxy, 124, *628*
connected, 125, *293, 294*
connected.im, *17*, *129*
connected.owin, *16*, *129, 130*
connected.pp3 *(*connected.ppp*)*, 128
connected.ppp, *14*, *126, 127*, 128, *295*
connected.tess, *19*, *126, 127*, 129
contour, *340*
contour.default, *60*, *130, 131*, *435*
contour.funxy *(*methods.funxy*)*, 340
contour.im, *17*, 130, *132, 133*, *340*, *410*, *439*, *597*
contour.imlist, 132
contour.listof, *428*, *446*, *471*
contour.listof *(*contour.imlist*)*, 132

convexhull, *14–16*, 133, *134*, *136*
convexhull.xy, *89*, *133*, 134, *163*, *296*, *550*
convexify, 135
convexmetric, 136, *291*, *347*
convolve.im, *17*, 138, *269*
coords, *14*, *19*, 139
coords.ppx, *148*, *370*, *383*, *403*
coords<- *(*coords*)*, 139
corners, 141, *157*, *519, 520*
covering, 142
crossdist, *21*, 143, *144–149*, *399–401*, *403–405*
crossdist.default, *143*, 144, *147*
crossdist.pp3, *21*, 145
crossdist.ppp, *143*, *145*, 146
crossdist.ppx, *21*, 147
crossdist.psp, *143*, *145*, *147*, 148
crossing.psp, *18*, 150, *574*, *599*
crosspairs, *599*
crosspairs *(*closepairs*)*, 107
crosspairs.pp3 *(*closepairs.pp3*)*, 109
cut, *151–153*
cut.default, *95*, *151–153*
cut.im, *17*, *95, 96*, 151, *264*, *267*, *525*
cut.ppp, *14*, *53*, *98*, 152, *221*, *604*, *634, 635*, *643*

data.frame, *59*, *61*, *258, 259*
default.colourmap, 154
default.dummy, 156, *519*, *597*
default.image.colours, 157, *439*
default.n.tiling, *157*
default.symbolmap, 158, *162*
default.symbolmap.ppp, *159*, 159, *459*
delaunay, *14*, *18*, *96*, 163, *164*, *174*, *485*, *518*, *634, 635*, *651*, *663*
delaunayDistance, *14*, *163*, 164
deldir, *174*
deltametric, 165
demopat, *490*
density.ppp, *16*, *60*, *422*, *596*
density.psp, *596*
density.splitppp, *428*, *446*, *471*
densityfun, *60*
dev.capabilities, *437*
dev.new, *567*
dev.off, *567*
dev.set, *567*
diameter, *92*, 166, *394*
diameter.box3, *19*, *93*, *167*, 167
diameter.boxx, *19*, *94*, *167*, 168
diameter.owin, *16*, *39*, *167*, 170, *393*, *406*, *586*
dilated.areas, *16*, *41*, 171
dilation, *15*, *87, 88*, *96*, *113*, *171*, 172, *206*, *247*, *349*, *388, 389*, *393, 394*
dilationAny, *173*
dilationAny *(*MinkowskiSum*)*, 348
dirichlet, *14*, *18*, *96*, *163, 164*, 173, *175, 176*, *485*, *518*, *634, 635*, *663*
dirichletAreas, 174, *176*
dirichletEdges *(*dirichletVertices*)*, 175
dirichletVertices, *174, 175*, 175
dirichletWeights, 176, *245*, *520*
disc, *15*, *172*, 178, *180*, *183, 184*, *203*, *391, 392*, *487*, *533*
discpartarea, 180
discretise, *14*, 181
discs, *179*, 182
distbdry, 184
distfun, *21*, *60*, 185, *188*, *191–193*, *316*, *339, 340*, *374*, *617*
distfun.owin, *16*
distfun.psp, *18*
distmap, *21*, *40*, *60*, *95, 96*, *165, 166*, *171*, *186*, 187, *189*, *191–193*, *267*, *270*, *376*, *530*
distmap.owin, *16*, *83*, *187, 188*, 188, *191*, *193*
distmap.ppp, *187–189*, 190, *193*
distmap.psp, *18*, *187–189*, *191*, 192, *354*, *511*
domain, 193, *194*
domain.lpp, *195*
domain.ppm, *195*
domain.quadrattest, *195*
domain.rmhmodel, *195*
dppeigen, *596*
duplicated, *196*
duplicated.data.frame, *196*, *654*
duplicated.ppp, *14*, 195, *351*, *621*, *654–656*
duplicated.ppx *(*duplicated.ppp*)*, 195
duplicatedxy, *196*

ecdf, *527*
edge.Ripley, *597*
edge.Trans, *597*
edges, *16, 17*, *78*, *170*, 196
edges2triangles, 197, *199*
edges2vees, *198*, 199

