Package ‘sp’
October 7, 2020

Version 1.4-4
Title Classes and Methods for Spatial Data
Depends R (>= 3.0.0), methods
Imports utils, stats, graphics, grDevices, lattice, grid
Suggests RColorBrewer, rgdal (>= 1.2-3), rgeos (>= 0.3-13), gstat,
maptools, deldir
Description Classes and methods for spatial
data; the classes document where the spatial location information
resides, for 2D or 3D data. Utility functions are provided, e.g. for
plotting data as maps, spatial selection, as well as methods for
retrieving coordinates, for subsetting, print, summary, etc.
License GPL (>= 2)
BugReports https://github.com/edzer/sp/issues
Collate bpy.colors.R AAA.R Class-CRS.R CRS-methods.R Class-Spatial.R
Spatial-methods.R projected.R Class-SpatialPoints.R
SpatialPoints-methods.R Class-SpatialPointsDataFrame.R
SpatialPointsDataFrame-methods.R Class-SpatialMultiPoints.R
SpatialMultiPoints-methods.R
Class-SpatialMultiPointsDataFrame.R
SpatialMultiPointsDataFrame-methods.R Class-GridTopology.R
Class-SpatialGrid.R Class-SpatialGridDataFrame.R
Class-SpatialLines.R SpatialLines-methods.R
Class-SpatialLinesDataFrame.R SpatialLinesDataFrame-methods.R
Class-SpatialPolygons.R Class-SpatialPolygonsDataFrame.R
SpatialPolygons-methods.R SpatialPolygonsDataFrame-methods.R
SpatialGridDataFrame-methods.R SpatialPolygons-internals.R
point.in.polygon.R SpatialPolygons-displayMethods.R zerodist.R
sp_spat1.R merge.R aggregate.R
NeedsCompilation: yes

Author: Edzer Pebesma [aut, cre],
        Roger Bivand [aut],
        Barry Rowlingson [ctb],
        Virgilio Gomez-Rubio [ctb],
        Robert Hijmans [ctb],
        Michael Sumner [ctb],
        Don MacQueen [ctb],
        Jim Lemon [ctb],
        Finn Lindgren [ctb],
        Josh OBrien [ctb],
        Joseph O'Rourke [ctb]

Maintainer: Edzer Pebesma <edzer.pebesma@uni-muenster.de>

Repository: CRAN

Date/Publication: 2020-10-07 15:00:02 UTC

R topics documented:

- addAttrToGeom-methods ........................................... 4
- aggregate .................................................................. 5
- as.SpatialPolygons.GridTopology ................................. 7
- as.SpatialPolygons.PolygonsList ................................. 8
- bbox-methods ........................................................... 10
- bpy.colors .................................................................. 11
- bubble ....................................................................... 12
- char2dms ................................................................... 14
- compassRose ............................................................. 15
- coordinates ............................................................... 16
- coordinates-methods .................................................... 17
- coordnames-methods ................................................... 18
- CRS-class ................................................................. 18
- degAxis ..................................................................... 21
- dimensions-methods .................................................... 22
- disaggregate-methods ................................................... 23
- DMS-class ................................................................. 24
- flip ............................................................................. 25
- geometry-methods ....................................................... 26
- gridded-methods ......................................................... 27
- gridIndex2nb ............................................................. 28
- gridlines ................................................................... 29
- GridTopology-class ..................................................... 31
- image.SpatialGridDataFrame ......................................... 33
- is.projected ............................................................... 35
- Line .......................................................................... 37
- Line-class ................................................................. 38
- Lines-class ................................................................ 39
R topics documented:

- loadMeuse ................................................. 39
- mapasp ............................................. 40
- merge .................................................. 41
- meuse ................................................. 42
- meuse.grid ........................................... 43
- meuse.grid_ll ........................................ 44
- meuse.riv ............................................ 45
- over-methods ......................................... 46
- panel.spplot ......................................... 49
- point.in.polygon ...................................... 51
- Polygon-class ......................................... 52
- polygons .............................................. 53
- Polygons-class ........................................ 53
- polygons-methods ..................................... 54
- read.asciigrid ........................................ 55
- recenter-methods ...................................... 56
- Rlogo .................................................. 57
- select.spatial ......................................... 58
- sp ...................................................... 59
- sp-deprecated ......................................... 60
- Spatial-class .......................................... 61
- SpatialGrid-class ...................................... 63
- SpatialGridDataFrame-class ......................... 64
- SpatialLines ........................................... 67
- SpatialLines-class .................................... 68
- SpatialLinesDataFrame-class ......................... 70
- SpatialMultiPoints .................................... 71
- SpatialMultiPoints-class ............................. 72
- SpatialMultiPointsDataFrame-class ................. 74
- SpatialPixels .......................................... 75
- SpatialPixels-class ................................... 78
- SpatialPixelsDataFrame ............................... 79
- SpatialPixelsDataFrame-class ....................... 80
- SpatialPoints .......................................... 82
- SpatialPoints-class ................................... 83
- SpatialPointsDataFrame-class ....................... 85
- SpatialPolygons ....................................... 87
- SpatialPolygons-class ................................ 88
- SpatialPolygonsDataFrame class .................... 91
- spChFIDs-methods ..................................... 92
- spDistsN1 ............................................. 93
- spsample .............................................. 95
- spplot ................................................ 96
- spsample ............................................. 100
- spTransform .......................................... 103
- stack ............................................... 104
- surfaceArea .......................................... 105
- zerodist .............................................. 107

Index 109
addAttrToGeom-methods

constructs SpatialXxxDataFrame from geometry and attributes

Description

constructs SpatialXxxDataFrame from geometry and attributes

Usage

addAttrToGeom(x, y, match.ID, ...)

Arguments

x geometry (locations) of the queries
y data.frame object with attributes
match.ID logical; if TRUE, the IDs of the geometry and of the data.frame are matched (possibly swapping records), and an error occurs when some IDs do not match
... (optional) arguments passed to the constructor functions

Value

an object of class XxxDataFrame, where Xxx is the class of x

Methods

x = "SpatialPoints", y = "data.frame"
x = "SpatialPixels", y = "data.frame"
x = "SpatialGrid", y = "data.frame"
x = "SpatialLines", y = "data.frame"
x = "SpatialPolygons", y = "data.frame"

Author(s)

Edzer Pebesma, <edzer.pebesma@uni-muenster.de>

See Also

over
aggregate

aggregation of spatial objects

Description

spatial aggregation of thematic information in spatial objects

Usage

```r
## S3 method for class 'Spatial'
aggregate(x, by = list(ID = rep(1, length(x))),
          FUN, ..., dissolve = TRUE, areaWeighted = FALSE)
```

Arguments

- **x**: object deriving from `Spatial`, with attributes
- **by**: aggregation predicate; if by is a `Spatial` object, the geometry by which attributes in x are aggregated; if by is a list, aggregation by attribute(s), see `aggregate.data.frame`
- **FUN**: aggregation function, e.g. `mean`; see details
- **...**: arguments passed on to function FUN, unless `minDimension` is specified, which is passed on to function `over`
- **dissolve**: logical; should, when aggregating based on attributes, the resulting geometries be dissolved? Note that if x has class `SpatialPointsDataFrame`, this returns an object of class `SpatialMultiPointsDataFrame`
- **areaWeighted**: logical; should the aggregation of x be weighted by the areas it intersects with each feature of by? See value.

Details

`FUN` should be a function that takes as first argument a vector, and that returns a single number. The canonical examples are `mean` and `sum`. Counting features is obtained when summing an attribute variable that has the value 1 everywhere.

Value

The aggregation of attribute values of x either over the geometry of by using `over` for spatial matching, or by attribute values, using aggregation function `FUN`.

If `areaWeighted` is TRUE, `FUN` is ignored and the area weighted mean is computed for numerical variables, or if all attributes are factors, the area dominant factor level (area mode) is returned. This will compute the `gIntersection` of x and by; see examples below.

If by is missing, aggregates over all features.

Note

uses `over` to find spatial match if by is a `Spatial` object
Author(s)

Edzer Pebesma, <edzer.pebesma@uni-muenster.de>

Examples

data("meuse")
coordinates(meuse) <- ~x+y
data("meuse.grid")
coordinates(meuse.grid) <- ~x+y
gridded(meuse.grid) <- TRUE
i = cut(meuse.grid$dist, c(0,.25,.5,.75,1), include.lowest = TRUE)
j = sample(1:2, 3103, replace=TRUE)
## Not run:
if (require(rgeos)) {
  # aggregation by spatial object:
  ab = gUnaryUnion(as(meuse.grid, "SpatialPolygons"), meuse.grid$part.a)
  x = aggregate(meuse["zinc"], ab, mean)
  spplot(x)
  # aggregation of multiple variables
  x = aggregate(meuse[c("zinc", "copper")], ab, mean)
  spplot(x)
  # aggregation by attribute, then dissolve to polygon:
  x = aggregate(meuse.grid["dist"], list(i=i), mean)
  spplot(x["i"], col.regions=bpy.colors())
  x = aggregate(meuse.grid["dist"], list(i=i,j=j), mean)
  spplot(x["dist"], col.regions=bpy.colors());
  spplot(x["i"], col.regions=bpy.colors(4))
  spplot(x["j"], col.regions=bpy.colors())
}
## End(Not run)

x = aggregate(meuse.grid["dist"], list(i=i,j=j), mean, dissolve = FALSE)
spplot(x["j"], col.regions=bpy.colors())
if (require(gstat) && require(rgeos)) {
  x = idw(log(zinc)~1, meuse, meuse.grid, debug.level=0)[1]
  spplot(x[1],col.regions=bpy.colors())
  i = cut(x$var1.pred, seq(4, 7.5, by=.5),
           include.lowest = TRUE)
  xa = aggregate(x["var1.pred"], list(i=i), mean)
  spplot(xa[1],col.regions=bpy.colors(8))
}
if (require(rgeos)) {
  # Area-weighted example, using two partly overlapping grids:
  gt1 = SpatialGrid(GridTopology(c(0,0), c(1,1), c(4,4)))
  gt2 = SpatialGrid(GridTopology(c(-1.25,-1.25), c(1,1), c(4,4)))
  # convert both to polygons; give p1 attributes to aggregate
  p1 = SpatialPolygonsDataFrame(as(gt1, "SpatialPolygons"),

as.SpatialPolygons.GridTopology

Make SpatialPolygons object from GridTopology object

Description

Converts grids of regular rectangles into a SpatialPolygons object, which can be transformed to a different projection or datum with spTransform in package rgdal. The function is not suitable for high-resolution grids. The ordering of the grid cells is as in coordinates() of the same object, and is reported by IDvaluesGridTopology.

Usage

as.SpatialPolygons.GridTopology(grd, proj4string = CRS(as.character(NA)))
IDvaluesGridTopology(obj)
as.SpatialPolygons.SpatialPixels(obj)
IDvaluesSpatialPixels(obj)
HexPoints2SpatialPolygons(hex, dx)

Arguments

grd GridTopology object
proj4string object of class CRS-class
obj SpatialPixels object
hex: SpatialPoints object with points that are generated by hexagonal sampling; see \texttt{spsample}

dx: spacing of two horizontally adjacent points; if missing, this will be computed from the points

**Value**

\texttt{as.SpatialPolygons.GridTopology} and \texttt{as.SpatialPolygons.SpatialPixels} return a \texttt{SpatialPolygons} object; \texttt{IDvaluesGridTopology} and \texttt{IDvaluesSpatialPixels} return a character vector with the object grid indices.

**See Also**

\texttt{GridTopology}, \texttt{SpatialPixels}, \texttt{SpatialPolygons spTransform} in package \texttt{rgdal}

**Examples**

```r
library(lattice)
grd <- GridTopology(cellcentre.offset=\texttt{c(-175,55)}, cellsize=\texttt{c(10,10)}, cells.dim=\texttt{c(4,4)})
SpP_grd <- as.SpatialPolygons.GridTopology(grd)
plot(SpP_grd)
text(coordinates(SpP_grd), sapply(slot(SpP_grd, "polygons"), function(i) slot(i, "ID")), cex=0.5)
trdata <- data.frame(A=rep(c(1,2,3,4), 4), B=rep(c(1,2,3,4), each=4),
  row.names=sapply(slot(SpP_grd, "polygons"), function(i) slot(i, "ID")))
SpPDF <- SpatialPolygonsDataFrame(SpP_grd, trdata)
spplot(SpPDF)
data(meuse.grid)
gridded(meuse.grid)=~x+y
xx = spsample(meuse.grid, type="hexagonal", cellsize=200)
xxpl = HexPoints2SpatialPolygons(xx)
image(meuse.grid["dist"])
plot(xxpl, add = TRUE)
points(xx, cex = .5)
## Not run:
spplot(aggregate(as(meuse.grid[,1:3], "SpatialPolygonsDataFrame"), xxpl,
  areaWeighted=TRUE), main = "aggregated meuse.grid")
## End(Not run)
```

---

**as.SpatialPolygons.PolygonsList**

*Making SpatialPolygons objects*

**Description**

This function is used in making SpatialPolygons objects from other formats.
Usage

as.SpatialPolygons.PolygonsList(Srl, proj4string=CRS(as.character(NA)))

Arguments

- Srl: A list of Polygons objects
- proj4string: Object of class "CRS"; holding a valid proj4 string

Value

The functions return a SpatialPolygons object

Author(s)

Roger Bivand

Examples
bbox-methods

Description

retrieves spatial bounding box from spatial data

Usage

bbox(obj)

Arguments

obj object deriving from class "Spatial", or one of classes: "Line", "Lines", "Polygon" or "Polygons", or ANY, which requires obj to be an array with at least two columns

Value
two-column matrix; the first column has the minimum, the second the maximum values; rows represent the spatial dimensions

Methods

obj = "Spatial" object deriving from class "Spatial"
obj = "ANY" an array with at least two columns
obj = "Line" object deriving from class "Line"
obj = "Lines" object deriving from class "Lines"
obj = "Polygon" object deriving from class "Polygon"
obj = "Polygons" object deriving from class "Polygons"
## Examples

```r
# just 9 points on a grid:
x <- c(1,1,1,2,2,2,3,3,3)
y <- c(1,2,3,1,2,3,1,2,3)
xy <- cbind(x,y)
S <- SpatialPoints(xy)
bbox(S)
```

```r
# data.frame
data(meuse.grid)
coordinates(meuse.grid) <- ~x+y
gridded(meuse.grid) <- TRUE
bbox(meuse.grid)
```

### Description

Create a vector of `n` “contiguous” colors.

### Usage

```r
bpy.colors(n = 100, cutoff.tails = 0.1, alpha = 1.0)
```

### Arguments

- `n`: number of colors (>= 1) to be in the palette
- `cutoff.tails`: tail fraction to be cut off on each side. If 0, this palette runs from black to white; by cutting off the tails, it runs from blue to yellow, which looks nicer.
- `alpha`: numeric; alpha transparency, 0 is fully transparent, 1 is opaque.

### Value

A character vector, `cv`, of color names. This can be used either to create a user-defined color palette for subsequent graphics by `palette(cv)`, a `col=` specification in graphics functions or in `par`.

### Note

This color map prints well on black-and-white printers.

### Author(s)

unknown; the palette was posted to gnuplot-info a few decades ago; R implementation Edzer Pebesma, `<edzer.pebesma@uni-muenster.de>`
See Also

rainbow, cm.colors

Examples

bpy.colors(10)
p <- expand.grid(x=1:30,y=1:30)
p$z <- p$x + p$y
coordinates(p) <- c("x", "y")
grided(p) <- TRUE
image(p, col = bpy.colors(100), asp = 1)
# require(lattice)
# trellis.par.set("regions", list(col=bpy.colors())) # make this default pallette

bubbleCreate a bubble plot of spatial data

Description

Create a bubble plot of spatial data, with options for bicolour residual plots (xyplot wrapper)

Usage

bubble(obj, zcol = 1, ..., fill = TRUE, maxsize = 3, do.sqrt = TRUE, pch,
col = c("d01c8b", "4dac26"), key.entries = quantile(data[,zcol]), main,
identify = FALSE, labels = row.names(data.frame(obj)), key.space = "right",
scales = list(draw = FALSE), xlab = NULL, ylab = NULL, panel = panel.bubble,
sp.layout = NULL,
xlim = bbexpand(bbox(obj)[1,], 0.04),
ylim = bbexpand(bbox(obj)[2,], 0.04))

Arguments

obj object of, or extending, class SpatialPointsDataFrame or SpatialGridDataFrame, see coordinates or SpatialPointsDataFrame; the object knows about its spatial coordinates
zcol z-variable column name, or column number after removing spatial coordinates from x@data: 1 refers to the first non-coordinate column
fill logical; if TRUE, filled circles are plotted (pch = 16), else open circles (pch = 1); the pch argument overrides this
maxsize cex value for largest circle
do.sqrt logical; if TRUE the plotting symbol area (sqrt(diameter)) is proportional to the value of the z-variable; if FALSE, the symbol size (diameter) is proportional to the z-variable
pch plotting character
col  colours to be used; numeric vector of size two: first value is for negative values, 
    second for positive values. Default colors: 5-class PiYG from colorbrewer.org.

key.entries the values that will be plotted in the key; by default the five quantiles min, q.25, 
    median q.75, max

main  main plotting title

identify logical; if true, regular plot is called instead of xyplot, and followed by a call 
    to identify().

labels labels argument passed to plot if identify is TRUE 

... arguments, passed to xyplot, or plot if identification is required.

key.space location of the key

scales scales argument as passed to xyplot

xlab  x-axis label

ylab  y-axis label

panel panel function used

sp.layout possible layout items; see spplot

xlim  x axis limit

ylim  y axis limit

Value

returns (or plots) the bubble plot; if identify is TRUE, returns the indexes (row numbers) of 
identified points.

Author(s)

Edzer Pebesma

See Also

xyplot, mapasp, identify

Examples

data(meuse)
coordinates(meuse) <- c("x", "y") # promote to SpatialPointsDataFrame
bubble(meuse, "cadmium", maxsize = 2.5, main = "cadmium concentrations (ppm)",
    key.entries = 2^(-1:4))
bubble(meuse, "zinc", main = "zinc concentrations (ppm)",
    key.entries = 100 * 2^(0:4))
**char2dms**  
*Convert character vector to DMS-class object*

**Description**

These two helper functions convert character vectors and decimal degree vectors to the DMS-class representation of degrees, minutes, and decimal seconds. "DMS" objects cannot contain NAs.

**Usage**

```
char2dms(from, chd = "d", chm = "'", chs = "\"")
dd2dms(dd, NS = FALSE)
```

**Arguments**

- `from`: character vector of degree, minute, decimal second data
- `chd`: degree character terminator
- `chm`: minute character terminator
- `chs`: second character terminator
- `dd`: numeric vector of decimal degrees
- `NS`: logical, TRUE for north/south decimal degrees, FALSE for east/west decimal degrees

**Details**

In char2dms, the input data vector should use a regular format, such as that used in the PROJ.4 library, with a trailing capital (NSWE) indicating compass direction.

**Value**

Both functions return a "DMS" object.

**Methods**

- `from = "DMS", to = "numeric"` coerce a "DMS" object to a "numeric" vector
- `from = "DMS", to = "character"` coerce a "DMS" object to a "character" vector (the as.character.DMS S3 method is also available)

**Author(s)**

Roger Bivand <Roger.Bivand@nhh.no>

**See Also**

[DMS-class](#)
compassRose

Examples

data(state)
str(state.center$y)
stateN <- dd2dms(state.center$y, NS=TRUE)
str(attributes(stateN))
ch.stateN <- as.character(stateN)
str(ch.stateN)
stateNa <- char2dms(ch.stateN)
str(attributes(stateNa))
ch.stateN <- as(stateN, "character")
str(ch.stateN)
stateNa <- char2dms(ch.stateN)
str(attributes(stateNa))

compassRose(x,y,rot=0,cex=1)

Arguments

x, y The position of the center of the compass rose in user units.
rot Rotation for the compass rose in degrees. See Details.
cex The character expansion to use in the display.

Details

‘compassRose’ displays a conventional compass rose at the position requested. The size of the compass rose is determined by the character expansion, as the central "rose" is calculated relative to the character size. Rotation is in degrees counterclockwise.

Value

nil

Author(s)

Jim Lemon
coordinates

set spatial coordinates to create a Spatial object, or retrieve spatial coordinates from a Spatial object

Description

set spatial coordinates to create a Spatial object, or retrieve spatial coordinates from a Spatial object

Usage

coordinates(obj, ...)
coordinates(object) <- value

Arguments

obj object deriving from class "Spatial"
object object of class "data.frame"
value spatial coordinates; either a matrix, list, or data frame with numeric data, or column names, column number or a reference: a formula (in the form of e.g. ~x+y), column numbers (e.g. c(1, 2)) or column names (e.g. c("x", "y")) specifying which columns in object are the spatial coordinates. If the coordinates are part of object, giving the reference does not duplicate them, giving their value does duplicate them in the resulting structure.

... additional arguments that may be used by particular methods

Value

usually an object of class SpatialPointsDataFrame; if the coordinates set cover the full set of variables in object, an object of class SpatialPoints is returned

Examples

# data.frame
data(meuse.grid)
coordinates(meuse.grid) <- ~x+y
gridded(meuse.grid) <- TRUE
class(meuse.grid)
bbox(meuse.grid)

data(meuse)
meuse.xy = meuse[c("x", "y")]
coordinates(meuse.xy) <- ~x+y
class(meuse.xy)
coordinates-methods

Description

retrieve (or set) spatial coordinates from (for) spatial data

Methods

obj = "list" list with (at least) two numeric components of equal length
obj = "data.frame" data.frame with at least two numeric components
obj = "matrix" numeric matrix with at least two columns
obj = "SpatialPoints" object of, or deriving from, SpatialPoints
obj = "SpatialPointsDataFrame" object of, or deriving from, SpatialPointsDataFrame
obj = "SpatialPolygons" object of, or deriving from, SpatialPolygons
obj = "SpatialPolygonsDataFrame" object of, or deriving from, SpatialPolygonsDataFrame
obj = "Line" object of class Line; returned value is matrix
obj = "Lines" object of class Lines; returned value is list of matrices
obj = "SpatialLines" object of, or deriving from, SpatialLines; returned value is list of lists of matrices
obj = "GridTopology" object of, or deriving from, GridTopology
obj = "SpatialPixels" object of, or deriving from, SpatialPixels
obj = "SpatialPixelsDataFrame" object of, or deriving from, SpatialPixelsDataFrame
obj = "SpatialGrid" object of, or deriving from, SpatialGrid
obj = "SpatialGridDataFrame" object of, or deriving from, SpatialGridDataFrame

Methods for "coordinates<-"

object = "data.frame", value="ANY" promote data.frame to object of class SpatialPointsDataFrame-class, by specifying coordinates; see coordinates
Description

retrieve or assign coordinate names for classes in `sp`

Methods for `coordnames`

- `x = "SpatialPoints"` retrieves coordinate names
- `x = "SpatialLines"` retrieves coordinate names
- `x = "Lines"` retrieves coordinate names
- `x = "Line"` retrieves coordinate names
- `x = "SpatialPolygons"` retrieves coordinate names
- `x = "Polygons"` retrieves coordinate names
- `x = "Polygon"` retrieves coordinate names

Methods for "coordnames<-"

- `x = "SpatialPoints", value = "character"` replace coordinate names
- `x = "SpatialLines", value = "character"` replace coordinate names
- `x = "Lines", value = "character"` replace coordinate names
- `x = "Line", value = "character"` replace coordinate names
- `x = "SpatialPolygons", value = "character"` replace coordinate names
- `x = "GridTopology", value = "character"` replace coordinate names
- `x = "SpatialGrid", value = "character"` replace coordinate names
- `x = "SpatialPixels", value = "character"` replace coordinate names

CRS-class

Class "CRS" of coordinate reference system arguments

Description

Interface class to the PROJ projection and transformation system. The class is defined as an empty stub accepting value NA in the sp package. The initiation function may call the PROJ library through `rgdal` to verify the argument set against those known in the library, returning error messages where necessary. If the "CRS" object is instantiated using `CRS()` with `rgdal` using PROJ >= 6 and GDAL >= 3, the object may also have a WKTI2 (2019) string carried as a comment. The arguments for a Proj.4 string must be entered exactly as in the Proj.4 documentation, in particular there cannot be any white space in `<key>=<value>` strings, and successive such strings can only be separated by blanks. Note that only `"+proj=longlat +ellps=WGS84"` is accepted for geographical coordinates, which must be ordered (eastings, northings); the `"+ellps="` definition must be given (or expanded internally from a given `"+datum="` value) for recent versions of the Proj.4 library, and should be set to an appropriate value.
**Usage**

CRS(projargs, doCheckCRSArgs=TRUE, SRS_string=NULL, get_source_if_boundcrs=TRUE)

identicalCRS(x,y)

**Arguments**

projargs A character string of projection arguments; the arguments must be entered exactly as in the PROJ.4 documentation; if the projection is unknown, use as.character(NA), it may be missing or an empty string of zero length and will then set to the missing value. With rgdal built with PROJ >= 6 and GDAL >= 3, the +init= key may only be used with value epsg:<code>. From sp version 1.4-4, the string associated with the SRS_string argument may be entered as-is and will be set as SRS_string if the projargs argument does not begin with a + (suggested by Mikko Vihtakari).

doCheckCRSArgs default TRUE, must be set to FALSE by package developers including CRS in an S4 class definition to avoid uncontrollable loading of the rgdal namespace

SRS_string default NULL, only used when rgdal is built with PROJ >= 6 and GDAL >= 3; a valid WKT string or SRS definition such as "EPSG:4326" or "ESRI:102761"

get_source_if_boundcrs (from rgdal 1.5-17, default TRUE) The presence of the +towgs84= key in a Proj4 string projargs= argument value may promote the output WKT2 CRS to BOUNDCRS for PROJ >= 6 and GDAL >= 3, which is a coordinate operation from the input datum to WGS84. This is often unfortunate, so a PROJ function is called through rgdal to retrieve the underlying source definition.

x object having a proj4string method, or if y is missing, list with objects that have a proj4string method

y object of class Spatial, or having a proj4string method

**Value**

CRS returns on success an object of class CRS, identicalCRS returns a logical, indicating whether x and y have identical CRS, or if y is missing whether all objects in list x have identical CRS.

**Objects from the Class**

Objects can be created by calls of the form CRS("projargs"), where "projargs" is a valid string of PROJ.4 arguments. If the argument is a zero-length string or a character NA, the object records NA. If the "CRS" object is instantiated using CRS() with rgdal using PROJ >= 6 and GDAL >= 3, the object may also have a WKT2 (2019) string carried as a comment. The initiation function may call the PROJ library through rgdal to verify the argument set against those known in the library, returning error messages where necessary. The function CRSargs() can be used to show the expanded Proj.4 string used by the PROJ library.

**Slots**

projargs: Object of class "character": projection arguments; the arguments must be entered exactly as in the PROJ.4 documentation, in particular there cannot be any white space in +<arg>=<value> strings, and successive such strings can only be separated by blanks.
CRS-class

Methods

- **show** signature(object = "CRS"): print projection arguments in object
- **wkt** signature(object = "CRS"): return WKT comment on object
- **rebuild_CRS** rebuild a CRS object, usually used to add a WKT comment with PROJ >= 6 and GDAL >= 3

Note

Lists of projections may be seen by using the programs installed with the PROJ.4 library, in particular proj and cs2cs; with the latter, -lp lists projections, -le ellipsoids, -lu units, and -ld datum(s) known to the installed software (available in **rgdal** using projInfo). These are added to in successive releases, so tracking the website or compiling and installing the most recent revisions will give the greatest choice. Finding the very important datum transformation parameters to be given with the +towgs84 tag is a further challenge, and is essential when the datums used in data to be used together differ. Tracing projection arguments is easier now than before the mass ownership of GPS receivers raised the issue of matching coordinates from different argument sets (GPS output and paper map, for example). See **GridsDatums**, make_EPSG and showEPSG for help in finding CRS definitions.

The 4.9.1 release of the PROJ library omitted a small file of defaults, leading to reports of “major axis or radius = 0 or not given” errors. From 0.9-3, rgdal checks for the presence of this file (proj_def.dat), and if not found, and under similar conditions to those used by PROJ.4, adds “+ellps=WGS84” to the input string being checked by checkCRSArgs The “+no_defs” tag ignores the file of defaults, and the default work-around implemented to get around this problem; strings including “init” and “datum” tags also trigger the avoidance of the work-around. Now messages are issued when a candidate CRS is checked; they may be suppressed using suppressMessages.

From release 6 of the PROJ library, when used in building **rgdal** with GDAL >= 3, the +datum= key in the Proj.4 string CRS representation is deprecated, and the +towgs84= and +nadgrids= keys may be deprecated soon. For this reason, **sp**, **rgdal** and **sf** are starting to use WKT2 (2019) string representations. In **sp**, the "CRS" object in itself remains unchanged, but the content of its "projargs" slot may be degraded. To work around the degradation, a comment is added around the "CRS" object containing a WKT2 (2019) string when **rgdal** is available and built with PROJ >= 6 and GDAL >=3.

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

References

https://github.com/OSGeo/PROJ

Examples

```r
CRS()
CRS(""")
CRS(as.character(NA))
CRS("+proj=longlat +datum=WGS84")
run <- FALSE
```
run <- require(rgdal)
if (run) {
  print(CRSargs(CRS("+proj=longlat +datum=NAD27")))
}
if (run) {
  print(CRSargs(CRS("+init=epsg:4267")))
}
if (run) {
  print(CRSargs(CRS("+init=epsg:26978")))
}
if (run) {
  print(CRSargs(CRS("+init=epsg:28992")))
}
if (run) {
  o <- try(CRS(SRS_string="ESRI:102760"))
  if (!inherits(o, "try-error")) print(CRSargs(o))
}
if (run) {
  o <- try(CRS("EPSG:4326"))
  if (!inherits(o, "try-error")) print(CRSargs(o))
}
if (run) {
  o <- try(CRS("ESRI:102760"))
  if (!inherits(o, "try-error")) print(CRSargs(o))
}
if (run) {
  o <- new("Spatial")
  proj4string(o) <- CRS("+init=epsg:27700")
}
if (run && !is.null(comment(slot(o, "proj4string")))) {
  cat(wkt(o), sep="\n")
  cat(wkt(slot(o, "proj4string")), sep="\n")
}

---

degAxis  axis with degrees

Description
draw axes on a plot using degree symbols in numbers

Usage
degAxis(side, at, labels, ...)

Arguments

side integer; see axis
at numeric; if missing, axTicks is called for nice values; see axis
labels character; if omitted labels are constructed with degree symbols, ending in N/S/E/W; in case of negative degrees, sign is reversed and S or W is added; see axis
...

Value

axis is plotted on current graph

Note

decimal degrees are used if variation is small, instead of minutes and seconds

Examples

xy = cbind(x = 2 * runif(100) - 1, y = 2 * runif(100) - 1)
plot(SpatialPoints(xy, proj4string = CRS("+proj=longlat +ellps=WGS84")),xlim=c(-1,1),ylim=c(-1,1))
degAxis(1)
degAxis(2, at = c(-1,-0.5,0,0.5,1))
#

Description

retrieves spatial dimensions box from spatial data

Usage

dimensions(obj)

Arguments

obj object deriving from class "Spatial"

Value

two-column matrix; the first column has the minimum, the second the maximum values; rows represent the spatial dimensions

Methods

obj = "Spatial" object deriving from class "Spatial"
Examples

```r
# just 9 points on a grid:
x <- c(1,1,1,2,2,2,3,3,3)
y <- c(1,2,3,1,2,3,1,2,3)
xy <- cbind(x,y)
S <- SpatialPoints(xy)
dimensions(S)
```

```r
# data.frame
data(meuse.grid)
coordinates(meuse.grid) <- ~x+y
gridded(meuse.grid) <- TRUE
dimensions(meuse.grid)
```

disaggregate-methods  

*disaggregate* SpatialLines, SpatialLinesDataFrame, SpatialPolygons, or SpatialPolygonsDataFrame objects

Description

disaggregate SpatialLines, SpatialLinesDataFrame, SpatialPolygons, or SpatialPolygonsDataFrame objects, using functions from rgeos to handle polygon hole nesting

Usage

```r
disaggregate(x, ...)
```

Arguments

- **x**: object of class *SpatialLines* or *SpatialPolygons*
- **...**: ignored

Value

object of class *SpatialLines* or *SpatialPolygons*, where groups of *Line* or *Polygon* are disaggregated to one *Line* per *Lines*, or one *Polygon* per *Polygons*, respectively.

Author(s)

Robert Hijmans, Edzer Pebesma
Examples

Sr1 = Polygon(cbind(c(2,4,4,1,2),c(2,3,5,4,2)), hole = FALSE)
Sr2 = Polygon(cbind(c(5,4,2,5),c(2,3,2,2)), hole = FALSE)
Sr3 = Polygon(cbind(c(4,4,5,10,4),c(5,3,2,5,5)), hole = FALSE)
Sr4 = Polygon(cbind(c(5,6,6,5,5),c(4,4,3,3,4)), hole = TRUE)

Srs1 = Polygons(list(Sr1, Sr2), "s1/2")
Srs3 = Polygons(list(Sr3, Sr4), "s3/4")
sp = SpatialPolygons(list(Srs1,Srs3), 1:2)
length(sp) ## [1] 2
length(disaggregate(sp)) ## [1] 3

l1 = cbind(c(1,2,3),c(3,2,2))
l1a = cbind(l1[,1]+.05,l1[,2]+.05)
l2 = cbind(c(1,2,3),c(1,1.5,1))
Sl1 = Line(l1)
Sl1a = Line(l1a)
Sl2 = Line(l2)
S1 = Lines(list(Sl1, Sl1a), ID="a")
S2 = Lines(list(Sl2), ID="b")
sl = SpatialLines(list(S1,S2))
length(sl)
length(disaggregate(sl))

DMS-class

Class "DMS" for degree, minute, decimal second values

Description

The class provides a container for coordinates stored as degree, minute, decimal second values.

Objects from the Class

Objects can be created by calls of the form new("DMS", ...), converted from decimal degrees using dd2dms(), or converted from character strings using char2dms().

Slots

WS: Object of class "logical" TRUE if input value negative
deg: Object of class "numeric" degrees
min: Object of class "numeric" minutes
sec: Object of class "numeric" decimal seconds
NS: Object of class "logical" TRUE if input value is a Northing

Methods

coerce signature(from = "DMS", to = "numeric"): convert to decimal degrees
show signature(object = "DMS"): print data values
**flip**

*rearrange data in SpatialPointsDataFrame or SpatialGridDataFrame for plotting with spplot (levelplot/xyplot wrapper)*

### Description

rearrange SpatialPointsDataFrame for plotting with spplot or levelplot

### Usage

```r
flipHorizontal(x)
flipVertical(x)
```

### Arguments

- `x` object of class SpatialGridDataFrame

### Value

object of class SpatialGridDataFrame, with pixels flipped horizontally or vertically. Note that the spatial structure is destroyed (or at least: drastically changed).

### Author(s)

Michael Sumner
Examples

data(meuse.grid) # data frame
gridded(meuse.grid) = c("x", "y") # promotes to
fullgrid(meuse.grid) = TRUE
d = meuse.grid["dist"]
image(d, axes=TRUE)
image(flipHorizontal(d), axes=TRUE)
image(flipVertical(d), axes=TRUE)

Methods

Methods for retrieving the geometry from a composite (geometry + attributes) object

Description

gerrhometry retrieves the SpatialXxx object from a SpatialXxxDataFrame object, with Xxx Lines, Points, Polygons, Grid, or Pixels. geometry<- converts a data.frame into a Spatial object.

Usage

gerrhometry(obj)
gerrhometry(obj) <- value

Arguments

obj in case of assignment, a data.frame, else an object of class Spatial
value object of class Spatial

Methods

obj = "Spatial"
obj = "SpatialPointsDataFrame"
obj = "SpatialMultiPointsDataFrame"
obj = "SpatialPolygonsDataFrame"
obj = "SpatialPixelsDataFrame"
obj = "SpatialGridDataFrame"
obj = "SpatialLinesDataFrame"

Author(s)

Edzer Pebesma, <edzer.pebesma@uni-muenster.de>
Examples

```r
data(meuse)
m = meuse
coordinates(m) = meuse[, c("x", "y")]
pts = geometry(m)
class(pts)
geometry(meuse) = pts
class(meuse)
identical(m, meuse) # TRUE
```

Description

returns logical (TRUE or FALSE) telling whether the object is gridded or not; in assignment promotes a non-gridded structure to a gridded one, or demotes a gridded structure back to a non-structured one.

Usage

```r
gridded(obj)
gridded(obj) <- value
fullgrid(obj)
fullgrid(obj) <- value
gridparameters(obj)
```

Arguments

- **obj**: object deriving from class "Spatial" (for gridded), or object of class `SpatialGridDataFrame-class` (for fullgrid and gridparameters)
- **value**: logical replacement values, TRUE or FALSE

Value

if obj derives from class Spatial, gridded(object) will tell whether it is has topology on a regular grid; if assigned TRUE, if the object derives from SpatialPoints and has gridded topology, grid topology will be added to object, and the class of the object will be promoted to `SpatialGrid-class` or `SpatialGridDataFrame-class`

fullgrid returns a logical, telling whether the grid is full and ordered (i.e., in full matrix form), or whether it is not full or unordered (i.e. a list of points that happen to lie on a grid. If assigned, the way the points are stored may be changed. Changing a set of points to full matrix form and back may change the original order of the points, and will remove duplicate points if they were present.

gridparameters returns, if obj inherits from SpatialGridDataFrame its grid parameters, else it returns numeric(0). The returned value is a data.frame with three columns, named cellcentre.offset ("lower left cell centre coordinates"), cellsize, and cells.dim (cell dimension); the rows correspond to the spatial dimensions.
gridIndex2nb

create neighbourhood (nb) object from grid geometry

Description

create neighbourhood (nb) object from grid geometry

Usage

gridIndex2nb(obj, maxdist = sqrt(2), fullMat = TRUE, ...)