edit, *200*, *201*, *254*
edit.data.frame, *200–202*
edit.hyperframe, 200, *202*
edit.im *(*edit.ppp*)*, 201
edit.ppp, *14*, *200*, 201
edit.psp *(*edit.ppp*)*, 201
ellipse, *15*, *179*, 202, *392*, *487*, *533*
endpoints.psp, *18*, *31*, 203, *231*, *314*, *348*, *510*
eroded.areas, *16*, *171*, 204, *206*, *393*, *394*
eroded.volumes, *19*, *93*, *94*
eroded.volumes *(*diameter.box3*)*, 167
eroded.volumes.boxx, *20*, *94*
eroded.volumes.boxx *(*diameter.boxx*)*, 168
erosion, *15*, *83*, *84*, *87*, *88*, *113*, *173*, *205*, 205, *207*, *247*, *269*, *296*, *388*, *389*, *393*, *394*, *577*, *652*
erosionAny, *206*, 207, *349*
eval.im, *17*, *123*, 208, *264*, *266–268*, *276*, *327–330*, *539*, *594*, *667*
ewcdf, *527*
exactdt, *21*
expression, *432*
Extract.anylist, 209
Extract.hyperframe, 210
Extract.im, 213
Extract.layered, 216
Extract.listof, 217
Extract.owin, 219
Extract.ppp, 220
Extract.ppx, 223
Extract.psp, 225
Extract.quad, 226
Extract.solist, 227
Extract.splitppp, 229
Extract.tess, 230
extrapolate.psp, *18*, *31*, *204*, 231, *314*, *348*, *510*

F3est, *597*
factor, *334*
fardist, 232
FeretBox, 233, *235*
FeretDiamFun, *234*, 234, *629*
fillholes.owin, 235, *590*
flipxy, *14*, *15*, *18*, *24*, *27*, *29*, *30*, 236, *237*, *339*, *342*, *531*, *558*
flipxy.distfun *(*methods.distfun*)*, 338
flipxy.infline *(*rotate.infline*)*, 557
flipxy.layered *(*methods.layered*)*, 341
flipxy.tess, *19*
flipxy.tess *(*affine.tess*)*, 29
fourierbasis, 237
fourierbasisraw *(*fourierbasis*)*, 237
Frame, *15*, *195*, 239
Frame<- *(*Frame*)*, 239
framedist.pixels, *83*, 240
funxy, 241, *340*, *617*

G3est, *146*, *368*, *369*
ganglia, *490*
Gcom, *596*
Gest, *145*, *147*, *149*, *366*
Gres, *596*
gridcenters *(*gridcentres*)*, 242
gridcentres, *157*, 242, *519*, *520*, *608*, *610*
gridweights, *177*, 244, *520*
grow.box3 *(*grow.boxx*)*, 245
grow.boxx, 245
grow.rectangle, *246*, 246, *652*
grow.simplepanel, *567*
grow.simplepanel *(*simplepanel*)*, 586

Halton, *564*, *565*
Halton *(*quasirandom*)*, 527
Hammersley, *564*
Hammersley *(*quasirandom*)*, 527
harmonise, 247, *248*, *249*, *346*
harmonise.fv, *247*, *248*
harmonise.im, *17*, *121*, *123*, *209*, *247*, *248*, 248, *250*, *266*, *327*, *329*
harmonise.owin, 249
harmonise.unitname *(*methods.unitname*)*, 345
harmoniseLevels, 250
harmonize *(*harmonise*)*, 247
harmonize.im *(*harmonise.im*)*, 248
harmonize.owin *(*harmonise.owin*)*, 249
harmonize.unitname *(*methods.unitname*)*, 345
has.close, 251
head, *253*
head.hyperframe, *20*
head.ppp *(*headtail*)*, 252
head.ppx *(*headtail*)*, 252
head.psp *(*headtail*)*, 252
head.tess *(*headtail*)*, 252
headtail, 252

heat.colors, *409*, *437*
hexagon, *255*, *392*, *487*
hexagon *(*regularpolygon*)*, 532
hexgrid *(*hextess*)*, 254
hextess, *18*, *96*, 254, *485*, *516*, *518*, *533*, *634*, *635*, *663*
hist, *256*, *257*
hist.default, *256*, *257*
hist.funxy, 255
hist.im, *17*, *255*, *256*, 256, *267*, *439*
hsv, *549*
hsvim, *17*
hsvim *(*rgbim*)*, 548
hyperframe, *20*, *45*, *54–56*, *99*, *212*, 258, *272*, *320*, *431*, *432*, *498*, *601*, *672*