Arguments

obj object of class SpatialGrid or SpatialPixels
maxdist maximum distance to be considered (inclusive), expressed in number of grid cell
(sqr(2) results in queen neighbours)
fullMat use dist to compute distances from grid (row/col) indices; FALSE avoids forming
the full distance matrix, at a large performance cost
... arguments passed on to dist

Methods

obj = "Spatial" object deriving from class "Spatial"

Examples

# just 9 points on a grid:
x <- c(1,1,1,2,2,2,3,3,3)
y <- c(1,2,3,1,2,3,1,2,3)
xy <- cbind(x,y)
S <- SpatialPoints(xy)
class(S)
plot(S)
gridded(S) <- TRUE
gridded(S)
class(S)
summary(S)
plot(S)
gridded(S) <- FALSE
gridded(S)
class(S)

# data.frame
data(meuse.grid)
coordinates(meuse.grid) <- ~x+y
gridded(meuse.grid) <- TRUE
plot(meuse.grid) # not much good
summary(meuse.grid)
Value

Object of class nb, which is a list. The nb object follows the convention of nb objects in package spdep; it is a list with each list element corresponding to a grid cell or pixel; the list element contains the indices of neighbours defined as cells less than maxdist away, measured in cell unit (N/S/E/W neighbour has distance 1).

Note

Unequal grid cell size is ignored; grid cell row/col indices are taken to be the coordinates from which distances are computed.

Author(s)

Edzer Pebesma, <edzer.pebesma@uni-muenster.de>

See Also

plot.nb in package spdep

gridlines

Create N-S and E-W grid lines over a geographic region

Description

Create N-S and E-W grid lines over a geographic region; create and plot corresponding labels

Usage

gridlines(x, easts = pretty(bbox(x)[1,]), norths = pretty(bbox(x)[2,]), ndiscr = 100)
gridat(x, easts = pretty(bbox(x)[1,]), norths = pretty(bbox(x)[2,]), offset = 0.5, side = "WS")
## S3 method for class 'SpatialLines'
labels(object, labelCRS, side = 1:2, ...)
## S3 method for class 'SpatialPointsDataFrame'
text(x, ...)

Arguments

x object deriving from class Spatial-class
easts numeric; east-west values for vertical lines
norths numeric; north-south values for horizontal lines
ndiscr integer; number of points used to discretize the line, could be set to 2, unless the grid is (re)projected
offset offset value to be returned, see text
object       SpatialLines-class object, as returned by gridlines
labelCRS    the CRS in which the grid lines were drawn and labels should be printed; if
            missing, the CRS from object is taken
side        for labels: integer, indicating side(s) at which gridlines labels will be drawn:
            1=below (S), 2=left (W), 3=above (N), and 4=right (E); for gridat: default
            “WS”, if “EN” labels placed on the top and right borders
...         for labels: ignored; for text: arguments passed on to text, see below for ex-
            ample use of adj

Value

gridlines returns an object of class SpatialLines-class, with lines as specified; the return object
inherits the projection information of x; gridat returns a SpatialPointsDataFrame with points at
the west and south ends of the grid lines created by gridlines, with degree labels.

The labels method for SpatialLines objects returns a SpatialPointsDataFrame-class object with
the parameters needed to print labels below and left of the gridlines. The locations for the labels are
those of proj4string(object) the labels also unless labelCRS is given, in which case they are in
that CRS. This object is prepared to be plotted with text:

The text method for SpatialPointsDataFrame puts text labels on its coordinates, and takes care
of attributes pos, labels, srt and offset; see text.

Author(s)

Edzer Pebesma, <edzer.pebesma@uni-muenster.de>, using example code of Roger Bivand.

See Also

spTransform; llgridlines in rgdal (recent versions) for plotting long-lat grid over projected data

Examples

data(meuse)
coordinates(meuse) = ~x+y
plot(meuse)
plot(gridlines(meuse), add = TRUE)
text(labels(gridlines(meuse)));

proj4string(meuse) <- CRS("+init=epsg:28992")
crs.longlat <- CRS("+init=epsg:4326")
if (require(rgdal, quietly=TRUE)) {
  meuse_ll <- spTransform(meuse, crs.longlat)
grd <- gridlines(meuse_ll)
grd_x <- spTransform(grd, CRS("+init=epsg:28992"))
}

# labels South and West:
plot(meuse)
plot(grd_x, add=TRUE, lty=2)
grdat_ll <- gridat(meuse_ll)
grdat_x <- spTransform(grdat_ll, CRS("+init=epsg:28992"))
text(grdat_x)

# labels North and East:
plot(meuse)
plot(grd_x, add=TRUE, lty=2)
grdat_ll <- gridat(meuse_ll, side="EN")
grdat_x <- spTransform(grdat_ll, CRS("+init=epsg:28992"))
text(grdat_x)

# now using labels:
plot(meuse)
plot(grd_x, add=TRUE, lty=2)
text(labels(grd_x, crs.longlat))

# demonstrate axis labels with angle, both sides:
sp = SpatialPoints(rbind(c(-101,9), c(-101,55), c(-19,9), c(-19,55)), crs.longlat)
laea = CRS("+proj=laea +lat_0=30 +lon_0=-40")
sp.l = spTransform(sp, laea)
plot(sp.l, expandBB = c(0, 0.05, 0, .05))
gl = spTransform(gridlines(sp), laea)
plot(gl, add = TRUE)
text(labels(gl, crs.longlat))
text(labels(gl, crs.longlat, side = 3:4), col = 'red')
title("curved text label demo")

# polar:
pts=SpatialPoints(rbind(c(-180,-70),c(0,-70),c(180,-89),c(180,-70)), crs.longlat)
polar = CRS("+init=epsg:3031")
ql = spTransform(gridlines(pts, easts = seq(-180,180,20), ndiscr = 100), polar)
plot(spTransform(pts, polar), expandBB = c(.05,0,.05,0))
lines(gl)
l = labels(gl, crs.longlat, side = 3)
l$pos = NULL # pos is too simple, use adj:
text(l, adj = c(0.5, -0.5))
l = labels(gl, crs.longlat, side = 4)
l$srt = 0 # otherwise they end up upside-down
text(l)
title("grid line labels on polar projection, epsg 3031")
}
## Not run:
if (require(maps)) demo(polar) # adds the map of the antarctic
## End(Not run)

GridTopology-class

Class "GridTopology"

Description
class for defining a rectangular grid of arbitrary dimension
**Objects from the Class**

Objects are created by using e.g.

```
GridTopology(c(0,0), c(1,1), c(5,5))
```

see `SpatialGrid`

**Slots**

- `cellcentre.offset`: numeric; vector with the smallest centroid coordinates for each dimension; coordinates refer to the cell centre
- `cellsize`: numeric; vector with the cell size in each dimension
- `cells.dim`: integer; vector with number of cells in each dimension

**Methods**

- `coordinates` signature(x = "SpatialGrid"): calculates coordinates for each point on the grid
- `summary` signature(object = "SpatialGrid"): summarize object
- `coerce` signature(from = "GridTopology", to = "data.frame"): convert to data.frame with columns `cellcentre.offset`, `cellsize` and `cells.dim`

**Author(s)**

Edzer Pebesma, <edzer.pebesma@uni-muenster.de>

**See Also**

`SpatialGridDataFrame-class, SpatialGrid-class`

**Examples**

```r
x = GridTopology(c(0,0), c(1,1), c(5,5))
class(x)
x
summary(x)
coordinates(x)
y = SpatialGrid(grid = x)
class(y)
y
```
image.SpatialGridDataFrame

Image or contour method for gridded spatial data; convert to and from image data structure

Description

Create image for gridded data in SpatialGridDataFrame or SpatialPixelsDataFrame objects.

Usage

```r
## S3 method for class 'SpatialGridDataFrame'
image(x, attr = 1, xcol = 1, ycol = 2,
col = heat.colors(12), red=NULL, green=NULL, blue=NULL,
axes = FALSE, xlim = NULL,
ylim = NULL, add = FALSE, ..., asp = NA, setParUsrBB=FALSE,
interpolate = FALSE, angle = 0,
useRasterImage = !(.Platform$GUI[[1]] == "Rgui" &&
getIdentification() == "R Console") && missing(breaks), breaks,
zlim = range(as.numeric(x[[attr]])[is.finite(x[[attr]])]))
## S3 method for class 'SpatialPixelsDataFrame'
image(x, ...)
## S3 method for class 'SpatialPixels'
image(x, ...)
## S3 method for class 'SpatialGridDataFrame'
contour(x, attr = 1, xcol = 1, ycol = 2,
col = 1, add = FALSE, xlim = NULL, ylim = NULL, axes = FALSE,
..., setParUsrBB = FALSE)
## S3 method for class 'SpatialPixelsDataFrame'
contour(x, ...)
as.image.SpatialGridDataFrame(x, xcol = 1, ycol = 2, attr = 1)
image2Grid(im, p4 = as.character(NA), digits=10)
```

Arguments

- `x` object of class `SpatialGridDataFrame`
- `attr` column of attribute variable; this may be the column name in the data.frame of data (as.data.frame(data)), or a column number
- `xcol` column number of x-coordinate, in the coordinate matrix
- `ycol` column number of y-coordinate, in the coordinate matrix
- `col` a vector of colors
- `red,green,blue` columns names or numbers given instead of the `attr` argument when the data represent an image encoded in three colour bands on the 0-255 integer scale; all three columns must be given in this case, and the attribute values will be constructed using function `rgb`
image.SpatialGridDataFrame

axes
logical; should coordinate axes be drawn?

xlim
x-axis limits

ylim
y-axis limits

zlim
data limits for plotting the (raster, attribute) values

add
logical; if FALSE, the image is added to the plot layout setup by plot(as(x,"Spatial"),axes=axes,xlim=xlim,ylim=ylim,asp=asp) which sets up axes and plotting region; if TRUE, the image is added to the existing plot.

... arguments passed to image, see examples

asp
aspect ratio to be used for plot

setParUsrBB
default FALSE, see Spatial-class for further details

useRasterImage
if TRUE, use rasterImage to render the image if available; for legacy rendering set FALSE; should be FALSE on Windows SDI installations

breaks
class breaks for coloured values

interpolate
default FALSE, a logical vector (or scalar) indicating whether to apply linear interpolation to the image when drawing, see rasterImage

angle
default 0, angle of rotation (in degrees, anti-clockwise from positive x-axis, about the bottom-left corner), see rasterImage

im
list with components named x, y, and z, as used for image

p4
CRS object, proj4 string

digits
default 10, number of significant digits to use for checking equal row/column spacing

Value

as.image.SpatialGridDataFrame returns the list with elements x and y, containing the coordinates of the cell centres of a matrix z, containing the attribute values in matrix form as needed by image.

Note

Providing xcol and ycol attributes seems obsolete, and it is for 2D data, but it may provide opportunities for plotting certain slices in 3D data. I haven’t given this much thought yet.

generatedContour seems to misinterpret the coordinate values, if we take the image.default manual page as the reference.

Author(s)

Edzer Pebesma

See Also

image.default, SpatialGridDataframe-class, levelplot in package lattice. Function image.plot in package fields can be used to make a legend for an image, see an example in https://stat.ethz.ch/pipermail/r-sig-geo/2007-June/002143.html
Examples

data(meuse.grid)
coordinates(meuse.grid) = c("x", "y") # promote to SpatialPointsDataFrame
gridded(meuse.grid) = TRUE # promote to SpatialGridDataFrame
data(meuse)
coordinates(meuse) = c("x", "y")
image(meuse.grid["dist"], main = "Distance to river Meuse")
points(coordinates(meuse), pch = "+")
image(meuse.grid["dist"], main = "Distance to river Meuse",
useRasterImage=TRUE)
points(coordinates(meuse), pch = "+")

# color scale:
layout(cbind(1,2), c(4,1),1)
image(meuse.grid["dist"])
imageScale(meuse.grid$dist, axis.pos=4, add.axis=FALSE)
axis(4,at=c(0,.2,.4,.8), las=2)

data(Rlogo)
d = dim(Rlogo)
cellsize = abs(c(gt[2],gt[6]))
cells.dim = c(d[1], d[2]) # c(d[2],d[1])
cellcentre.offset = c(x = gt[1] + 0.5 * cellsize[1], y = gt[4] - (d[2] - 0.5) * abs(cellsize[2]))
grid = GridTopology(cellcentre.offset, cellsize, cells.dim)
df = as.vector(Rlogo[,1])
for (band in 2:d[3]) df = cbind(df, as.vector(Rlogo[,band]))
df = as.data.frame(df)
names(df) = paste("band", 1:d[3], sep="")
Rlogo <- SpatialGridDataFrame(grid = grid, data = df)
summary(Rlogo)
image(Rlogo, red="band1", green="band2", blue="band3")

is.projected

Sets or retrieves projection attributes on classes extending SpatialData

Description

Sets or retrieves projection attributes on classes extending SpatialData; set or retrieve option value for error or warning on exceedance of geographical coordinate range, set or retrieve option value for exceedance tolerance of geographical coordinate range. Note that only ‘+proj=longlat +ellps=WGS84’ is accepted for geographical coordinates, which must be ordered (eastings, northings); the ‘+ellps=’
is.projected

Definition must be given (or expanded internally from a given “+datum=” value) for recent versions of the PROJ library, and should be set to an appropriate value.

From release 6 of the PROJ library, when used in building `rgdal` with GDAL >= 3, the +datum= key in the Proj.4 string CRS representation is deprecated, and the +towgs84= and +nadgrids= keys may be deprecated soon. For this reason, `sp`, `rgdal` and `sf` are starting to use WKT2 (2019) string representations. In `sp`, the "CRS" object in itself remains unchanged, but the content of its "projargs" slot may be degraded. To work around the degradation, a comment is added around the "CRS" object containing a WKT2 (2019) string when `rgdal` is available and built with PROJ >= 6 and GDAL >=3. The `wkt()` accessor function returns the WKT2 (2019) string comment belonging to the "CRS" object.

Usage

```r
is.projected(obj)
proj4string(obj)
proj4string(obj) <- value
wkt(obj)
get_ll_warn()
get_ll_TOL()
get_ReplCRS_warn()
set.ll_warn(value)
set.ll_TOL(value)
set.ReplCRS_warn(value)
```

Arguments

- `obj` An object of class or extending `Spatial-class`
- `value` For `proj4string` CRS object, containing a valid proj4 string; attempts to assign an object containing “longlat” to data extending beyond longitude [-180, 360] or latitude [-90, 90] will be stopped. For `set.ll_warn` a single logical value, if FALSE (default) error on range exceedance, if TRUE, warning. For `set.ll_TOL` the value of the power of `Machine$double.eps` (default 0.25) to use as tolerance in testing range exceedance. `set.ReplCRS_warn` may be used to turn off warnings issued when changing object CRS with the `proj4string` replacement method (by setting `value=FALSE`).

Details

proj4 strings are operative through CRAN package `rgdal`. For strings defined as “longlat”, the minimum longitude should be -180, the maximum longitude 360, the minimum latitude -90, and the maximum latitude 90. Note that the `proj4string` replacement method does not project spatial data - for this use `spTransform` methods in the `rgdal` package.

Value

- `is.projected` returns a logical that may be NA; `proj4string` returns a character vector of length 1.
**Line**

Create objects of class Line or Lines from coordinates.

### Usage

- `Line(coords)`
- `Lines(slinelist, ID)`

### Arguments

- `coords` 2-column numeric matrix with coordinates for a single line
- `slinelist` list with elements of class `Line-class`
- `ID` a single word unique character identifier, character vector of length one

### Value

- `Line` returns an object of class `Line-class`;
- `Lines` returns an object of class `Lines-class`

### See Also

- `SpatialLines-class`
Examples

```r
# from the sp vignette:
l1 = cbind(c(1,2,3),c(3,2,2))
l1a = cbind(l1[,1]+.05,l1[,2]+.05)
l2 = cbind(c(1,2,3),c(1,1.5,1))
Sl1 = Line(l1)
Sl1a = Line(l1a)
Sl2 = Line(l2)
S1 = Lines(list(Sl1, Sl1a), ID="a")
S2 = Lines(list(Sl2), ID="b")
```

Line-class

Class "Line"

Description

class for line objects

Objects from the Class

Objects can be created by calls of the form `new("Line", ...). or (preferred) by calls to the function

```r
Line
```

Slots

`coords`: Object of class "matrix", containing the line coordinates

Methods

`coordinates` signature(obj = "Line") retrieve coordinates from line

`lines` signature(x = "Line") add lines to a plot

Author(s)

Roger Bivand, Edzer Pebesma

See Also

`Lines-class, SpatialLines-class`
**Description**

class for sets of line objects

**Arguments**

SL, Lines an Lines object

**Objects from the Class**

Objects can be created by calls to the function Line

**Slots**

- **Lines**: Object of class "list", containing elements of class Line-class
- **ID**: "character" vector of length one, with unique identifier string

**Methods**

- `coordinates` signature(obj = "Line"): retrieve coordinates from lines; returns list with matrices
- `lines` signature(x = "Line"): add lines to a plot

**Author(s)**

Roger Bivand, Edzer Pebesma

**See Also**

Lines-class, SpatialLines-class

---

**loadMeuse**

deprecated function to load the Meuse data set

**Description**

deprecated function to load the Meuse data set

**Usage**

`loadMeuse()`

**Value**

none; it prints a warning to run demo(meuse)
mapasp

See Also
meuse, meuse.grid

Examples
demo(meuse)

Description
Calculate aspect ratio for plotting geographic maps; create nice degree axis labels

Usage
mapasp(data, xlim, ylim)
degreeLabelsEW(x)
degreeLabelsNS(x)

Arguments
data object of class or extending Spatial
xlim the xlim argument passed (or derived from bounding box)
ylim the ylim argument passed (or derived from bounding box)
x numeric; values at which tics and marks will be generated

Value
mapasp is used for the aspect argument in lattice plots and spatstat;
let x = dy/dx, with dy and dx the y- and x-size of the map.
let s = 1/cos((My * pi)/180) with My the y coordinate of the middle of the map (the mean of ylim)
for latlong (longlat) data, mapasp returns s * x. for other data, mapasp returns "iso".

Note
the values for x are typically obtained from axTicks

See Also
levelplot in package lattice
### Description

Merge a Spatial object having a data.frame (i.e. merging of non-spatial attributes).

### Usage

```r
## S4 method for signature 'Spatial, data.frame'
merge(x, y, by = intersect(names(x), names(y)),
      by.x = by, by.y = by, all.x = TRUE, suffixes = c(".x",".y"),
      incomparables = NULL, duplicateGeoms = FALSE, ...)
```

### Arguments

- `x`: object deriving from `Spatial`
- `y`: object of class `data.frame`, or any other class that can be coerced to a `data.frame` with `as.data.frame`
- `by, by.x, by.y`: specifications of the common columns. See 'Details' in (base) `merge`.
- `all.x`: logical; if TRUE, then the returned object will have all rows of `x`, even those that has no matching row in `y`. These rows will have NAs in those columns that are usually filled with values from `y`
- `suffixes`: character(2) specifying the suffixes to be used for making non-by names() unique.
- `incomparables`: values which cannot be matched. See `match`.
- `duplicateGeoms`: logical; if TRUE geometries in `x` are duplicated if there are multiple matches between records in `x` and `y`
- `...`: arguments to be passed to or from methods.

### Value

a Spatial* object

### Author(s)

Robert J. Hijmans

### See Also

`merge`
Meuse river data set

Description

This data set gives locations and topsoil heavy metal concentrations, along with a number of soil and landscape variables at the observation locations, collected in a flood plain of the river Meuse, near the village of Stein (NL). Heavy metal concentrations are from composite samples of an area of approximately 15 m x 15 m.

Usage

data(meuse)

Format

This data frame contains the following columns:

- **x** a numeric vector; Easting (m) in Rijksdriehoek (RDH) (Netherlands topographical) map coordinates
- **y** a numeric vector; Northing (m) in RDH coordinates
- **cadmium** topsoil cadmium concentration, mg kg\(^{-1}\) soil ("ppm"); zero cadmium values in the original data set have been shifted to 0.2 (half the lowest non-zero value)
- **copper** topsoil copper concentration, mg kg\(^{-1}\) soil ("ppm")
- **lead** topsoil lead concentration, mg kg\(^{-1}\) soil ("ppm")
- **zinc** topsoil zinc concentration, mg kg\(^{-1}\) soil ("ppm")
- **elev** relative elevation above local river bed, m
- **dist** distance to the Meuse; obtained from the nearest cell in `meuse.grid`, which in turn was derived by a spread (spatial distance) GIS operation, horizontal precision 20 metres; then normalized to \([0,1]\)$
- **om** organic matter, kg (100 kg\(^{-1}\)) soil (percent)
- **ffreq** flooding frequency class: 1 = once in two years; 2 = once in ten years; 3 = one in 50 years
- **soil** soil type according to the 1:50 000 soil map of the Netherlands. 1 = Rd10A (Calcareous weakly-developed meadow soils, light sandy clay); 2 = Rd90C/VII (Non-calcareous weakly-developed meadow soils, heavy sandy clay to light clay); 3 = Bkd26/VII (Red Brick soil, fine-sandy, silty light clay)
- **lime** lime class: 0 = absent, 1 = present by field test with 5% HCl
- **landuse** landuse class: Aa Agriculture/unspecified = , Ab = Agr/sugar beetsm, Ag = Agr/small grains, Ah = Agr/??, Am = Agr/maize, B = woods, Bw = trees in pasture, DEN = ??, Fh = tall fruit trees, Fl = low fruit trees; Fw = fruit trees in pasture, Ga = home gardens, SPO = sport field, STA = stable yard, Tv = ?? , W = pasture
- **dist.m** distance to river Meuse in metres, as obtained during the field survey
Note

row.names refer to the original sample number.

Soil units were mapped with a minimum delineation width of 150 m, and so somewhat generalize the landscape.

Approximate equivalent World Reference Base 2002 for Soil Resources names are: Rd10A Gleyic Fluvisols; Rd90C Haplic Fluvisols; Bkd26 Haplic Luvisols. Units Rd90C and Bkd26 have winter groundwater > 80cm, summer > 120cm depth.

Author(s)

Field data were collected by Ruud van Rijn and Mathieu Rikken; compiled for R by Edzer Pebesma; description extended by David Rossiter

References

M G J Rikken and R P G Van Rijn, 1993. Soil pollution with heavy metals - an inquiry into spatial variation, cost of mapping and the risk evaluation of copper, cadmium, lead and zinc in the floodplains of the Meuse west of Stein, the Netherlands. Doctoraalveldwerkverslag, Dept. of Physical Geography, Utrecht University


Stichting voor Bodemkartering (STIBOKA), 1970. Bodemkaart van Nederland : Blad 59 Peer, Blad 60 West en 60 Oost Sittard: schaal 1 : 50 000. Wageningen, STIBOKA.

http://www.gstat.org/

Examples

data(meuse)
summary(meuse)
coordinates(meuse) <- ~x+y
proj4string(meuse) <- CRS("+init=epsg:28992")

meuse.grid Prediction Grid for Meuse Data Set

Description

The meuse.grid data frame has 3103 rows and 7 columns; a grid with 40 m x 40 m spacing that covers the Meuse study area (see meuse)

Usage

data(meuse.grid)
Format
This data frame contains the following columns:

- **x** a numeric vector; x-coordinate (see `meuse`)
- **y** a numeric vector; y-coordinate (see `meuse`)
- **dist** distance to the Meuse river; obtained by a spread (spatial distance) GIS operation, from border of river; normalized to $[0,1]$
- **ffreq** flooding frequency class, for definitions see this item in `meuse`; it is not known how this map was generated
- **part.a** arbitrary division of the area in two areas, a and b
- **part.b** see part.a
- **soil** soil type, for definitions see this item in `meuse`; it is questionable whether these data come from a real soil map, they do not match the published 1:50 000 map

Details
$x$ and $y$ are in RD New, the Dutch topographical map coordinate system. Roger Bivand projected this to UTM in the R-Grass interface package.

Source
http://www.gstat.org/

References
See the `meuse` documentation

Examples
```r
data(meuse.grid)
coordinates(meuse.grid) = ~x+y
proj4string(meuse.grid) <- CRS("+init=epsg:28992")
gridded(meuse.grid) = TRUE
spplot(meuse.grid)
```

---

meuse.grid_ll  
Prediction Grid for Meuse Data Set, geographical coordinates

Description
The object contains the meuse.grid data as a SpatialPointsDataFrame after transformation to WGS84 and geographical coordinates.

Usage
```r
data(meuse.grid_ll)
```
Format

The format is: Formal class 'SpatialPointsDataFrame' [package "sp"].

Source

See the meuse documentation

Examples

data(meuse.grid.ll)

<table>
<thead>
<tr>
<th>meuse.riv</th>
<th>River Meuse outline</th>
</tr>
</thead>
</table>

Description

The meuse.riv data consists of an outline of the Meuse river in the area a few kilometers around the meuse data set.

The meuse.area polygon has an outline of meuse.grid. See example below how it can be created from meuse.grid.

Usage

data(meuse.riv)
data(meuse.area)

Format

meuse.riv: two-column data.frame containing 176 coordinates.
meuse.area: two-column matrix with coordinates of outline.

Details

x and y are in RDM, the Dutch topographical map coordinate system. See examples of spTransform in the rgdal package for projection parameters.

References

See the meuse documentation
Examples

```r
data(meuse.riv)
plot(meuse.riv, type = "l", asp = 1)
data(meuse.grid)
coordinates(meuse.grid) = c("x", "y")
gridded(meuse.grid) = TRUE
image(meuse.grid, "dist", add = TRUE)
data(meuse)
coordinates(meuse) = c("x", "y")
meuse.sr = SpatialPolygons(list(Polygons(list(Polygon(meuse.riv)),"meuse.riv")))
spplot(meuse.grid, col.regions=bpy.colors(), main = "meuse.grid",
       sp.layout=list(
list("sp.polygons", meuse.sr),
list("sp.points", meuse, pch="+", col="black")
))
spplot(meuse, "zinc", col.regions=bpy.colors(), main = "zinc, ppm",
cuts = c(100,200,400,700,1200,2000), key.space = "right",
       sp.layout= list("sp.polygons", meuse.sr, fill = "lightblue")
)

# creating meuse.area from meuse.grid:
if (require(rgdal)) {
  meuse.area = gUnaryUnion(as(meuse.grid, "SpatialPolygons"))
plot(meuse.area)
}
```

---

over-methods  

consistent spatial overlay for points, grids and polygons

Description

consistent spatial overlay for points, grids and polygons: at the spatial locations of object x retrieves the indexes or attributes from spatial object y

Usage

```r
over(x, y, returnList = FALSE, fn = NULL, ...)
x %over% y
```

Arguments

- `x`: geometry (locations) of the queries
- `y`: layer from which the geometries or attributes are queried
- `returnList`: logical; see value
- `fn`: (optional) a function; see value
... arguments passed on to function fn, except for the special argument minDimension: minimal dimension for an intersection to be counted; -1 takes any intersection, and does not order; 0 takes any intersection but will order according to dimensionality of the intersections (if returnType is TRUE, 1 (2) selects intersections with dimension 1, meaning lines (2, meaning areas); see vignette("over") for details)

Value

If y is only geometry an object of length length(x). If returnType is FALSE, a vector with the (first) index of y for each geometry (point, grid cell centre, polygon or lines) matching x. if returnType is TRUE, a list of length length(x), with list element i the vector of all indices of the geometries in y that correspond to the i-th geometry in x.

If y has attribute data, attribute data are returned. returnType is FALSE, a data.frame with number of rows equal to length(x) is returned, if it is TRUE a list with length(x) elements is returned, with a list element the data.frame elements of all geometries in y that correspond to that element of x.

In case the rgeos over methods are used, matching is done by gRelate, which uses DE-9IM (https://en.wikipedia.org/wiki/DE-9IM). From the string returned, characters 1, 2, 4 and 5 are used, indicating the dimension of the overlap of the inner and boundary of each x geometry with the inner and boundary of each y geometry. The order in which matched y geometries are returned is determined by the dimension of the overlap (2: area overlap, 1: line in common, 0: point in common), and then by the position in the string (1, 2, 4, 5, meaning points in polygons are preferred over points on polygon boundaries).

Methods

x = "SpatialPoints", y = "SpatialPolygons" returns a numeric vector of length equal to the number of points; the number is the index (number) of the polygon of y in which a point falls; NA denotes the point does not fall in a polygon; if a point falls in multiple polygons, the last polygon is recorded.

x = "SpatialPointsDataFrame", y = "SpatialPolygons" equal to the previous method, except that an argument fn=xxx is allowed, e.g. fn = mean which will then report a data.frame with the mean attribute values of the x points falling in each polygon (set) of y

x = "SpatialPoints", y = "SpatialPolygonsDataFrame" returns a data.frame of the second argument with row entries corresponding to the first argument

x = "SpatialPolygons", y = "SpatialPoints" returns the polygon index of points in y; if x is a SpatialPolygonsDataFrame, a data.frame with rows from x corresponding to points in y is returned.

x = "SpatialGridDataFrame", y = "SpatialPoints" returns object of class SpatialPointsDataFrame with grid attribute values x at spatial point locations y; NA for NA grid cells or points outside grid, and NA values on NA grid cells.

x = "SpatialGrid", y = "SpatialPoints" returns grid values x at spatial point locations y; NA for NA grid cells or points outside the grid

x = "SpatialPixelsDataFrame", y = "SpatialPoints" returns grid values x at spatial point locations y; NA for NA grid cells or points outside the grid
\texttt{x = "SpatialPixels", y = "SpatialPoints"} returns grid values \(x\) at spatial point locations \(y\); NA for NA grid cells or points outside the grid
\texttt{x = "SpatialPoints", y = "SpatialGrid"} xx
\texttt{x = "SpatialPoints", y = "SpatialGridDataFrame"} xx
\texttt{x = "SpatialPoints", y = "SpatialPixels"} xx
\texttt{x = "SpatialPoints", y = "SpatialPixelsDataFrame"} xx
\texttt{x = "SpatialPolygons", y = "SpatialGridDataFrame"} xx

\textbf{Note}

\texttt{over} can be seen as a left outer join in SQL; the match is a spatial intersection.
points on a polygon boundary and points corresponding to a polygon vertex are considered to be inside the polygon.
These methods assume that pixels and grid cells are never overlapping; for objects of class \texttt{SpatialPixels} this is not guaranteed.
\texttt{over} methods that involve \texttt{SpatialLines} objects, or pairs of \texttt{SpatialPolygons} require package \texttt{rgeos}, and use \texttt{gIntersects}.

\textbf{Author(s)}

Edzer Pebesma, \texttt{<edzer.pebesma@uni-muenster.de>}

\textbf{See Also}

vignette("over") for examples and figures; \texttt{point.in.polygon}, package \texttt{gIntersects}

\textbf{Examples}

\begin{verbatim}
r1 = cbind(c(180114, 180553, 181127, 181477, 181294, 181007, 180409, 180162, 180114), c(332349, 332057, 332342, 33250, 333558, 333676, 332618, 332413, 332349))
r2 = cbind(c(180042, 180545, 180553, 180314, 179955, 179142, 179437, 179524, 179979, 180042), c(332373, 332026, 331426, 331426, 330889, 330683, 331133, 331623, 332152, 332357, 332373))
r3 = cbind(c(179110, 179907, 180433, 180712, 180752, 180329, 179875, 179668, 179572, 179269, 178879, 178600, 178544, 179046, 179110), c(331086, 330620, 330494, 330265, 330075, 330233, 330336, 330004, 329783, 329665, 329720, 329933, 330478, 331062, 331086))
r4 = cbind(c(180304, 180403, 179632, 179420, 180304), c(332791, 333204, 333635, 333058, 332791))
sr1=Polygons(list(Polygon(r1)),"r1")
sr2=Polygons(list(Polygon(r2)),"r2")
sr3=Polygons(list(Polygon(r3)),"r3")
sr4=Polygons(list(Polygon(r4)),"r4")
sr=SpatialPolygons(list(sr1,sr2,sr3,sr4))
srdf= SpatialPolygonsDataFrame(sr, data.frame(cbind(1:4,5:2), row.names=c("r1","r2","r3","r4")))
\end{verbatim}
data(meuse)
coordinates(meuse) = ~x+y

plot(meuse)
polygon(r1)
polygon(r2)
polygon(r3)
polygon(r4)

# retrieve mean heavy metal concentrations per polygon:
over(sr, meuse[,1:4], fn = mean)

# return the number of points in each polygon:
sapply(over(sr, geometry(meuse), returnList = TRUE), length)

data(meuse.grid)
coordinates(meuse.grid) = ~x+y
gridded(meuse.grid) = TRUE

over(sr, geometry(meuse))
over(sr, meuse)
over(sr, geometry(meuse), returnList = TRUE)
over(sr, meuse, returnList = TRUE)

over(meuse, sr)
over(meuse, srdf)

# same thing, with grid:
over(sr, meuse.grid)
over(sr, meuse.grid, fn = mean)
over(sr, meuse.grid, returnList = TRUE)

over(meuse.grid, sr)
over(meuse.grid, srdf, fn = mean)
over(as(meuse.grid, "SpatialPoints"), sr)
over(as(meuse.grid, "SpatialPoints"), srdf)

---

**panel.spplot** panel and panel utility functions for spplot

**Description**

panel functions for spplot functions, and functions that can be useful within these panel functions

**Usage**

```
spplot.key(sp.layout, rows = 1, cols = 1)
SpatialPolygonsRescale(obj, offset, scale = 1, fill = "black", col = "black", plot.grid = TRUE, ...)
sp.lines(obj, col = 1, ...)
sp.points(obj, pch = 3, ...)
```
sp.polygons(obj, col = 1, fill = "transparent", ...)
sp.grid(obj, col = 1, alpha = 1,..., at = pretty(obj[[1]]), col.regions = col)
sp.text(loc, txt, ...)
sp.panel.layout(lst, p.number, ...)
bbexpand(x, fraction)

Arguments

sp.layout list; see spplot for definition
rows integer; panel row(s) for which the layout should be drawn
cols integer; panel column(s) for which the layout should be drawn
obj object of class SpatialPolygons-class for SpatialPolygonsRescale; of class SpatialLines-class, Lines-class or Line-class for sp.lines of a class that has a coordinates-methods for sp.points; of class SpatialPolygons-class for sp.polygons. When obj is character, the actual object is retrieved by get(obj) before its class is evaluated.
offset offset for shifting a Polygons object
scale scale for rescaling
fill fill color
col line color
plot.grid logical; plot through grid functions (TRUE), or through traditional graphics functions (FALSE)
pch plotting character
at numeric; values at which colour breaks should occur
col.regions colours to fill the grid cells, defaults to col
loc numeric vector of two elements
txt text to be plotted
alpha alpha (transparency) level
lst sp.layout argument, see spplot
p.number panel number; in a panel, panel.number() should be passed to this argument
x length two numeric vector, containing a range
fraction fraction to expand the range by
... arguments passed to the underlying panel, lattice or grid functions

Note

The panel functions of spplot, panel.gridplot for grids, panel.pointsplot for points, or panel.polygonsplot for lines or polygons can be called with arguments (x,y,...). Customizing spplot plots can be done by extending the panel function, or by supplying an sp.layout argument; see the documentation for spplot. Inside these panel functions, sp.panel.layout is called to deal with plotting the items in a sp.layout object.

SpatialPolygonsRescale scales and shifts an object of class SpatialPolygons-class; this is useful e.g. for scale bars, or other layout items.
point.in.polygon

sp.lines, sp.points, sp.polygons and sp.text plot lines, points, polygons or text in a panel.
spplot.key draws the sp.layout object at given rows/cols.
sp.pagefn can be passed as a page argument, and will call function spplot.key for the last panel
drawn on a page.

Author(s)
Edzer Pebesma, <edzer.pebesma@uni-muenster.de>

References
https://edzer.github.io/sp/ has a graph gallery with examples with R code.

See Also
spplot, spplot-methods

---

point.in.polygon do point(s) fall in a given polygon?

Description
verifies for one or more points whether they fall in a given polygon

Usage
point.in.polygon(point.x, point.y, pol.x, pol.y, mode.checked=FALSE)

Arguments
point.x numerical array of x-coordinates of points
point.y numerical array of y-coordinates of points
pol.x numerical array of x-coordinates of polygon
pol.y numerical array of y-coordinates of polygon
mode.checked default FALSE, used internally to save time when all the other argument are
known to be of storage mode double

Value
integer array; values are: 0: point is strictly exterior to pol; 1: point is strictly interior to pol; 2:
point lies on the relative interior of an edge of pol; 3: point is a vertex of pol.