identify, *260–263*
identify.default, *260*
identify.ppp, *14*, *105*, *106*, 260, *262*
identify.psp, 261
identify.tess, 262
im, *13*, *16*, *97*, *98*, 263, *267*, *268*, *397*, *422*, *539*, *602*, *604*, *640*
im.apply, *17*, *209*, 265
im.object, *33*, *59*, *127*, *129*, *131*, *151*, *152*, *208*, *209*, *214*, *215*, *257*, *264*, 267, *297*, *315*, *333*, *409*, *410*, *434*, *439*, *500*, *525*, *535*, *549*, *593*, *594*, *618*
image, *597*
image.default, *60*, *429*, *434–439*, *449–451*, *649*
image.im *(*plot.im*)*, 433
image.imlist *(*plot.imlist*)*, 440
image.listof, *427*, *428*, *445*, *446*, *470*, *471*
image.listof *(*plot.imlist*)*, 440
imcov, *17*, *139*, 268, *577*
incircle, *16*, 269
infline, *102*, *107*, *231*, 271, *558*, *667*
inradius, *16*
inradius *(*incircle*)*, 269
inside.boxx, 272
inside.owin, *16*, *243*, 273, *608*, *610*
integral, *275–277*
integral.im, *17*, 275, *619*
integral.tessfun, *19*, *53*, 276
intensity, *21*, 277, *279–283*
intensity.ppm, *278*, *279*
intensity.ppp, *278*, 278
intensity.ppx, 280
intensity.psp, 281
intensity.quadratcount, 282, *515*, *516*
intensity.splitppp *(*intensity.ppp*)*, 278
interp.colourmap, *20*, *116*, *117*, *120*, 283, *653*
interp.colours *(*colourtools*)*, 118
interp.im, *17*, 284
intersect.boxx, 285
intersect.owin, *16*, *219*, *286*, 286, *289*, *296*, *390*
intersect.tess, *18*, *231*, 288, *484*, *485*, *635*, *663*
invoke.metric, *137*, 290, *347*
invoke.symbolmap, 291, *474*, *632*
is.boxx, 292, *338*
is.colour *(*colourtools*)*, 118
is.connected, 293, *295*
is.connected.ppp, *294*, 294
is.convex, *16*, *133*, 295
is.empty, *287*, 296
is.grey *(*colourtools*)*, 118
is.im, *17*, 297
is.linim, 298
is.linnet, 298
is.lpp, 299
is.marked, 300, *301*
is.marked.ppm, *300*, *301*
is.marked.ppp, *300*, 301
is.mask, *16*
is.mask *(*is.rectangle*)*, 308
is.multitype, 302, *303*
is.multitype.ppm, *302*, *303*
is.multitype.ppp, *302*, 303
is.na.hyperframe, 304, *351*, *352*
is.na.solist, 305, *351*, *352*
is.NAobject, 306, *351*, *352*
is.owin, 307
is.polygonal, *16*
is.polygonal *(*is.rectangle*)*, 308
is.ppp, 307
is.psp, *17*
is.rectangle, *16*, 308
is.subset.owin, *16*, *287*, 309

K3est, *401*
Kcom, *596*, *597*
Kcross, *109*
Kest, *109*, *399*, *400*, *403*, *597*
Kmeasure, *16*, *267*

Kmodel, *596*
Kovesi, *158*, *437*
Kres, *596*, *597*

lansing, *490*
lapply, *330*, *591*
layered, *20*, *64*, *71*, *217*, 310, *311*, *312*, *342*, *438*, *442*, *443*, *450*, *458*, *465*, *673*
layerplotargs, *311*, 311, *438*, *443*, *450*, *458*, *465*
layerplotargs<- *(*layerplotargs*)*, 311
layout.boxes, 312, *587*, *588*
lengths, *314*
lengths_psp, *18*, *31*, *204*, *231*, 313, *348*, *510*, *511*
letterR, *15*
levels, *251*
levels.im, *251*
levelset, *17*, 314, *316*, *594*
levelset.distfun, 316
levelset.im, *316*
lgcp.estK, *596*
lineardirichlet, *174*
lines, *414*
locator, *103–106*
longleaf, *300–303*, *490*
lut, *116*, 317

marks, *14*, 319, *321–323*
marks.psp, *18*, *31*, *204*, *314*, 321, *348*, *510*
marks.tess, 322, *634*, *635*
marks<-, *13*
marks<-.psp, *18*
marks<- *(*marks*)*, 319
marks<-.psp *(*marks.psp*)*, 321
marks<-.tess *(*marks.tess*)*, 322
markstat, *21*, *36*, *37*, *109*, 323
marktable, *36*, *37*, *324*, *325*, *378*
matchingdist, 325, *494–497*
Math.im, 327, *330*, *333*, *594*, *667*
Math.imlist, 329
matrix, *264*
maxFeretDiam *(*FeretDiamFun*)*, 234
maxnndist, 331, *366*
mean, *333*
mean.im, *17*, *267*, 332, *619*
median, *333*
median.im *(*mean.im*)*, 332
mergeLevels, *251*, 333, *534*
mergeTiles, 335
methods.box3, 336
methods.boxx, *293*, 337
methods.distfun, *186*, *316*, 338
methods.funxy, *186*, *339*, 340
methods.layered, *311*, *312*, 341
methods.pp3, 342
methods.ppx, 344
methods.unitname, 345
metric.object, *136*, *137*, *146*, *188*, *190*, *192*, *290*, 346, *357*
midpoints.psp, *18*, *31*, *204*, *231*, *314*, 347, *510*, *511*
minFeretDiam *(*FeretDiamFun*)*, 234
MinkowskiSum, *207*, 348
minnndist, *366*
minnndist *(*maxnndist*)*, 331
miplot, *516*
mtext, *434*, *436*, *457*
multiplicity *(*multiplicity.ppp*)*, 350
multiplicity.ppp, *196*, 350, *489*, *655*