References
Uses the C function InPoly(). InPoly is Copyright (c) 1998 by Joseph O’Rourke. It may be freely
redistributed in its entirety provided that this copyright notice is not removed.
Examples

# open polygon:
point.in.polygon(1:10,1:10,c(3,5,5,3),c(3,3,5,5))
# closed polygon:
point.in.polygon(1:10,rep(4,10),c(3,5,5,3,3),c(3,3,5,5,3))

Polygon-class

Class "Polygon"

Description

class for spatial polygon

Objects from the Class

Objects can be created by calls to the function Polygon

Slots

ringDir: Object of class "integer"; the ring direction of the ring (polygon) coordinates, holes are expected to be anti-clockwise

labpt: Object of class "numeric"; an x, y coordinate pair forming the label point of the polygon

area: Object of class "numeric"; the planar area of the polygon, does not respect projection as objects of this class have no projection defined

hole: Object of class "logical"; does the polygon seem to be a hole

coords: Object of class "matrix"; coordinates of the polygon; first point should equal the last point

Extends

Class "Line", directly.

Methods

No methods defined with class "Polygon" in the signature.

Author(s)

Roger Bivand

See Also

Polygons-class, SpatialPolygons-class
polygons

<table>
<thead>
<tr>
<th>Description</th>
<th>sets spatial coordinates to create spatial data, or retrieves spatial coordinates</th>
</tr>
</thead>
</table>

**Usage**

```r
desc <- GridTopology(c(1,1), c(1,1), c(10,10))
polys <- as.SpatialPolygons.GridTopology(desc)
centroids <- coordinates(polys)
x <- centroids[,1]
y <- centroids[,2]
z <- 1.4 + 0.1*x + 0.2*y + 0.002*x^2
df <- data.frame(x=x, y=y, z=z, row.names=row.names(polys))
polygons(df) <- polys
class(df)
summary(df)
```

**Value**

polygons returns the SpatialPolygons of obj; polygons<- promotes a data.frame to a SpatialPolygonsDataFrame object.

**Examples**

```r
g <- GridTopology(c(1,1), c(1,1), c(10,10))
polys <- as.SpatialPolygons.GridTopology(g)
centroids <- coordinates(polys)
x <- centroids[,1]
y <- centroids[,2]
z <- 1.4 + 0.1*x + 0.2*y + 0.002*x^2
df <- data.frame(x=x, y=y, z=z, row.names=row.names(polys))
polygons(df) <- polys
class(df)
summary(df)
```

---

**Polygons-class Class "Polygons"**

**Description**

Collection of objects of class "Polygon"

**Objects from the Class**

Objects can be created by calls to the function Polygons
Slots

Polygons: Object of class "list": list with objects of class Polygon-class
plotOrder: Object of class "integer": order in which the Polygon objects should be plotted, currently by order of decreasing size
labpt: Object of class "numeric": pair of x, y coordinates giving a label point, the label point of the largest polygon component
ID: Object of class "character": unique identifier string
area: Object of class "numeric": the gross total planar area of the Polygon list but not double-counting holes (changed from 0.9-58 - islands are summed, holes are ignored rather than subtracted); these values are used to make sure that polygons of a smaller area are plotted after polygons of a larger area, does not respect projection as objects of this class have no projection defined

Methods

No methods defined with class "Polygons" in the signature.

Note

By default, single polygons (where Polygons is a list of length one) are not expected to be holes, but in multiple polygons, hole definitions for member polygons can be set. Polygon objects belonging to an Polygons object should either not overlap one-other, or should be fully included (as lakes or islands in lakes). They should not be self-intersecting. Checking of hole FALSE/TRUE status for Polygons objects is included in the maptools package using functions in the rgeos package, function checkPolygonsHoles().

Author(s)

Roger Bivand

polygons-methods

Retrieve polygons from SpatialPolygonsDataFrame object

Description

Retrieve polygons from SpatialPolygonsDataFrame object

Methods for polygons

obj = "SpatialPolygons" object of, or deriving from, SpatialPolygons
obj = "SpatialPolygonsDataFrame" object of, or deriving from, SpatialPolygonsDataFrame

Methods for "polygons<-"

object = "data.frame", value="SpatialPolygons" promote data.frame to object of class SpatialPolygonsDataFrame-class, by specifying polygons
read.asciigrid

read/write to/from (ESRI) asciigrid format

Description

read/write to/from ESRI asciigrid format

Usage

read.asciigrid(fname, as.image = FALSE, plot.image = FALSE, colname = fname, proj4string = CRS(as.character(NA)))
write.asciigrid(x, fname, attr = 1, na.value = -9999, ...)

Arguments

fname file name
as.image logical; if FALSE, a list is returned, ready to be shown with the image command; if FALSE an object of class SpatialGridDataFrame-class is returned
plot.image logical; if TRUE, an image of the map is plotted
colname alternative name for data column if not file name
proj4string A CRS object setting the projection arguments of the Spatial Grid returned
x object of class SpatialGridDataFrame
attr attribute column; if missing, the first column is taken; a name or a column number may be given
na.value numeric; value given to missing valued cells in the resulting map
... arguments passed to write.table, which is used to write the numeric data

Value

read.asciigrid returns the grid map read; either as an object of class SpatialGridDataFrame-class or, if as.image is TRUE, as list with components x, y and z.

Author(s)

Edzer Pebesma

See Also

as.image, SpatialGridDataFrame, image

Examples

x <- read.asciigrid(system.file("external/test.ag", package="sp")[[1]])
class(x)
image(x)
Methods for Function recenter in Package ‘sp’

Description

Methods for function recenter in package sp to shift or re-center geographical coordinates for a Pacific view. All longitudes < 0 are added to 360, to avoid for instance parts of Alaska being represented on the far left and right of a plot because they have values straddling 180 degrees. In general, using a projected coordinate reference system is to be preferred, but this method permits a geographical coordinate reference system to be used. This idea was suggested by Greg Snow, and corresponds to the two world representations in the maps package.

Methods

obj = "SpatialPolygons"  recenter a SpatialPolygons object
obj = "Polygons"  recenter a Polygons object
obj = "Polygon"  recenter an Polygon object
obj = "SpatialLines"  recenter a SpatialLines object
obj = "Lines"  recenter a Lines object
obj = "Line"  recenter an Line object

Examples

```r
crds <- matrix(c(179, -179, -179, 179, 50, 50, 52, 52), ncol=2)
SL <- SpatialLines(list(Lines(list(Line(crds)), "1")),
    CRS("+proj=longlat +ellps=WGS84"))
bbox(SL)
SLr <- recenter(SL)
bbox(SLr)
rcrds <- rbind(crds, crds[1,])
SpP <- SpatialPolygons(list(Polygons(list(Polygon(rcrds)), ID="r1")),
    proj4string=CRS("+proj=longlat +ellps=WGS84"))
bbox(SpP)
SpPr <- recenter(SpP)
bbox(SpPr)
```

```r
opar <- par(mfrow=c(1,2))
plot(SpP)
plot(SpPr)
par(opar)
crds <- matrix(c(-1, 1, 1, -1, 50, 50, 52, 52), ncol=2)
SL <- SpatialLines(list(Lines(list(Line(crds)), "1")),
    CRS("+proj=longlat +ellps=WGS84"))
bbox(SL)
SLr <- recenter(SL)
bbox(SLr)
rcrds <- rbind(crds, crds[1,])
SpP <- SpatialPolygons(list(Polygons(list(Polygon(rcrds)), ID="r1")),
    proj4string=CRS("+proj=longlat +ellps=WGS84"))
bbox(SpP)
SpPr <- recenter(SpP)
bbox(SpPr)
```
Rlogo

proj4string = CRS("+proj=longlat +ellps=WGS84")
bbox(SpP)
SpPr <- recenter(SpP)
bbox(SpPr)
opar <- par(mfrow=c(1,2))
plot(SpP)
plot(SpPr)
par(opar)

Rlogo jpeg image

Description

Rlogo jpeg image data as imported by getRasterData in the rgdal package

Usage

data(Rlogo)

Format

The format is: int [1:101, 1:77, 1:3] 255 255 255 255 255 255 255 255 255 255 ...

Examples

## Not run:
library(rgdal)
logo <- system.file("pictures/Rlogo.jpg", package="rgdal")[1]
x <- GDAL.open(logo)
gt = .Call("RGDAL_GetGeoTransform", x, PACKAGE="rgdal")
data <- getRasterData(x)
GDAL.close(x)

## End(Not run)
data(Rlogo)
d = dim(Rlogo)
cellsize = abs(c(gt[2],gt[6]))
cells.dim = c(d[1], d[2]) # c(d[2],d[1])
cellcentre.offset = c(x = gt[1] + 0.5 * cellsize[1], y = gt[4] - (d[2] - 0.5) * abs(cellsize[2]))
grid = GridTopology(cellcentre.offset, cellsize, cells.dim)
df = as.vector(Rlogo[,1])
for (band in 2:d[3]) df = cbind(df, as.vector(Rlogo[,band]))
df = as.data.frame(df)
names(df) = paste("band", 1:d[3], sep="")
Rlogo <- SpatialGridDataFrame(grid = grid, data = df)
summary(Rlogo)
spplot(Rlogo, zcol=1:3, names.attr=c("red","green","blue"),
col.regions=grey(0:100/100),
main="example of three-layer (RGB) raster image", as.table=TRUE)
select.spatial  

select points spatially

Description

select a number of points by digitizing the area they fall in

Usage

select.spatial(data, digitize = TRUE, pch = "+", rownames = FALSE)

Arguments

data  data object of class, or extending SpatialPoints; this object knows about its x and y coordinate
digitize  logical; if TRUE, points in a digitized polygon are selected; if FALSE, points identified by mouse clicks are selected
pch  plotting character used for points
rownames  logical; if FALSE, row (coordinate) numbers are returned; if TRUE and data contains a data.frame part, row.names for selected points in the data.frame are returned.

Value

if rownames == FALSE, array with either indexes (row numbers) of points inside the digitized polygon; if rownames == TRUE, character array with corresponding row names in the data.frame part

See Also

point.in.polygon, locator, SpatialPoints-class, SpatialPointsDataFrame-class

Examples

data(meuse)
## the following command requires user interaction: left mouse
## selects points, right mouse ends digitizing
data(meuse)
coordinates(meuse) = c("x", "y")
# select.spatial(meuse)
Description

This package provides S4 classes for importing, manipulating and exporting spatial data in R, and for methods including print/show, plot, subset, [], [[]], $, names, dim, summary, and a number of methods specific to spatial data handling.

Introduction

Several spatial statistical packages have been around for a long while, but no organized set of classes for spatial data has yet been devised. Many of the spatial packages make their own assumptions, or use their own class definitions for spatial data, making it inconvenient to move from one package to another. This package tries to provide a solid set of classes for many different types of spatial data. The idea is that spatial statistical packages will either support these classes (i.e., directly read and write them) or will provide conversion to them, so that we have a base class set with which any package can exchange. This way, many-to-many conversions can be replace with one-to-many conversions, provided either in this package or the spatial packages. Wherever possible conversion (coercion) functions are automatic, or provided by sp.

External packages that depend on sp will provide importing and exporting from and to external GIS formats, e.g. through GDAL, OGR or shapelib.

In addition, this package tries to provide convenient methods to print, summarize and plot such spatial data.

Dimensions

In principal, geographical data are two-dimensional, on a flat surface (a map) or on a sphere (the earth). This package provides space for dealing with higher dimensional data where possible; this is e.g. very simple for points and grids, but hard to do for polygons. Plotting functions are devised primarily for two-dimensional data, or two-dimensional projections of higher dimensional data.

Coordinate reference systems

Central to spatial data is that they have a coordinate reference system, which is coded in object of CRS class. Central to operations on different spatial data sets is that their coordinate reference system is compatible (i.e., identical).

This CRS can be a character string describing a reference system in a way understood by the PROJ.4 projection library, or a (character) missing value. An interface to the PROJ.4 library is available only if the R package rgdal is present.

Class structure

All spatial classes derive from a basic class Spatial, which only provides a bounding box and a CRS. This class has no useful instances, but useful derived classes.
SpatialPoints extends Spatial and has coordinates. The method coordinates extracts the numeric matrix with coordinates from an object of class SpatialPoints, or from other (possibly derived) classes that have points.

Objects of class SpatialGrid points on a regular grid. Either a full grid is stored or a partial grid (i.e., only the non-missing valued cells); calling coordinates on them will give the coordinates for the grid cells.

SpatialPoints, SpatialPixels and SpatialGrid can be of arbitrary dimension, although most of the effort is in making them work for two dimensional data.

SpatialLines provides lines, and SpatialPolygons provides polygons, i.e., lines that end where they start and do not intersect with itself. SpatialLines and SpatialPolygons only have two-dimensional data.

SpatialPointsDataFrame extends SpatialPoints with a data slot, having a data.frame with attribute data. Similarly, SpatialPixelsDataFrame, SpatialLinesDataFrame, SpatialPolygonsDataFrame extend the primary spatial information with attribute data.

References

PROJ.4: https://github.com/OSGeo/PROJ
GDAL and OGR: https://gdal.org/.

Authors

sp is a collaborative effort of Edzer Pebesma, Roger Bivand, Barry Rowlingson and Virgilo Gi’omez-Rubio.

sp-deprecated

Description

Deprecated functions is sp: getSpP*, getPolygon*, getLines* getSL*

Note

For overlay the new implementation is found in the over method; this works slightly different and more consistent.
Spatial-class

Class "Spatial"

Description

An abstract class from which useful spatial classes are derived

Usage

Spatial(bbox, proj4string = CRS(as.character(NA)))
## S3 method for class 'Spatial'
subset(x, subset, select, drop = FALSE, ...)

Arguments

bbox a bounding box matrix
proj4string a CRS object
x object of class Spatial
subset see subset.data.frame
select see subset.data.frame
drop see subset.data.frame
... passed through

Objects from the Class

are never to be generated; only derived classes can be meaningful

Slots

bbox: Object of class "matrix"; 2-column matrix holding the minimum in first and maximum in second column for the x-coordinate (first row), y-coordinate (second row) and optionally, for points and grids only, further coordinates. The constructed Spatial object will be invalid if any bbox values are NA or infinite. The column names must be c("min","max")

proj4string: Object of class "CRS". The name of this slot was chosen to reflect the use of Proj.4 strings to represent coordinate reference systems (CRS). The slot name will continue to be used, but as PROJ >= 6 and GDAL >= 3 are taken into use for reading files and for projection and transformation, the Proj.4 string CRS representation is being supplemented by a WKT2 (2019) representation. The reason for the modification is that important keys in the Proj.4 string representation are being deprecated in PROJ >= 6 and GDAL >= 3. Legacy "CRS" objects hold only a valid Proj.4 string, which can be used for unprojecting or reprojecting coordinates; it is initialised to NA. If the "CRS" object is instantiated using CRS() with rgdal using PROJ >= 6 and GDAL >= 3, the object may also have a WKT2 (2019) string carried as a comment. Non-NA strings may be checked for validity in the rgdal package, but attempts to assign a string containing "longlat" to data extending beyond longitude [-180, 360] or latitude [-90, 90] will be stopped or warned, use set.ll_warn to warn rather than stop, and set.ll_TOL to change the default tolerance for the range exceedance tests.
Methods

bbox signature(obj = "Spatial"): retrieves the bbox element

dimensions signature(obj = "Spatial"): retrieves the number of spatial dimensions spanned

gridded signature(obj = "Spatial"): logical, tells whether the data is on a regular spatial grid

plot signature(x = "Spatial", y = "missing"): plot method for spatial objects; does nothing but setting up a plotting region choosing a suitable aspect if not given (see below), colouring the plot background using either a bg= argument or par("bg"), and possibly drawing axes.

summary signature(object = "Spatial"): summarize object

$ retrieves attribute column

$<- sets or replaces attribute column, or promote a geometry-only object to an object having an attribute

rebuild_CRS rebuild a CRS object, usually used to add a WKT comment with PROJ >= 6 and GDAL >= 3

plot method arguments

The plot method for “Spatial” objects takes the following arguments:

x object of class Spatial
xlim default NULL; the x limits (x1, x2) of the plot
ylim default NULL; the y limits of the plot
asp default NA; the y/x aspect ratio
axes default FALSE; a logical value indicating whether both axes should be drawn
bg default par("bg"); colour to be used for the background of the device region
xaxs The style of axis interval calculation to be used for the x-axis
yaxs The style of axis interval calculation to be used for the y-axis
lab A numerical vector of the form c(x,y,len) which modifies the default way that axes are annotated

setParUsrBB default FALSE; set the par "usr" bounding box; see below
bgMap object of class ggmap, or returned by function RgoogleMaps::GetMap
expandBB numeric; factor to expand the plotting region default: bbox(x) with on each side (1=below, 2=left, 3=above and 4=right); defaults to c(0,0,0,0); setting xlim or ylim overrides this.

... passed through

Warning

this class is not useful in itself, but all spatial classes in this package derive from it
Note

The default aspect for map plots is 1; if however data are not projected (coordinates are longlat), the aspect is by default set to 1/cos(My * pi)/180) with My the y coordinate of the middle of the map (the mean of ylim, which defaults to the y range of bounding box). The argument setParUsrBB may be used to pass the logical value TRUE to functions within plot.Spatial. When set to TRUE, par("usr") will be overwritten with c(xlim,ylim), which defaults to the bounding box of the spatial object. This is only needed in the particular context of graphic output to a specified device with given width and height, to be matched to the spatial object, when using par("xaxs") and par("yaxs") in addition to par(mar=c(0,0,0,0)).

Author(s)

r-spatial team; Edzer Pebesma, <edzer.pebesma@uni-muenster.de> Roger Bivand, Barry Rowlingson, Virgilio Gómez-Rubio

See Also

SpatialPoints-class, SpatialGrid-class,
SpatialPointsDataFrame-class, SpatialGridDataFrame-class

Examples

```r
o <- new("Spatial")
proj4string(o) <- CRS("+init=epsg:27700")
if (!is.null(comment(slot(o, "proj4string")))) {
  cat(strsplit(wkt(o), "\n")[[1]], sep="\n")
  cat(strsplit(wkt(slot(o, "proj4string")), "\n")[[1]], sep="\n")
}
```

SpatialGrid-class Class "SpatialGrid"

Description

class for defining a full, rectangular grid of arbitrary dimension

Objects from the Class

Objects are created by using e.g.

SpatialGrid(grid)

with grid of class GridTopology-class

Slots

ggrid: object of class GridTopology-class, defining the grid topology (offset, cellsize, dim)
bbox: Object of class "matrix"; bounding box
proj4string: Object of class "CRS"; projection
SpatialGridDataFrame-class

Extends

Class "SpatialPoints" directly; Class "Spatial", by class "SpatialPoints".