NAobject, *304–306*, 351
nearest.raster.point, *15*, 352, *393*, *394*
nearestsegment, *18*, *193*, 353, *507*
nearestValue, 354
nestsplit, 355
nncross, *18*, *21*, *109*, *191–193*, 356, *366*, *368*, *370*, *380–383*, *508*
nncross.pp3, *21*, 359
nncross.ppx, 362
nndist, *21*, *109*, *143*, *145–149*, *252*, *331*, *359*, *361*, *363*, 364, *369*, *371–373*, *380–383*, *399–401*, *403–405*
nndist.pp3, *21*, *365*, *366*, 367
nndist.ppp, *290*, *373*
nndist.ppx, *21*, *365*, *366*, 369
nndist.psp, *365*, *366*, 371
nnfun, *21*, *60*, 373
nnmap, *21*, *60*, 374
nnmark, *14*, 376
nnwhich, *21*, *365*, *366*, *368–371*, *378*, 378, *382*, *383*
nnwhich.pp3, *21*, *379*, 381
nnwhich.ppx, *21*, 382
nobjects, 384
npoints, *14*, *19*, *385*, 385, *386*
nsegments, 386
nvertices, 387

nztrees, *490*

onearrow, *447, 448*, *478*
onearrow *(*yardstick*)*, 672
opening, *15*, *113*, *296*, 388, *393, 394*
Ops.im *(*Math.im*)*, 327
Ops.imlist *(*Math.imlist*)*, 329
options, *596*, *599*
overlap.owin, *287*, 389
owin, *13*, *15*, *23*, *38*, *72, 73*, *80*, *89, 90*, *101*, *113*, *123, 124*, *133, 134*, *141*, *170, 171*, *173*, *179, 180*, *203*, *205, 206*, *225*, *236*, *243*, *269*, *274*, *296*, *309*, *389*, 390, *393, 394*, *449*, *487–491*, *509–511*, *530*, *533*, *550*, *565*, *577*, *586*, *590*, *596*, *609*, *657*
owin.object, *39*, *59*, *66*, *70*, *72*, *76*, *78*, *83*, *100*, *123, 124*, *170*, *179*, *203*, *205*, *215*, *221*, *226*, *247*, *274*, *287*, *307*, *315*, *352, 353*, *391, 392*, 393, *406*, *451*, *489*, *510*, *535*, *544*, *560*, *570*, *577*, *594*, *609*, *652*, *664, 665*, *670, 671*
owin2mask, *59*, *61*, *66*, *213*, 394, *419–424*

padimage, 397
pairdist, *21*, *143*, *145–149*, *366*, *369*, *371*, 398, *401*, *403–405*
pairdist.default, *398, 399*, 399, *403*
pairdist.pp3, *21*, 400
pairdist.ppp, *398, 399*, 402, *405*
pairdist.ppx, *21*, 403
pairdist.psp, *398, 399*, *403*, 404
palette, *119, 120*
paletteindex *(*colourtools*)*, 118
par, *104*, *106*, *160*, *428*, *432*, *437–439*, *446*, *451*, *454*, *457–459*, *465, 466*, *471*, *631*
parent.frame, *671*
pdf.options, *438*
perimeter, *16*, *39*, *170*, *197*, *393*, 406, *586*
periodify, *14, 15*, *18*, 407, *578*, *581, 582*, *585*
persp, *340*, *413*, *597*
persp.default, *409, 410*, *412, 413*
persp.funxy *(*methods.funxy*)*, 340
persp.im, *17*, *131*, *267, 268*, *340*, 408, *414, 415*, *439*
persp.ppp, *14*, 411
perspContour *(*perspPoints*)*, 414
perspLines, *410*
perspLines *(*perspPoints*)*, 414
perspPoints, *410*, 414
perspSegments *(*perspPoints*)*, 414
pHcolour *(*pHcolourmap*)*, 416
pHcolourmap, 416
pictex, *450*
pixelcentres, *16, 17*, 417, *530*
pixellate, *16*, 418, *420–424*
pixellate.linnet, *66*
pixellate.owin, *15*, *418, 419*, 419
pixellate.ppp, *14*, *66*, *418–420*, 420
pixellate.psp, *18*, *66*, *418, 419*, 423, *512*
pixelquad, 424
plot, *340*, *344*, *426*, *432*, *443, 444*, *459*, *465*, *469*, *476*, *479*, *513*
plot.anylist, *32*, *210*, 426, *471*, *606*, *616*
plot.colourmap, *20*, *115, 116*, *427*, 429, *469*
plot.default, *437*, *449, 450*, *454*
plot.funxy, *186*, *242*, *374*
plot.funxy *(*methods.funxy*)*, 340
plot.fv, *597, 598*
plot.hyperframe, *20*, *259*, 431, *672*
plot.im, *17*, *131*, *158*, *215*, *267, 268*, *340*, *410*, 433, *441*, *476*, *597*, *640*
plot.imlist, *438*, 440
plot.infline *(*infline*)*, 271
plot.layered, *20*, *310, 311*, 442
plot.linim, *158*
plot.listof, *218*, 443, *456*, *472*, *620*
plot.onearrow, 447, *673*
plot.owin, *15*, *23*, *262*, *393, 394*, *412*, *438*, 448, *456, 457*, *459*, *461*, *463–465*, *476*, *597*
plot.pp3, *19*, 452, *597*
plot.ppm, *597*
plot.ppp, *14*, *160*, *162*, *260*, *451*, 453, *466*, *472*, *488*, *490*, *597, 598*
plot.pppmatching, 461, *494*, *497*
plot.ppx *(*methods.ppx*)*, 344
plot.psp, *17*, *261*, *461*, 462, *511*
plot.quad, 465, *503*, *513*
plot.quadratcount, 467, *516*
plot.quadrattest, *468*
plot.solist, *132, 133*, *228*, *441*, 468, *476*, *592*, *624*
plot.splitppp, *229*, 471, *603, 604*
plot.symbolmap, *292*, *427*, *454*, *469*, 473, *632*