Methods

coordinates signature(x = "SpatialGrid"): calculates coordinates for each point on the grid; coordinates are not stored in objects of class SpatialGrid

summary signature(object = "SpatialGrid"): summarize object

plot signature(x = "SpatialGrid"): plots cell centers

"[" signature(x = "SpatialGrid"): select rows and columns

Author(s)

Edzer Pebesma, <edzer.pebesma@uni-muenster.de>

See Also

SpatialGridDataFrame-class, SpatialGrid

Examples

x = GridTopology(c(0,0), c(1,1), c(5,5))
class(x)
x
summary(x)
coordinates(x)
y = SpatialGrid(grid = x)
class(y)
y

SpatialGridDataFrame-class

Class "SpatialGridDataFrame"

Description

Class for spatial attributes that have spatial locations on a (full) regular grid.

Objects from the Class

Objects can be created by calls of the form as(x, "SpatialGridDataFrame"), where x is of class SpatialPixelsDataFrame-class, or by importing through rgdal. Ordered full grids are stored instead or unordered non-NA cells;
SpatialGridDataFrame-class

Slots
- grid: see GridTopology-class; grid parameters
- bbox: Object of class "matrix"; bounding box
- proj4string: Object of class "CRS"; projection
- data: Object of class data.frame, containing attribute data

Extends
Class "SpatialGrid", directly. Class "Spatial", by class "SpatialGrid".

Methods
- coordinates signature(x = "SpatialGridDataFrame"): retrieves (and calculates!) coordinates
- [ signature(x = "SpatialGridDataFrame"): selects rows, columns, and attributes; returns an object of class SpatialGridDataFrame
- as.matrix signature(x = "SpatialGridDataFrame"): coerce to matrix; increasing col index corresponds to decreasing y coordinate, row index increases with coordinate index
- as.array signature(x = "SpatialGridDataFrame"): coerce to array; increasing array index for the second dimension corresponds to decreasing coordinates, all other coordinate dimensions increase with array index
- cbind signature(...): if arguments have identical topology, combine their attribute values

Plot method arguments
The plot methods for “SpatialPixelsDataFrame” or “SpatialGridDataFrame” objects take the following arguments:
- x object of class SpatialPixelsDataFrame or SpatialGridDataFrame
- ... arguments passed on to image.SpatialGridDataFrame
- attr integer or character, indicating the attribute variable to be plotted; default 1
- col color ramp to be used; default bpy.colors(100) for continuous, or RColorBrewer::brewer_pal(nlevels(x[[1]]),"Set2") for factor variables
- breaks for continuous attributes: values at which color breaks should take place
- zlim for continuous attributes: numeric of length 2, specifying the range of attribute values to be plotted; default to data range range(as.numeric(x[[attr]])[is.finite(x[[attr]])])
- axes logical: draw x and y axes? default FALSE
- xaxs character, default "i", see par
- yaxs character, default equal to xaxs, see par
- at numeric or NULL, values at which axis tics and labels should be drawn; default NULL (use pretty)
- border color, to be used for drawing grid lines; default NA (don’t draw grid lines)
- axis.pos integer, 1-4; default 4, see axis
- add.axis logical: draw axis along scale? default TRUE
what  what to draw: "image", "scale", or "both"; default "both"
scale.size  size for the scale bar; use lcm to specify in absolute size, or a numeric value such as 1/6 to specify relative size; default lcm(2.8)
scale.shrink  non-negative numeric indicating the amount to shrink the scale length, default 0
scale.frac  for categorical attributes: numeric between 0 and 1, indicating the scale width, default 0.3
scale.n  for categorical attributes: integer, indicating how many scale categories should fill a complete width; default 15

Author(s)
Edzer Pebesma, <edzer.pebesma@uni-muenster.de>

See Also
SpatialGrid-class, which does not contain the attribute data, and SpatialPixelsDataFrame-class which holds possibly incomplete grids
Plotting gridded data with sp: https://www.r-spatial.org/r/2016/03/08/plotting-spatial-grids.html

Examples
data(meuse.grid) # only the non-missing valued cells
coordinates(meuse.grid) = c("x", "y") # promote to SpatialPointsDataFrame
gridded(meuse.grid) <- TRUE # promote to SpatialPixelsDataFrame
x = as(meuse.grid, "SpatialGridDataFrame") # creates the full grid
x[['idist']] = 1 - x[['dist']] # assigns new attribute
image(x[['idist']]) # note the single [ for attribute selection

# toy example:
df = data.frame(z = c(1:6, NA, 8, 9),
               xc = c(1,1,1,2,2,2,3,3,3),
               yc = c(rep(c(0, 1.5, 3),3)))
coordinates(df) = ~xc+yc
gridded(df) = TRUE
df = as(df, "SpatialGridDataFrame") # to full grid
image(df[['z']])

# draw labels to verify:
cc = coordinates(df)
z=df[['z']] 
zc=as.character(z) 
zc[is.na(zc)]="NA"
text(cc[,1],cc[,2],zc)

# the following is weird, but illustrates the concept of row/col selection:
fullgrid(meuse.grid) = TRUE
image(meuse.grid)
image(meuse.grid[20:70, 10:70, "dist"], add = TRUE, col = bpy.colors())

# as.matrix, as.array
sgdim = c(3,4)
SG = SpatialGrid(GridTopology(rep(0,2), rep(10,2), sgdim))
SGDF = SpatialGridDataFrame(SG, data.frame(val = 1:12))
as.array(SGDF)
as.matrix(SGDF)
as(SGDF, "array")

**SpatialLines**

create objects of class `SpatialLines` or `SpatialLinesDataFrame`

**Description**

create objects of class `SpatialLines` or `SpatialLinesDataFrame` from lists of `Lines` objects and data.frames; extract list of `Lines` from a `SpatialLines` object

**Usage**

```r
SpatialLines(LinesList, proj4string = CRS(as.character(NA)))
SpatialLinesDataFrame(sl, data, match.ID = TRUE)
as.SpatialLines.SLDF(SLDF)
getSpatialLinesMidPoints(SL)
LineLength(cc, longlat = FALSE, sum = TRUE)
LinesLength(Ls, longlat = FALSE)
SpatialLinesLengths(SL, longlat)
```

**Arguments**

- **LinesList** list with objects of class `Lines-class`
- **proj4string** Object of class "CRS": holding a valid proj4 string
- **sl, SL** object of class `SpatialLines-class`
- **data** object of class data.frame; the number of rows in data should equal the number of `Lines` elements in sl
- **match.ID** logical; (default TRUE): match `SpatialLines` member `Lines` ID slot values with data.frame row names, and re-order the data frame rows if necessary; if character: indicates the column in data with `Lines` IDs to match
- **SLDF** `SpatialLinesDataFrame` object
- **Ls** Object of class `Lines`
- **cc** Object of class `Line`, or two-column matrix with points
- **longlat** if FALSE, Euclidean distance, if TRUE Great Circle distance in kilometers
- **sum** logical; if TRUE return scalar length of sum of segments in `Line`, if FALSE return vector with segment lengths
SpatialLines-class

Value

SpatialLines returns object of class SpatialLines; SpatialLinesDataFrame returns object of class SpatialLinesDataFrame getSpatialLinesMidPoints returns an object of class SpatialPoints, each point containing the (weighted) mean of the lines elements; weighted in the sense that mean is called twice.

See Also

SpatialLines-class

Description

a class that holds spatial lines

Objects from the Class

hold a list of Lines objects; each Lines object holds a list of Line (line) objects.

Slots

lines: Object of class "list"; list members are all of class Lines-class
bbox: Object of class "matrix"; see Spatial-class
proj4string: Object of class "CRS"; see CRS-class

Extends

Class "Spatial", directly.

Methods

[ signature(obj = "SpatialLines"): select subset of (sets of) lines; NAs are not permitted in the row index
coordinates value is a list of lists with matrices
plot signature(x = "SpatialLines", y = "missing"): plot lines in SpatialLines object
lines signature(x = "SpatialLines"): add lines in SpatialLines object to a plot
rbind signature(object = "SpatialLines"): rbind-like method, see notes
summary signature(object = "SpatialLines"): summarize object
plot method arguments

The plot method for “SpatialLines” objects takes the following arguments:

- **x**: object of class SpatialLines
- **xlim**: default NULL; the x limits (x1, x2) of the plot
- **ylim**: default NULL; the y limits of the plot
- **col**: default 1; default plotting color
- **lwd**: default 1; line width
- **lty**: default 1; line type
- **add**: default FALSE; add to existing plot
- **axes**: default FALSE; a logical value indicating whether both axes should be drawn
- **lend**: default 0; line end style
- **ljoin**: default 0; line join style
- **lmitre**: default 10; line mitre limit
- and... passed through

**setParUsrBB** set the par “usr” bounding box, see note in Spatial-class

Note

*rbind* calls the function SpatialLines, where it is checked that all IDs are unique. If *rbinding* SpatialLines without unique IDs, it is possible to set the argument makeUniqueIDs = TRUE, although it is preferred to change these explicitly with spChFIDs.

Author(s)

Roger Bivand, Edzer Pebesma

See Also

Line-class, Lines-class

Examples

```r
# from the sp vignette:
l1 = cbind(c(1,2,3),c(3,2,2))
rownames(l1) = letters[1:3]
l1a = cbind(l1[,1] + .05, l1[,2] + .05)
rownames(l1a) = letters[1:3]
l2 = cbind(c(1,2,3),c(1,1.5,1))
rownames(l2) = letters[1:3]
S1l = Line(l1)
S1la = Line(l1a)
S12 = Line(l2)
S1 = Lines(list(S1l, S1la), ID="a")
S2 = Lines(list(S12), ID="b")
S1 = SpatialLines(list(S1,S2))
```
SpatialLinesDataFrame-class

a class for spatial lines with attributes

Description

this class holds data consisting of (sets of lines), where each set of lines relates to an attribute row in a data.frame

Objects from the Class

can be created by the function SpatialLinesDataFrame

Slots

data: Object of class data.frame containing the attribute table
lines: Object of class "list"; see SpatialLines-class
bbox: Object of class "matrix"; see Spatial-class
proj4string: Object of class "CRS"; see CRS-class

Extends

Class "SpatialLines", directly. Class "Spatial", by class "SpatialLines".

Methods

Methods defined with class "SpatialLinesDataFrame" in the signature:

subset rows or columns; in case of row subsetting, the line sets are also subsetted; NAs are not permitted in the row index

coordinates signature(obj = "SpatialLinesDataFrame"): retrieves a list with lists of coordinate matrices

show signature(object = "SpatialLinesDataFrame"): print method

plot signature(x = "SpatialLinesDataFrame"): plot points

lines signature(object = "SpatialLinesDataFrame"): add lines to plot

rbind signature(object = "SpatialLinesDataFrame"): rbind-like method

Note

rbind for SpatialLinesDataFrame is only possible for objects with unique IDs. If you want to rbind objects with duplicated IDs, see spChFIDs.
Author(s)
Roger Bivand; Edzer Pebesma

See Also
SpatialLines-class

SpatialMultiPoints
create objects of class SpatialMultiPoints or SpatialMultiPointsDataFrame-class from coordinates, and from coordinates and data.frames

Usage
SpatialMultiPoints(coords, proj4string=CRS(as.character(NA)), bbox = NULL)
SpatialMultiPointsDataFrame(coords, data, proj4string = CRS(as.character(NA)), match.ID, bbox = NULL)

Arguments
coords list with in each element a numeric matrix or data.frame with coordinates (each row representing a point); in case of SpatialMultiPointsDataFrame an object of class SpatialMultiPoints-class is also allowed
proj4string projection string of class CRS-class
bbox bounding box matrix, usually NULL and constructed from the data, but may be passed through for coercion purposes if clearly needed
data object of class data.frame; the number of rows in data should equal the number of points in the coords object
match.ID logical or character; if missing, and coords and data both have row names, and their order does not correspond, matching is done by these row names and a warning is issued; this warning can be suppressed by setting match.ID to TRUE. If TRUE AND coords has non-automatic rownames (i.e., coerced to a matrix by as.matrix, dimnames(coords)[[1]] is not NULL), AND data has row.names (i.e. is a data.frame), then the SpatialMultiPointsDataFrame object is formed by matching the row names of both components, leaving the order of the coordinates in tact. Checks are done to see whether both row names are sufficiently unique, and all data are matched. If FALSE, coordinates and data are simply "glued" together, ignoring row names. If character: indicates the column in data with coordinates IDs to use for matching records. See examples below.
SpatialMultiPoints-class

Value

SpatialMultiPoints returns an object of class SpatialMultiPoints; SpatialMultiPointsDataFrame returns an object of class SpatialMultiPointsDataFrame;

See Also

coordinates, SpatialMultiPoints-class, SpatialMultiPointsDataFrame-class

Examples

cl1 = cbind(rnorm(3, 10), rnorm(3, 10))
c12 = cbind(rnorm(5, 10), rnorm(5, 0))
c13 = cbind(rnorm(7, 0), rnorm(7, 10))

mp = SpatialMultiPoints(list(cl1, cl2, cl3))
mpx = rbind(mp, mp) # rbind method
plot(mp, col = 2, cex = 1, pch = 1:3)
mp
mp[1:2]

print(mp, asWKT=TRUE, digits=3)

mpdf = SpatialMultiPointsDataFrame(list(cl1, cl2, cl3), data.frame(a = 1:3))
mpdf
mpdfx = rbind(mpdf, mpdf) # rbind method

plot(mpdf, col = mpdf$a, cex = 1:3)
as(mpdf, "data.frame")
mpdf[1:2,]

SpatialMultiPoints-class

Class "SpatialMultiPoints"

Description

Class for (irregularly spaced) MultiPoints

Objects from the Class

Objects can be created by calls of the form SpatialPoints(x).

Slots

cords: Object of class "list", containing the coordinates of point sets (each list element is a matrix)
bbox: Object of class "matrix", with bounding box
proj4string: Object of class "CRS", projection string
SpatialMultiPoints-class

Extends

Class "Spatial", directly.

Methods

[ signature(x = "SpatialMultiPoints"): subsets point sets
coerce  signature(from = "SpatialPoints", to = "data.frame"): coerce to data.frame
coordinates  signature(obj = "SpatialMultiPoints"): retrieves all the coordinates, as one single matrix
plot  signature(x = "SpatialPoints", y = "missing"): plot points
summary  signature(object = "SpatialPoints"): summarize object
points  signature(x = "SpatialPoints"): add point symbols to plot
show  signature(object = "SpatialPoints"): prints coordinates
rbind  signature(object = "SpatialPoints"): rbind-like method

plot method arguments

The plot method for “SpatialPoints” objects takes the following arguments:

x  object of class SpatialPoints
pch  default 3; either an integer specifying a symbol or a single character to be used as the default in plotting points
axes  default FALSE; a logical value indicating whether both axes should be drawn
add  default FALSE; add to existing plot
xlim  default NULL; the x limits (x1, x2) of the plot
ylim  default NULL; the y limits of the plot
... passed through
setParUsrBB  default FALSE; set the par “usr” bounding box, see note in Spatial-class

cex  default 1; numerical value giving the amount by which plotting text and symbols should be magnified relative to the default
col  default 1; default plotting color
lwd  default 1; line width
bg  default 1; colour to be used for the background of the device region

Author(s)

Edzer Pebesma, <edzer.pebesma@uni-muenster.de>

See Also

SpatialMultiPointsDataFrame-class SpatialPoints-class
Examples

c11 = cbind(rnorm(3, 10), rnorm(3, 10))
c12 = cbind(rnorm(5, 10), rnorm(5, 0))
c13 = cbind(rnorm(7, 0), rnorm(7, 10))

mp = SpatialMultiPoints(list(c11, c12, c13))
plot(mp, col = 2, cex = 1, pch = 1:3)
mp
mp[1:2]

print(mp, asWKT=TRUE, digits=3)

---

SpatialMultiPointsDataFrame-class

Class "SpatialMultiPointsDataFrame"

Description

Class for spatial attributes that correspond to point sets

Usage

```r
## S4 method for signature 'SpatialMultiPointsDataFrame'
x[i, j, ..., drop = TRUE]
## S4 method for signature 'SpatialMultiPointsDataFrame, data.frame'
coerce(from, to, strict = TRUE)
## S4 method for signature 'SpatialMultiPointsDataFrame'
coordinates(obj)
## S4 method for signature 'SpatialMultiPointsDataFrame'
show(object)
## S4 method for signature 'SpatialMultiPointsDataFrame'
points(x)
```

Arguments

- `x`: SpatialMultiPointsDataFrame object
- `from`: class to which to coerce
- `strict`: see `as`
- `i`: row indices
- `j`: column indices
- `drop`: see `Extract`
- `...`: indices passed through
SpatialPixels

Slots

data: Object of class data.frame containing the attribute data (may or may not contain the coordi-
nates in its columns)

coords: Object of class "list"; the list with coordinates matrices; points are rows in the matrix,
the list length equals the number of rows in the data slot

bbox: Object of class "matrix"; bounding box

proj4string: Object of class "CRS"; projection string

Extends

Class "SpatialMultiPoints", directly. Class "Spatial", by class "SpatialMultiPoints".

Author(s)

Edzer Pebesma, <edzer.pebesma@uni-muenster.de>

See Also

coordinates, SpatialMultiPoints-class

Examples

# create three sets of points:
c11 = cbind(rnorm(3, 10), rnorm(3, 10))
c12 = cbind(rnorm(5, 10), rnorm(5, 0))
c13 = cbind(rnorm(7, 0), rnorm(7, 10))

mpdf = SpatialMultiPointsDataFrame(list(c11, c12, c13), data.frame(a = 1:3))
mpdf

plot(mpdf, col = mpdf$a, cex = 1:3)
as(mpdf, "data.frame")
mpdf[1:2,]

---

SpatialPixels  
**define spatial grid**

Description

defines spatial grid by offset, cell size and dimensions
Usage

GridTopology(cellcentre.offset, cellsize, cells.dim)
SpatialPixels(points, tolerance = sqrt(.Machine$double.eps),
proj4string = CRS(as.character(NA)), round = NULL, grid = NULL)
SpatialGrid(grid, proj4string = CRS(as.character(NA)))
coordinatevalues(obj)
points2grid(points, tolerance = sqrt(.Machine$double.eps), round=NULL)
getGridIndex(cc, grid, all.inside = TRUE)
getGridTopology(obj)
areaSpatialGrid(obj)

Arguments

  cellcentre.offset numeric; vector with the smallest centroid coordinates for each dimension; co-
          ordinates refer to the cell centre
  cellsize numeric; vector with the cell size in each dimension
  cells.dim integer; vector with number of cells in each dimension
  points coordinates, object of class SpatialPoints-class
  grid grid topology; object of class GridTopology-class; for calls to SpatialPixels,
          a value of NULL implies that this will be derived from the point coordinates
  tolerance precision, used to which extent points are exactly on a grid
  round default NULL, otherwise a value passed to as the digits argument to round for
          setting cell size
  proj4string object of class CRS-class
  obj object of class or deriving from SpatialGrid-class
  cc numeric matrix with coordinates
  all.inside logical; if TRUE and cc points fall outside the grid area, an error message is
          generated; if FALSE, NA values are generated for such points

Value

GridTopology returns a value of class GridTopology-class; SpatialGrid returns an object of class
SpatialGrid-class
coordinatevalues returns a list with the unique x-coordinates, the unique y-coordinate, etc. in-
stead of the coordinates of all grid cells
SpatialGrid returns an object of class SpatialGrid-class.
points2grid returns the GridTopology-class from a set of points.
getGridIndex finds the index of a set of point coordinates in a given grid topology, and depending
on all.inside setting, generates NA or an error message if points are outside the grid domain.
getGridTopology returns the slot of class GridTopology-class from obj.
areaSpatialGrid returns the spatial area of (the non-missing valued cells of) the grid. For objects of
class SpatialGridDataFrame-class the area refers to cells where any (one or more) of the attribute
columns are non-missing valued.
Note

SpatialGrid stores grid topology and may or may not store the coordinates of the actual points, which may form a subset of the full grid. To find out or change this, see `fullgrid`.

points2grid tries to figure out the grid topology from points. It succeeds only if points on a grid line have constant y column, and points on a grid column have constant x coordinate, etc. In other cases, use signif on the raw coordinate matrices to make sure this is the case.