plot.tess, *18*, *262, 263*, *467, 468*, 475, *635*
plot.tessfun, *19*
plot.textstring, 477, *673*
plot.texturemap, 478
plot.yardstick, 480, *673*
points, *160*, *414, 415*, *452*, *454*, *456–459*, *473*
points.default, *412*
pointsOnLines, *18*, 482
polarbody, 483, *629*
polartess, *18*, *96*, 484, *518*, *634, 635*, *663*
polyclip, *286*
polygon, *102*, *104*, *449–451*, *461*, *464*
polynom, *596*
polypath, *449, 450*
pp3, *13*, *19*, *42*, *93*, *140*, *343*, *453*, 485, *499*
ppm, *425*, *513, 514*, *519–523*, *597*
ppm.ppp, *519*
ppp, *13*, *74–76*, *98*, *140*, *163*, *174*, *182*, *320*, *392*, *458*, *483*, 486, *490, 491*, *573*, *626*, *657*
ppp.object, *37*, *74–76*, *84*, *151*, *153*, *157*, *196*, *204*, *215*, *221*, *308*, *320*, *325*, *351*, *377*, *386*, *392*, *455*, *459*, *472*, *488, 489*, 490, *535*, *561*, *570*, *573, 574*, *604*, *654, 655*, *659*
pppdist, *21*, *326*, 492, *495*, *497*
pppmatching, 495, *497*
pppmatching.object, *326*, *461, 462*, *494*, *496*, 496
ppx, *13*, *19*, *56*, *94*, *140*, *224*, *272*, *320*, *345*, *486*, 498, *527*, *583*, *606*
predict.ppm, *597*
print, *271*, *336–338*, *343–346*, *500–502*
print.box3 *(*methods.box3*)*, 336
print.boxx *(*methods.boxx*)*, 337
print.default, *345*
print.im, *267*, 499
print.infline *(*infline*)*, 271
print.listof, *445, 446*
print.owin, *394*, 500, *501, 502*, *620*
print.pp3, *386*, *486*
print.pp3 *(*methods.pp3*)*, 342
print.ppm, *597*
print.ppp, *501*, 501, *598*, *621*
print.ppx, *386*, *499*
print.ppx *(*methods.ppx*)*, 344
print.psp, *17*, 502, *622*
print.quad, 503
print.summary.im *(*summary.im*)*, 618
print.summary.pp3 *(*methods.pp3*)*, 342
print.summary.quad *(*summary.quad*)*, 623
print.unitname *(*methods.unitname*)*, 345
proc.time, *647*
progressreport, 504, *598*
project.ppm, *598*
project2segment, *18*, *192, 193*, *354*, 506, *508*
project2set, 507
ps.options, *438*
psp, *13*, *17*, *31*, *34*, *77, 78*, *107*, *231*, *483*, 509, *511*, *555*, *596*, *626*, *636*
psp.object, *77, 78*, *151*, *204*, *226*, *322*, *386*, *464, 465*, *510*, 510, *562*, *574*, *659*
psp2mask, *66*, *424*, 512
psst, *596*
psstA, *596*, *598*
psstG, *596*, *598*

quad.object, *75*, *141*, *157*, *177*, *227*, *243–245*, *425*, *466*, *503*, 513, *519, 520*, *608*, *610*, *623*, *653, 654*
quadrat.test, *81*, *515, 516*, *518*
quadratcount, *21*, *81, 82*, *282, 283*, *467, 468*, 514, *518*
quadratresample, *516*, *518*
quadrats, *18*, *96*, *355*, *485*, *516*, 517, *523, 524*, *634, 635*, *663*
quadscheme, *125*, *141*, *157*, *243*, *425*, *503*, *513, 514*, 518, *566*, *607, 608*, *610*, *628*
quadscheme.logi, 521
quantess, *18*, *96*, *356*, *485*, *518*, 523, *634, 635*, *663*
quantile, *524–527*
quantile.default, *523*, *525, 526*
quantile.ewcdf, *527*
quantile.im, *17*, *267*, *333*, 525
quantilefun, *116*, *318*, *527*, *632*
quantilefun.im, 526
quasirandom, 527
quote, *431, 432*

rainbow, *155*
raster.x, *15*, *391*, *393, 394*, 529
raster.xy, *16*, *418*
raster.xy *(*raster.x*)*, 529
raster.y, *16*, *391*, *393, 394*
raster.y *(*raster.x*)*, 529

rasterImage, *437*, *438*
rbind, *99*
rbind.hyperframe, *20*, *259*
rbind.hyperframe *(*cbind.hyperframe*)*, 99
read.table, *490*, *573*
rectdistmap, 530
redraw.simplepanel *(*run.simplepanel*)*, 566
redwood, *490*
reflect, *14*, *24*, *30*, *138*, *139*, *237*, *339*, *342*, 531, *558*
reflect.default, *30*
reflect.distfun *(*methods.distfun*)*, 338
reflect.im, *30*
reflect.infline *(*rotate.infline*)*, 557
reflect.layered *(*methods.layered*)*, 341
reflect.tess, *19*
reflect.tess *(*affine.tess*)*, 29
regularpolygon, *392*, *487*, 532
relevel, *334*, *533*
relevel.im, 533
relevel.ppp *(*relevel.im*)*, 533
relevel.ppx *(*relevel.im*)*, 533
Replace.im, 534
requireversion, 536
rescale, *191*, *279*, *281*, *339*, *341*, *342*, *345*, 537, *539–543*, *657*
rescale.distfun *(*methods.distfun*)*, 338
rescale.im, *538*, 539
rescale.layered, *538*
rescale.layered *(*methods.layered*)*, 341
rescale.owin, *538*, 540, *541*, *542*
rescale.ppp, *538*, 541
rescale.psp, *538*, 542
rescale.unitname, *538*
rescale.unitname *(*methods.unitname*)*, 345
rescue.rectangle, *26*, *287*, 544, *559*
reset.default.image.colours *(*default.image.colours*)*, 157
reset.spatstat.options *(*spatstat.options*)*, 595
restrict.colourmap, *116*, 545
rev, *546*
rev.colourmap, *116*, 546
rexplode, 547, *552*
rgb, *118*, *119*, *155*, *549*
rgb2hex *(*colourtools*)*, 118
rgb2hsv, *119*, *120*
rgb2hsva *(*colourtools*)*, 118
rgbim, *17*, 548
ripras, *15*, *89*, *134*, 549, *573*, *627*
rjitter, *14*, 551, *554*
rjitter.pp3, *552*, 553
rjitter.ppp, *547*, *548*
rlinegrid, *18*, 555
rMatClust, *491*
rMaternI, *491*
rMaternII, *491*
rmh, *71*, *596*, *598*, *621*
rmh.default, *598*
rmhcontrol, *596*
rmhcontrol.default, *598*
rNeymanScott, *491*
rotate, *14*, *15*, *24–27*, *29*, *30*, *237*, *339*, *342*, *393*, *538*, *541–543*, 556, *557*, *558*, *578*, *581*, *582*, *585*
rotate.distfun *(*methods.distfun*)*, 338
rotate.im, *17*, *30*, 556
rotate.infline, *272*, 557
rotate.layered *(*methods.layered*)*, 341
rotate.owin, *30*, *394*, *556*, 559, *560–562*
rotate.ppp, *556*, 560, *562*
rotate.psp, *18*, 561
rotate.tess, *19*
rotate.tess *(*affine.tess*)*, 29
round, *562–564*
round.pp3 *(*round.ppp*)*, 562
round.ppp, 562, *564*
round.ppx *(*round.ppp*)*, 562
rounding, *563*, *564*
rounding.pp3 *(*rounding.ppp*)*, 563
rounding.ppp, *563*, 563
rounding.ppx *(*rounding.ppp*)*, 563
rpoint, *570*
rpoisline, *555*
rpoislinetess, *96*, *485*, *518*, *634*, *635*, *663*
rpoispp, *491*, *570*, *596*
rQuasi, *528*, 564
rSSI, *491*
rstrat, *566*
rsyst, *14*, 565
rthin, *596*
rThomas, *491*
run.simplepanel, 566, *587*, *588*
runifpoint, *491*, *566*, *569*, *570*, *597*
runifpointOnLines, *483*