Author(s)

Edzer Pebesma, <edzer.pebesma@uni-muenster.de>

See Also

SpatialGrid-class, SpatialGridDataFrame-class.

Examples

```r
x = GridTopology(c(0,0), c(1,1), c(5,4))
class(x)
x
summary(x)
coordinates(x)
coordinatevalues(x)
data(meuse.grid)
coordinates(meuse.grid) <- c("x", "y")
points2grid(meuse.grid)
data(meuse.grid)
set.seed(1)
meuse.grid$x <- meuse.grid$x + rnorm(length(meuse.grid$x), 0, 0.002)
meuse.grid$y <- meuse.grid$y + rnorm(length(meuse.grid$y), 0, 0.002)
coordinates(meuse.grid) <- c("x", "y")
#EJP
# points2grid(meuse.grid, tolerance=0.76, round=1)
data(meuse.grid)
a <- which(meuse.grid$x == 180140)
b <- which(meuse.grid$x == 180180)
c <- which(meuse.grid$x == 179260)
d <- which(meuse.grid$y == 332460)
e <- which(meuse.grid$y == 332420)
f <- which(meuse.grid$y == 330740)
meuse.grid <- meuse.grid[-c(a, b, c, d, e, f),]
coordinates(meuse.grid) <- c("x", "y")
points2grid(meuse.grid)
data(meuse.grid)
set.seed(1)
meuse.grid$x <- meuse.grid$x + rnorm(length(meuse.grid$x), 0, 0.002)
meuse.grid$y <- meuse.grid$y + rnorm(length(meuse.grid$y), 0, 0.002)
meuse.grid <- meuse.grid[-c(a, b, c, d, e, f),]
coordinates(meuse.grid) <- c("x", "y")
# EJP
```
SpatialPixels-class

Class "SpatialPixels"

Description

class for defining a pixels, forming a possibly incomplete rectangular grid of arbitrary dimension

Objects from the Class

Objects are created by using e.g.
SpatialPixels(points)
with points of class SpatialPoints-class

Slots

grid object of class GridTopology-class, defining the grid topology (offset, cellsize, dim)
grid.index integer; index of points in full grid
coords coordinates of points, or bbox of gridbbox: Object of class "matrix": bounding boxproj4string: Object of class "CRS": projection

Extends

Class "SpatialPoints" directly; Class "Spatial", by class "SpatialPoints".

Methods

coordinates signature(x = "SpatialPixels"): calculates coordinates for each point on the grid; coordinates are not stored in objects of class SpatialGrid
summary signature(object = "SpatialPixels"): summarize object
plot signature(x = "SpatialPixels"): plots cell centers
"[l" signature(x = "SpatialPixels"): select pixel cells; the argument drop=FALSE (default) does not recalculate grid topology for the selection, if drop=TRUE the grid topology is recomputed, and might change.
rbind signature(x = "SpatialPixels"): rbind-like method

Author(s)

Edzer Pebesma, <edzer.pebesma@uni-muenster.de>

See Also

SpatialPixelsDataFrame-class, SpatialGrid-class
SpatialPixelsDataFrame

Examples

```r
data(meuse.grid)
pts = meuse.grid[, c("x", "y")]
y = SpatialPixels(SpatialPoints(pts))
class(y)
y
summary(y)
plot(y) # plots grid
plot(y, grid = FALSE) # plots points
```

---

**SpatialPixelsDataFrame**

*define spatial grid with attribute data*

---

**Description**

defines spatial grid by offset, cell size and dimensions

**Usage**

```r
SpatialPixelsDataFrame(points, data, tolerance = sqrt(.Machine$double.eps),
proj4string = CRS(as.character(NA)), round = NULL, grid = NULL)
SpatialGridDataFrame(grid, data, proj4string = CRS(as.character(NA)))
```

**Arguments**

- `points` coordinates, either as numeric matrix or as object of class `SpatialPoints-class`
- `grid` grid topology; object of class `GridTopology-class`; for calls to `SpatialPixelsDataFrame`
a value of `NULL` implies that this will be derived from the point coordinates
- `data` data.frame; contains the attribute (actual grid) data
- `tolerance` precision up to which extent points should be exactly on a grid
- `round` default `NULL`, otherwise a value passed to as the digits argument to `round` for setting cell size
- `proj4string` object of class `CRS-class` in the first form only used when `points` does not inherit from `Spatial-class`

**Value**

- `SpatialPixelsDataFrame` returns an object of class `SpatialPixelsDataFrame-class`
- `SpatialGridDataFrame` returns an object of class `SpatialGridDataFrame-class`

**Note**

SpatialPixels stores grid topology and coordinates of the actual points, which may be in the form of a subset (set of pixels) of a full grid. To find out or change this, see `fullgrid` and `SpatialGrid-class`. 
Author(s)

Edzer Pebesma

See Also

girded, gridded<-, SpatialGrid, SpatialGrid-class

Examples

data(meuse.grid)
m = SpatialPixelsDataFrame(points = meuse.grid[c("x", "y")], data = meuse.grid)
class(m)
summary(m)

SpatialPixelsDataFrame-class

Class "SpatialPixelsDataFrame"

Description

Class for spatial attributes that have spatial locations on a regular grid.

Objects from the Class

Objects can be created by calls of the form as(x, "SpatialPixelsDataFrame"), where x is of class SpatialPointsDataFrame-class, or by importing through rgdal. Ordered full grids are stored instead of unordered non-NA cells;

Slots

bbox: Object of class "matrix": bounding box
proj4string: Object of class "CRS": projection
coords: see SpatialPoints: points slot
coords.nrs see SpatialPointsDataFrame
grid: see GridTopology-class: grid parameters
grid.index: integer; index of points in the list to points in the full (ordered) grid. x cycles fastest; all coordinates increase from low to high except y, which decreases from high to low
data: Object of class data.frame, containing the attribute data

Extends

Class "SpatialPixels", directly. Class "Spatial", by class "SpatialPixels".
Methods

coordinates signature(x = "SpatialPixelsDataFrame"): retrieves coordinates

[ signature(x = "SpatialPixelsDataFrame"): selects row(s) and/or attribute(s), and returns an object of class SpatialPixelsDataFrame; rows refer here to the pixel numbers, not grid lines. For selecting a square block in a grid, coerce to a SpatialGridDataFrame-class first, and use [ on that object.

as.matrix signature(x = "SpatialPixelsDataFrame"): coerce to matrix

rbind signature(x = "SpatialPixelsDataFrame"): rbind-like method

plot signature(x = "SpatialPixelsDataFrame", y = "missing"): see SpatialGridDataFrame-class for details

Author(s)

Edzer Pebesma, <edzer.pebesma@uni-muenster.de>

See Also

SpatialPixels-class, which does not contain the attribute data

Examples

data(meuse.grid) # only the non-missing valued cells
coordinates(meuse.grid) = c("x", "y") # promote to SpatialPointsDataFrame
gridded(meuse.grid) <- TRUE # promote to SpatialPixelsDataFrame
meuse.grid["idist"] = 1 - meuse.grid["dist"] # assigns new attribute
image(meuse.grid["idist"]) # note the single [ ]

# toy example:
df = data.frame(z = c(1:6,NA,8,9),
            xc = c(1,1,1,2,2,2,3,3,3),
            yc = c(rep(c(0, 1.5, 3),3)))
coordinates(df) = ~xc+yc
gridded(df) = TRUE
image(df["z"]) # draw labels to verify:
cc = coordinates(df)
z=df["z"]
zc=as.character(z)
zc[is.na(zc)]="NA"
text(cc[,1],cc[,2],zc)
SpatialPoints

create objects of class SpatialPoints or SpatialPointsDataFrame

Description

create objects of class SpatialPoints-class or SpatialPointsDataFrame-class from coordinates, and from coordinates and data.frames

Usage

SpatialPoints(coords, proj4string=CRS(as.character(NA)), bbox = NULL)
SpatialPointsDataFrame(coords, data, coords.nrs = numeric(0),
proj4string = CRS(as.character(NA)), match.ID, bbox = NULL)

Arguments

coops numeric matrix or data.frame with coordinates (each row is a point); in case of SpatialPointsDataFrame an object of class SpatialPoints-class is also allowed
proj4string projection string of class CRS-class
bbox bounding box matrix, usually NULL and constructed from the data, but may be passed through for coercion purposes if clearly needed
data object of class data.frame; the number of rows in data should equal the number of points in the coords object
coords.nrs numeric; if present, records the column positions where in data the coordinates were taken from (used by coordinates<-)
match.ID logical or character; if missing, and coords and data both have row names, and their order does not correspond, matching is done by these row names and a warning is issued; this warning can be suppressed by setting match.ID to TRUE. If TRUE AND coords has non-automatic rownames (i.e., coerced to a matrix by as.matrix, dimnames(coords)[[1]] is not NULL), AND data has row.names (i.e. is a data.frame), then the SpatialPointsDataFrame object is formed by matching the row names of both components, leaving the order of the coordinates in tact. Checks are done to see whether both row names are sufficiently unique, and all data are matched. If FALSE, coordinates and data are simply "glued" together, ignoring row names. If character: indicates the column in data with coordinates IDs to use for matching records. See examples below.

Value

SpatialPoints returns an object of class SpatialPoints; SpatialPointsDataFrame returns an object of class SpatialPointsDataFrame;

See Also

coordinates, SpatialPoints-class, SpatialPointsDataFrame-class
Examples

```r
set.seed(1331)
pts = cbind(1:5, 1:5)
dimnames(pts)[[1]] = letters[1:5]
df = data.frame(a = 1:5)
row.names(df) = letters[5:1]

library(sp)
options(warn=1) # show warnings where they occur
SpatialPointsDataFrame(pts, df) # warn
SpatialPointsDataFrame(pts, df, match.ID = TRUE) # don't warn
SpatialPointsDataFrame(pts, df, match.ID = FALSE) # don't warn

df$m = letters[5:1]
SpatialPointsDataFrame(pts, df, match.ID = "m") # don't warn

dimnames(pts)[[1]] = letters[5:1]
SpatialPointsDataFrame(pts, df) # don't warn: ID matching doesn't reorder
```

SpatialPoints-class  
Class "SpatialPoints"

Description

Class for (irregularly spaced) points

Objects from the Class

Objects can be created by calls of the form SpatialPoints(x).

Slots

- coords: Object of class "matrix", containing the coordinates (each row is a point)
- bbox: Object of class "matrix", with bounding box
- proj4string: Object of class "CRS", projection string

Extends

Class "Spatial", directly.

Methods

- `[ signature(x = "SpatialPoints")`: subsets the points; only rows (points) can be subsetted
- `coerce signature(from = "SpatialPoints",to = "data.frame")`: retrieves the data part
- `coerce signature(from = "SpatialPoints",to = "SpatialPixels")`: equivalent to assigning gridded TRUE for a copy of the object
- `coerce signature(from = "SpatialPointsDataFrame",to = "SpatialPixelsDataFrame")`: equivalent to assigning gridded TRUE for a copy of the object
 coerce signature(from = "data.frame", to = "SpatialPoints"): sets coordinates, which may be in a data frame
 coerce signature(from = "matrix", to = "SpatialPoints"): set coordinates, which may be in a matrix
 coordinates signature(obj = "SpatialPoints"): retrieves the coordinates, as matrix
 plot signature(x = "SpatialPoints", y = "missing"): plot points
 summary signature(object = "SpatialPoints"): summarize object
 points signature(x = "SpatialPoints"): add point symbols to plot
 show signature(object = "SpatialPoints"): prints coordinates
 rbind signature(object = "SpatialPoints"): rbind-like method

 plot method arguments
 The plot method for “SpatialPoints” objects takes the following arguments:
 x object of class SpatialPoints
 pch default 3; either an integer specifying a symbol or a single character to be used as the default in plotting points
 axes default FALSE; a logical value indicating whether both axes should be drawn
 add default FALSE; add to existing plot
 xlim default NULL; the x limits (x1, x2) of the plot
 ylim default NULL; the y limits of the plot
 ... passed through
 setParUsrBB default FALSE; set the par “usr” bounding box, see note in Spatial-class
 cex default 1; numerical value giving the amount by which plotting text and symbols should be magnified relative to the default
 col default 1; default plotting color
 lwd default 1; line width
 bg default 1; colour to be used for the background of the device region

 Author(s)
 Edzer Pebesma, <edzer.pebesma@uni-muenster.de>

 See Also
 SpatialPointsDataFrame-class

 Examples
 x = c(1,2,3,4,5)
y = c(3,2,5,1,4)
S <- SpatialPoints(cbind(x,y))
S <- SpatialPoints(list(x,y))
S <- SpatialPoints(data.frame(x,y))
S
plot(S)
Class "SpatialPointsDataFrame"

Description

Class for spatial attributes that have spatial point locations

Usage

```r
## S4 method for signature 'SpatialPointsDataFrame'
x[i, j, ..., drop = TRUE]
## S4 method for signature 'SpatialPointsDataFrame, SpatialPoints'
coerce(from, to, strict = TRUE)
## S4 method for signature 'SpatialPointsDataFrame, data.frame'
coerce(from, to, strict = TRUE)
## S4 method for signature 'SpatialPointsDataFrame'
coordinates(obj)
## S4 method for signature 'SpatialPointsDataFrame'
show(object)
## S4 method for signature 'SpatialPointsDataFrame'
points(x)
## S3 method for class 'SpatialPointsDataFrame'
rbind(...)
```

Arguments

- `x, from, obj, object`: SpatialPointsDataFrame object
- `to`: class to which to coerce
- `strict`: see `as`
- `i`: row indices
- `j`: column indices
- `drop`: see `Extract`
- `...`: indices passed through

Objects from the Class

Objects can be created by calls of the form `coordinates(x) = c("x", "y")`. or of the form `coordinates(x) = xy`; see `coordinates`. 
Slots

data: Object of class data.frame containing the attribute data (may or may not contain the coordinates in its columns)

coords: Object of class "matrix": the coordinates matrix (points are rows in the matrix)

coords.nrs Object of class logical; if TRUE, when the object was created the coordinates were retrieved from the data.frame, and hence stripped from it; after coercion to data.frame, e.g. by as.data.frame(x), coordinates will again be added (as first few columns) to the data.frame

bbox: Object of class "matrix": bounding box

proj4string: Object of class "CRS": projection string

Extends

Class "SpatialPoints", directly. Class "Spatial", by class "SpatialPoints".

Author(s)

Edzer Pebesma, <edzer.pebesma@uni-muenster.de>

See Also

coordinates, SpatialPoints-class

Examples

data(meuse)
xy = meuse[c("x", "y")]) # retrieve coordinates as data.frame
class(meuse)
data(meuse) # reload data.frame
coordinates(meuse) = c("x", "y") # specify column names
class(meuse)
data(meuse) # reload data.frame
coordinates(meuse) = c(1, 2) # specify column names
class(meuse)
data(meuse) # reload data.frame
coordinates(meuse) = ~x+y # formula
class(meuse)
data(meuse) # reload data.frame
coordinates(meuse) = xy # as data frame
class(meuse)
data(meuse) # reload data.frame
coordinates(meuse) = as.matrix(xy) # as matrix
meuse$log.zn = log(meuse$zinc)
class(meuse)
dim(meuse)
SpatialPolygons

create objects of class SpatialPolygons or SpatialPolygonsDataFrame from lists of Polygons objects and data.frames

Usage

Polygon(coords, hole=as.logical(NA))
Polygons(srl, ID)
SpatialPolygons(srl, pO, proj4string=CRS(as.character(NA)))
SpatialPolygonsDataFrame(Sr, data, match.ID = TRUE)
getSpatialPolygonsLabelPoints(SP)

Arguments

coords 2-column numeric matrix with coordinates; first point (row) should equal last coordinates (row); if the hole argument is not given, the status of the polygon as a hole or an island will be taken from the ring direction, with clockwise meaning island, and counter-clockwise meaning hole

hole logical value for setting polygon as hole or not; if the hole argument is not given, the status of the polygon as a hole or an island will be taken from the ring direction, with clockwise meaning island, and counter-clockwise meaning hole

proj4string projection string of class CRS-class

srl list with Polygon-class objects

ID character vector of length one with identifier

Sr1 list with objects of class Polygons-class

pO integer vector; plotting order; if missing in reverse order of Polygons area

Sr object of class SpatialPolygons-class

data object of class data.frame; the number of rows in data should equal the number of Polygons-class objects in Sr

match.ID logical: (default TRUE): match SpatialPolygons member Polygons ID slot values with data frame row names, and re-order the data frame rows if necessary. If character: indicates the column in data with Polygons IDs to match

SP object of class SpatialPolygons-class
SpatialPolygons-class

Details

In Polygon, if the hole argument is not given, the status of the polygon as a hole or an island will be taken from the ring direction, with clockwise meaning island, and counter-clockwise meaning hole. In Polygons, if all of the member Polygon objects are holes, the largest by area will be converted to island status. Until 2010-04-17, version 0.9-61, the area of this converted object was erroneously left at its hole value of zero. Thanks to Patrick Giraudoux for spotting the bug.

The class definitions used for polygons in sp do not accord with those of the simple features specification of the Open Geospatial Consortium. The rgeos package, an interface to Geometry Engine – Open Source (GEOS), uses this specification, in which each hole (interior ring) must be associated with its containing exterior ring. In order to avoid introducing incompatible changes into the class definition of Polygons objects, a comment has been added as a single character string to each such object. Here we can trust the data source to assign the hole status correctly, and use the simple function createSPComment to add such comments to each Polygons member of the polygons slot of this SpatialPolygons object. Exterior rings are coded zero, while interior rings are coded with the 1-based index of the exterior ring to which they belong. SpatialPolygons objects created by reading using readOGR from rgdal have the comments set on input, as OGR also uses SFS.