runifrect, *14*, 569

samecolour *(colourtools)*, 118
scalardilate, *14*, *30*, *339*, *342*, 570
scalardilate.distfun *(methods.distfun)*, 338
scalardilate.im, *30*
scalardilate.layered *(methods.layered)*, 341
scalardilate.owin, *30*
scalardilate.tess *(affine.tess)*, 29
scale, *337, 338*, *344*, *572*
scale.boxx *(methods.boxx)*, 337
scale.default, *337*, *344*
scale.ppx *(methods.ppx)*, 344
scaletointerval, *17*, 571
scanpp, *490*, 572
segments, *412*, *414, 415*, *447*, *452*, *463, 464*, *476*, *480*
selfcrossing.psp, *18*, *151*, 574, *576*, *599*
selfcut.psp, *18*, 575
sessionInfo, *576*
sessionLibs, 576
set.seed, *155*, *162*, *458*, *465*
setcov, *16*, *267*, *269*, *390*, 577, *596*
setmarks *(marks)*, 319
setminus.owin, *16*, *96*
setminus.owin *(intersect.owin)*, 286
shift, *14, 15*, *24–27*, *29, 30*, *237*, *339*, *342*, *393*, *408*, *428*, *446*, *470*, *538*, *541–543*, *558*, *571*, 578, *579–585*, *673*
shift.boxx *(shift.ppx)*, 582
shift.distfun *(methods.distfun)*, 338
shift.im, *17*, *30*, *557*, 579
shift.infline *(rotate.infline)*, 557
shift.layered *(methods.layered)*, 341
shift.owin, *30*, *394*, *578*, 580, *582*, *585*
shift.ppp, *578*, *581*, 581, *585*
shift.ppx, 582
shift.psp, *18*, 584
shift.tess, *19*
shift.tess *(affine.tess)*, 29
shortside, *586*
shortside *(diameter.box3)*, 167
shortside.box3, *19*
shortside.boxx, *19*
shortside.boxx *(diameter.boxx)*, 168
shortside.owin *(sidelengths.owin)*, 585
sidelengths, *586*
sidelengths *(diameter.box3)*, 167
sidelengths.boxx *(diameter.boxx)*, 168
sidelengths.owin, 585
simdat, *490*
simplepanel, *313*, *567, 568*, 586
simplify.owin, *15*, *73*, *135*, *236*, *406*, 589
Smooth.ppp, *60*, *355*, *378*, *422*, *596*
Smoothfun, *60*
solapply, *80*, *330*, 591, *593*
solist, *32*, *80*, *228*, *591*, 592, *624*, *673*
solutionset, *17*, *268*, *315*, 593, *667*
spatdim, 594
spatialcdf, *256, 257*
spatstat.geom *(spatstat.geom-package)*, 12
spatstat.geom-package, 12
spatstat.options, *15, 16*, *61*, *66*, *131*, *162*, *266*, *340*, *439*, *450, 451*, *453*, *457*, 595
split, *600, 601*, *606*, *625*, *660*
split.hyperframe, *259*, 600
split.im, *97*, *592*, 601
split.ppp, *14*, *53*, *64*, *98*, *221*, *229*, *355, 356*, *366*, *380*, *472*, *515*, *592*, 602, *625*, *634, 635*, *643*, *660*
split.ppx, 605
split<-.hyperframe *(split.hyperframe)*, 600
split<-.ppp *(split.ppp)*, 602
spokes, *157*, *519, 520*, 607
square, *15*, *391, 392*, *487*, 608
stratrand, *157*, *243*, *519, 520*, *608*, 609
subset, *611–613*, *615*
subset.hyperframe, *20*, 611
subset.pp3, *19*
subset.pp3 *(subset.ppp)*, 612
subset.ppp, *14*, *153*, *221*, 612
subset.ppx, *19*
subset.ppx *(subset.ppp)*, 612
subset.psp, *17*, 614
summary, *17*, *20*, *343*, *345*, *616–625*
summary.anylist, *210*, 616
summary.distfun, *186*, 617
summary.funxy, *242*
summary.funxy *(summary.distfun)*, 617
Summary.im, *333*, *667*
Summary.im *(Math.im)*, 327

summary.im, *257*, *267*, *333*, *500*, *617*, 618
Summary.imlist *(*Math.imlist*)*, 329
summary.listof, *218*, 619
summary.owin, *394*, *501*, 620, *621, 622*
summary.pp3 *(*methods.pp3*)*, 342
summary.ppp, *501*, *620*, 621
summary.psp, *17*, *31*, *204*, *314*, *348*, *502*, *510*, 622
summary.quad, *503*, 623
summary.solist, *228*, 624
summary.splitppp, *229*, 625
summary.unitname *(*methods.unitname*)*, 345
superimpose, *14*, *18*, *34*, *124, 125*, *490*, *604*, 626
SupportFun, *235*, *484*, 628
Sweave, *576*
swedishpines, *490*
symbolmap, *160*, *162*, *292*, *454–457*, *474*, 630, *662*
symbols, *160*, *454*, *456–459*, *465*, *473*

tail, *253*
tail.hyperframe, *20*
tail.ppp *(*headtail*)*, 252
tail.ppx *(*headtail*)*, 252
tail.psp *(*headtail*)*, 252
tail.tess *(*headtail*)*, 252
terrain.colors, *409*, *437*
tess, *13*, *18*, *82*, *85*, *96–98*, *127*, *129*, *153*, *163*, *174*, *230, 231*, *255*, *289*, *335*, *355*, *476, 477*, *485*, *516–518*, *523, 524*, *602*, *604*, 633, *640–642*, *644–646*, *651*, *663*
test.crossing.psp, 635
test.selfcrossing.psp *(*test.crossing.psp*)*, 635
text, *477, 478*, *480, 481*, *637*
text.default, *261, 262*, *463*, *467, 468*, *476*, *637*
text.ppp, *459*, 636
text.psp, *465*
text.psp *(*text.ppp*)*, 636
textstring, *477, 478*
textstring *(*yardstick*)*, 672
texturemap, *23*, *479*, 638
textureplot, *23*, *479*, *638*, 639
tile.areas, *18*, *634, 635*, 640, *642*, *645, 646*
tile.centroids, 641
tileindex, *53*, 642
tilenames, *524*, *634*, *641*, 643, *645, 646*
tilenames<- *(*tilenames*)*, 643
tiles, *18*, *231*, *634, 635*, *641*, *644*, 645, *646*
tiles.empty, *641*, *645*, 646
timed, 647, *648, 649*
timeTaken, *648*, 648
to.grey, *438*
to.grey *(*colourtools*)*, 118
to.opaque *(*colourtools*)*, 118
to.saturated *(*colourtools*)*, 118
to.transparent *(*colourtools*)*, 118
topo.colors, *409*, *437*
trans3d, *410*
transmat, *17*, 649
triangulate.owin, *16*, 650
trim.rectangle, *247*, 651
Tstat, *112*
tweak.colourmap, *20*, *116, 117*, *120*, *284*, 652
txtProgressBar, *505*

union.owin, *16*, *184*
union.owin *(*intersect.owin*)*, 286
union.quad, *513*, 653
unique.ppp, *14*, *196*, *351*, *489*, 654, *656*
unique.ppx *(*unique.ppp*)*, 654
uniquemap, *655*
uniquemap.default, *656*
uniquemap.lpp *(*uniquemap.ppp*)*, 655
uniquemap.ppp, *14*, 655
uniquemap.ppx *(*uniquemap.ppp*)*, 655
unit.square *(*square*)*, 608
unitname, *93, 94*, *186, 187*, *189*, *191*, *193*, *242*, *336–338*, *343–346*, *366*, *368*, *370*, *372*, *398*, *401*, *403–405*, *538–543*, 656
unitname.box3, *19*
unitname.box3 *(*methods.box3*)*, 336
unitname.boxx *(*methods.boxx*)*, 337
unitname.pp3, *19*
unitname.pp3 *(*methods.pp3*)*, 342
unitname.ppx, *19*
unitname.ppx *(*methods.ppx*)*, 344
unitname<- *(*unitname*)*, 656
unitname<-.box3 *(*methods.box3*)*, 336
unitname<-.boxx *(*methods.boxx*)*, 337
unitname<-.pp3 *(*methods.pp3*)*, 342
unitname<-.ppx *(*methods.ppx*)*, 344
units, *543*
unmark, *14*, *221*, *224*, *320*, *323*, 658

unmark.psp, *18*
unmark.tess *(*marks.tess*)*, 322
unstack, *659–661*
unstack.layered *(*unstack.solist*)*, 660
unstack.msr, *660, 661*
unstack.ppp, 659, *661*
unstack.psp, *661*
unstack.psp *(*unstack.ppp*)*, 659
unstack.solist, 660
unstack.tess *(*unstack.ppp*)*, 659
update, *662*
update.symbolmap, *160*, *458*, *632*, 662

vdCorput *(*quasirandom*)*, 527
venn.tess, *18*, *96*, *485*, *518*, *634, 635*, 663
vertices, *295, 296*, *387*, 664
View, *253, 254*
volume, 665
volume.box3, *19*, *93*, *666*
volume.box3 *(*diameter.box3*)*, 167
volume.boxx, *19*, *94*, *666*
volume.boxx *(*diameter.boxx*)*, 168
volume.owin *(*area.owin*)*, 38
voronoiFlower *(*polarbody*)*, 483

where.max, 666
where.min *(*where.max*)*, 666
which.max, *666*
which.min, *666*
whichhalfplane, *272*, 667
Window, *15*, *195*, *240*, 668, *670, 671*
Window.distfun *(*Window.tess*)*, 670
Window.funxy *(*Window.tess*)*, 670
Window.layered *(*Window.tess*)*, 670
Window.nnfun *(*Window.tess*)*, 670
Window.ppm, *669*
Window.ppp, *671*
Window.psp, *671*
Window.quadratcount *(*Window.tess*)*, 670
Window.tess, *669*, 670
Window<- *(*Window*)*, 668
with, *672*
with.hyperframe, *20*, *259*, *432*, *611*, 671

X11, *450, 451*
X11.options, *451*
xfig, *450*
xy.coords, *284*, *292*, *414*, *667*

yardstick, *448*, *478*, *480, 481*, 672

zapsmall, *674*
zapsmall.im, *17*, 674