Refer to Bivand et al. (2013), pages 47-48 and 132-133 for a further discussion.

Value

Polygon returns an object of class Polygon; Polygons returns an object of class Polygons; SpatialPolygons returns object of class SpatialPolygons; SpatialPolygonsDataFrame returns object of class SpatialPolygonsDataFrame getSpatialPolygonsLabelPoints returns an object of class SpatialPoints with label points.

References


See Also

SpatialPolygons-class, SpatialPolygonsDataFrame-class

SpatialPolygons-class

Class "SpatialPolygons"

Description

class to hold polygon topology (without attributes)

Objects from the Class

Objects can be created by calls to the function SpatialPolygons
Slots

- polygons: Object of class "list"; list elements are all of class \texttt{Polygons-class}
- plotOrder: Object of class "integer"; integer array giving the order in which objects should be plotted
- bbox: Object of class "matrix"; see \texttt{Spatial-class}
- proj4string: Object of class "CRS"; see \texttt{CRS-class}

Extends

Class "Spatial", directly.

Methods

Methods defined with class "SpatialPolygons" in the signature:

- \texttt{[ signature(obj = "SpatialPolygons")}: select subset of (sets of) polygons; NAs are not permitted in the row index
- \texttt{plot signature(x = "SpatialPolygons", y = "missing")}: plot polygons in SpatialPolygons object
- \texttt{summary signature(object = "SpatialPolygons")}: summarize object
- \texttt{rbind signature(object = "SpatialPolygons")}: rbind-like method

\textbf{plot method arguments}

The plot method for spatial polygons takes the following arguments:

- \texttt{x} a SpatialPolygons object
- \texttt{col} a vector of colour values
- \texttt{border} default \texttt{par("fg")}; the colour to draw the border
- \texttt{add} default \texttt{FALSE}; if \texttt{TRUE}, add to existing plot
- \texttt{xlim, ylim} default \texttt{NULL}; ranges for the plotted ‘x’ and ‘y’ values
- \texttt{xpd} default \texttt{NULL}; controls clipping, see \texttt{par}
- \texttt{density} default \texttt{NULL}; the density of shading lines, in lines per inch, see \texttt{polygon}
- \texttt{angle} default 45; the slope of shading lines, given as an angle in degrees (counter-clockwise), see \texttt{polygon}
- \texttt{pbg} default \texttt{NULL}, set to \texttt{par("bg")} by default “transparent”; the colour to paint holes
- \texttt{axes} default \texttt{FALSE}; draw axes
- \texttt{lty} default \texttt{par("lty")}; border line type
- ... other arguments passed through
- \texttt{setParUsrBB} default \texttt{FALSE}; see \texttt{Spatial-class} for further details
- \texttt{usePolypath} default \texttt{NULL} to set from option value; use \texttt{polypath} for hole-handling in plot
- \texttt{rule} default \texttt{NULL} to set from option value; character value specifying the path fill mode, see \texttt{polypath}
The options for usePolygonPath and rule may be retrieved with get_PolygonPath (default TRUE on package load) and get_PolygonPathRule (default “winding” on package load), and set with set_PolygonPath and set_PolygonPathRule.

The class definitions used for polygons in sp do not accord with those of the simple features specification of the Open Geospatial Consortium. The rgeos package, an interface to Geometry Engine – Open Source (GEOS), uses this specification, in which each hole (interior ring) must be associated with its containing exterior ring. In order to avoid introducing incompatible changes into the class definition of Polygons objects, a comment has been added as a single character string to each such object. Here we can trust the data source to assign the hole status correctly, and use the simple function `createSPComment` to add such comments to each Polygons member of the polygons slot of this SpatialPolygons object. Exterior rings are coded zero, while interior rings are coded with the 1-based index of the exterior ring to which they belong. SpatialPolygons objects created by reading using `readOGR` from rgdal have the comments set on input, as OGR also uses SFS.

Refer to Bivand et al. (2013), pages 47-48 and 132-133 for a further discussion.

Note

`rbind` calls the function `SpatialPolygons`, where it is checked that all IDs are unique. If `rbind`-ing `SpatialPolygons` without unique IDs, it is possible to set the argument `makeUniqueIDs = TRUE`, although it is preferred to change these explicitly with `spChFIDs`.

Author(s)

Roger Bivand

References


See Also

`SpatialPolygons`

Examples

```r
# simple example, from vignette("sp"):
Sr1 = Polygon(cbind(c(2,4,4,1,2),c(2,3,5,4,2)))
Sr2 = Polygon(cbind(c(5,4,2,5),c(2,3,2,2)))
Sr3 = Polygon(cbind(c(4,4,5,10,4),c(5,3,2,5,5)))
Sr4 = Polygon(cbind(c(5,6,6,5,5),c(4,4,3,3,4)), hole = TRUE)

Srs1 = Polygons(list(Sr1), "s1")
Srs2 = Polygons(list(Sr2), "s2")
Srs3 = Polygons(list(Sr3, Sr4), "s3/4")
SpP = SpatialPolygons(list(Srs1,Srs2,Srs3), 1:3)
plot(SpP, col = 1:3, pbg="white")

grd <- GridTopology(c(1,1), c(1,1), c(10,10))
polys <- as(grd, "SpatialPolygons")
```
plot(polys)
text(coordinates(polys), labels=row.names(polys))

---

SpatialPolygonsDataFrame-class

Class "SpatialPolygonsDataFrame"

Description

class to hold polygons with attributes

Objects from the Class

Objects can be created by calls to the function `SpatialPolygonsDataFrame`

Slots

data: Object of class "data.frame"; attribute table
polygons: Object of class "list"; see `SpatialPolygons-class`
plotOrder: Object of class "integer"; see `SpatialPolygons-class`
bbox: Object of class "matrix"; see `Spatial-class`
proj4string: Object of class "CRS"; see `CRS-class`

Extends

Class "SpatialPolygons", directly. Class "Spatial", by class "SpatialPolygons".

Methods

Methods defined with class "SpatialPolygonsDataFrame" in the signature:

- `[ signature(x = "SpatialPolygonsDataFrame")`: select subset of (sets of) polygons; NAs are not permitted in the row index
- `rbind` signature(object = "SpatialPolygonsDataFrame")`: rbind-like method, see notes below

Note

`SpatialPolygonsDataFrame` with default ID matching checks the data frame row names against the Polygons ID slots. They must then agree with each other, and be unique (no Polygons objects can share IDs); the data frame rows will be re-ordered if needed to match the Polygons IDs.

If you want to `rbind` objects with duplicated IDs, see `spChFIDs`.

Author(s)

Roger Bivand
See Also

SpatialPolygons-class

Examples

# simple example, from scratch:
Sr1 = Polygon(cbind(c(2,4,4,1,2),c(2,3,5,4,2)))
Sr2 = Polygon(cbind(c(5,4,2,5),c(2,3,2,2)))
Sr3 = Polygon(cbind(c(4,4,5,10,4),c(5,3,2,5,5)))
Sr4 = Polygon(cbind(c(5,6,6,5,5),c(4,4,3,3,4)), hole = TRUE)
Srs1 = Polygons(list(Sr1), "s1")
Srs2 = Polygons(list(Sr2), "s2")
Srs3 = Polygons(list(Sr3, Sr4), "s3/4")
SpP = SpatialPolygons(list(Srs1,Srs2,Srs3), 1:3)
plot(SpP, col = 1:3, pbg="white")

grd <- GridTopology(c(1,1), c(1,1), c(10,10))
polys <- as(grd, "SpatialPolygons")
centroids <- coordinates(polys)
x <- centroids[,1]
y <- centroids[,2]
z <- 1.4 + 0.1*x + 0.2*y + 0.002*x*x
ex_1.7 <- SpatialPolygonsDataFrame(polys,
data=data.frame(x=x, y=y, z=z, row.names=row.names(polys)))
brks <- quantile(z, seq(0,1,1/7))
cols <- grey((length(brks):2)/length(brks))
dens <- (2:length(brks))*3
plot(ex_1.7, col=cols[findInterval(z, brks, all.inside=TRUE)])
plot(ex_1.7, density=dens[findInterval(z, brks, all.inside=TRUE)])

spChFIDs-methods

change feature IDs in spatial objects

Description

When the feature IDs need to be changed in SpatialLines* or SpatialPolygons* objects, these methods may be used. The new IDs should be a character vector of unique IDs of the correct length.

Methods

obj = "SpatialLines", x = "character" replace IDs in a SpatialLines object

obj = "SpatialLinesDataFrame", x = "character" replace IDs in a SpatialLinesDataFrame object

obj = "SpatialPolygons", x = "character" replace IDs in a SpatialPolygons object

obj = "SpatialPolygonsDataFrame", x = "character" replace IDs in a SpatialPolygonsDataFrame object
Note
It is usually sensible to keep a copy of the original feature IDs in the object, but this should be done by the user.

Author(s)
Roger Bivand

See Also
spCbind-methods, spRbind-methods

Examples
## Not run:
require(maptools)
xx <- readShapePoly(system.file("shapes/sids.shp", package="maptools")[[1]],
  IDvar="FIPSNO", proj4string=CRS("+proj=longlat +ellps=clrk66"))
row.names(as(xx, "data.frame"))
xx1 <- spChFIDs(xx, as.character(xx$CNTY_ID))
row.names(as(xx1, "data.frame"))
## End(Not run)

spDistsN1

Euclidean or Great Circle distance between points

Description
The function returns a vector of distances between a matrix of 2D points, first column longitude, second column latitude, and a single 2D point, using Euclidean or Great Circle distance (WGS84 ellipsoid) methods.

Usage
spDistsN1(pts, pt, longlat = FALSE)
spDists(x, y = x, longlat = FALSE, segments = FALSE, diagonal = FALSE)

Arguments
pts
  A matrix of 2D points, first column x/longitude, second column y/latitude, or a SpatialPoints or SpatialPointsDataFrame object
pt
  A single 2D point, first value x/longitude, second value y/latitude, or a SpatialPoints or SpatialPointsDataFrame object with one point only
x
  A matrix of n-D points with row denoting points, first column x/longitude, second column y/latitude, or a Spatial object that has a coordinates method
spDistsN1

y  A matrix of n-D points with row denoting points, first column x/longitude, second column y/latitude, or a Spatial object that has a coordinates method
longlat  logical; if FALSE, Euclidean distance, if TRUE Great Circle (WGS84 ellipsoid) distance; if x is a Spatial object, longlat should not be specified but will be derived from is.projected(x)
segments  logical; if TRUE, y must be missing; the vector of distances between consecutive points in x is returned.
diagonal  logical; if TRUE, y must be given and have the same number of points as x; the vector with distances between points with identical index is returned.

Value

spDistsN1 returns a numeric vector of distances in the metric of the points if longlat=FALSE, or in kilometers if longlat=TRUE.
spDists returns a full matrix of distances in the metric of the points if longlat=FALSE, or in kilometers if longlat=TRUE; it uses spDistsN1 in case points are two-dimensional. In case of spDists(x,x), it will compute all n x n distances, not the sufficient n x (n-1).

Note

The function can also be used to find a local kilometer equivalent to a plot scaled in decimal degrees in order to draw a scale bar.

Author(s)

Roger Bivand, Edzer Pebesma

References

http://www.abecedarical.com/javascript/script_greatcircle.html

See Also

is.projected

Examples

ll <- matrix(c(5, 6, 60, 60), ncol=2)
km <- spDistsN1(ll, ll[,1], longlat=TRUE)
zapsmall(km)
utm32 <- matrix(c(276.9799, 332.7052, 6658.1572, 6655.2055), ncol=2)
spDistsN1(utm32, utm32[,1])
dg <- spDistsN1(ll, ll[1,])
dg
dg[2]/km[2]
data(meuse)
coordinates(meuse) <- c("x", "y")
res <- spDistsN1(meuse, meuse[,1])
summary(res)
**spplot**  
Plot methods for spatial data with attributes

**Description**
Lattice (trellis) plot methods for spatial data with attributes

**Usage**

```r
spplot(obj, ...)  
spplot.grid(obj, zcol = names(obj), ..., names.attr, scales = list(draw = FALSE),  
            xlab = NULL, ylab = NULL, aspect = mapasp(obj, xlim, ylim),
```
spplot

panel = panel.gridplot, sp.layout = NULL, formula, xlim = bbox(obj)[1], ylim = bbox(obj)[2], checkEmptyRC = TRUE, col.regions = get_col_regions()
spplot.polygons(obj, zcol = names(obj), ..., names.attr, scales = list(draw = FALSE), xlab = NULL, ylab = NULL, aspect = mapasp(obj,xlim,ylim),
panel = panel.polygonsplot, sp.layout = NULL, formula, xlim = bbox(obj)[1], ylim = bbox(obj)[2], col.regions = get_col_regions()
spplot.points(obj, zcol = names(obj), ..., names.attr, scales = list(draw = FALSE),
panel = panel.pointsplot, sp.layout = NULL, formula, xlab = NULL, ylab = NULL, aspect = mapasp(obj,xlim,ylim),
edge.col = "transparent", colorkey = FALSE, col.regions = get_col_regions())
mapLegendGrob(obj, widths = unit(1, "cm"), heights = unit(1, "cm"), fill = "black", just = "right")
sp.theme(set = FALSE, regions = list(col = bpy.colors(100)), ...)
layout.north.arrow(type = 1)
layout.scale.bar(height = 0.05)
spplot.locator(n = 512, type = "n", ...)
set_col_regions(value)
get_col_regions()

Arguments

obj object of class extending Spatial-class
zcol character; attribute name(s) or column number(s) in attribute table
names.attr names to use in panel, if different from zcol names
scales scales argument to be passed to Lattice plots; use list(draw = TRUE) to draw axes scales; see xyplot for full options...
... other arguments passed to levelplot (grids, polygons) or xypoint (points)
xlab label for x-axis
ylab label for y-axis
aspect aspect ratio for spatial axes; defaults to "iso" (one unit on the x-axis equals one unit on the y-axis) but may be set to more suitable values if the data are e.g. if coordinates are latitude/longitude
panel depending on the class of obj, panel.polygonsplot (for polygons or lines), panel.gridplot (grids) or panel.pointsplot (points) is used; for further control custom panel functions can be supplied that call one of these panel functions, but do read below how the argument sp.layout may help
sp.layout NULL or list; see notes below
identify if not FALSE, identify plotted objects (currently only working for points plots). Labels for identification are the row.names of the attribute table row.names(as.data.frame(obj)). If TRUE, identify on panel (1,1); for identifying on panel i,j, pass the value c(i,j)
formula optional; may be useful to plot a transformed value. Defaults to z~x+y for single and z~x+y|name for multiple attributes; use e.g. exp(x)~x+y|name to plot the exponent of the z-variable
spplot

**Arguments**

- `xlim` numeric; x-axis limits
- `ylim` numeric; y-axis limits
- `edge.col` color of symbol edge
- `colorkey` if FALSE, use symbol key; if TRUE, use continuous, levelplot-like colorkey; if list, follow syntax of argument `colorkey` in `levelplot` (see below for an example)
- `widths` width of grob
- `heights` heights of grob
- `fill` fill color of grob
- `just` grob placement justification
- `set` logical; if TRUE, `trellis.par.set` is called, else a list is returned that can be passed to `trellis.par.set()`
- `regions` color ramp for the theme
- `height` height of scale bar; width is 1.0
- `n` see locator
- `type` see locator
- `checkEmptyRC` logical; if TRUE, a check is done to see if empty rows or columns are present, and need to be taken care of. Setting to FALSE may improve speed.
- `col.regions` vector with fill colours; in case the variable to be plotted is a factor, this vector should have length equal to the number of factor levels
- `value` vector with color values, default for `col.regions`

**Value**

`spplot` returns a lattice plot of class "trellis", if you fail to "see" it, explicitly call `print(spplot(...))`. If `identify` is TRUE, the plot is plotted and the return value is a vector with row names of the selected points.

`spplot.locator` returns a matrix with identified point locations; use `trellis.focus` first to focus on a given panel.

`get_col_regions` returns the default value for `col.regions`

**Methods**

- `obj = "SpatialPixelsDataFrame"` see `spplot`
- `obj = "SpatialGridDataFrame"` see `spplot`
- `obj = "SpatialPolygonsDataFrame"` see `spplot`
- `obj = "SpatialLinesDataFrame"` see `spplot`
- `obj = "SpatialPointsDataFrame"` see `spplot`
Note

Missing values in the attributes are (currently) not allowed.

spplot.grid, spplot.polygons and spplot.points are S4 methods for spplot; see spplot-methods.

Useful arguments that can be passed as ... are:

layout integer; for the layout of panels (cols,rows)
pretty logical; choose colour breaks at pretty numbers?
at specify at which values colours change
as.table logical; start drawing panels upper-left instead of lower-left
page to add marks to each plotted page

for useful values see the appropriate documentation of xyplot (in case of points), and levelplot (otherwise).

If obj is of SpatialPointsDataFrame, the following options are useful to pass:

data.cuts number of cuts, or, for objects of class SpatialPointsDataFrame only, the actual cuts to use
do.log logical; if TRUE use log-linear scale to divide range in equal cuts, else use a linear scale

pch integer; plotting character to use; defaults to 16 if fill is TRUE, else 1
cex numeric; character expansion, proportional to default value of 1
fill logical; use filled circles?

layout.north.arrow and layout.scale.bar can be used to set a north arrow or scale bar.

The sp.layout argument is either a single layout item, or a list with one or more layout items. A layout item is one of

- a list with one or more Spatial* objects, along with style arguments like col, lty, pch, fill etc.
- a list with its first argument the layout function or the name of the layout function to be called: sp.points for SpatialPoints, sp.polygons for SpatialPolygons object, sp.lines for a SpatialLines object, and sp.text for text to place. The second argument contains the object (or text) to be plotted; remaining arguments are passed to the corresponding panel.* functions.

The order of items in sp.layout matters; objects are drawn in the order they appear. With respect to obj, default plot order and precedence of sp.layout items is as follows: for points and lines, sp.layout items are drawn over (after) obj; for grids and polygons, sp.layout items are drawn behind (before) obj. Transparency may further help making multiple things visible. Adding a first argument to a layout item overrides its default plotting order with respect to obj:

Special control elements of sp.layout items:

first logical; should the layout item be drawn before the obj (TRUE), or after (FALSE)? This overrides the default order (points and lines in front, polygons and grids behind).
which integer; controls to which panel a layout item should be added. If which is present in the main, top-level list it applies to all layout items; in sub-lists with layout items it denotes the (set of) panel(s) in which the layout item should be drawn. Without a which item, layout items are drawn in each panel.

sp.theme returns a lattice theme; use, after loading package lattice, the command trellis.par.set(sp.theme()) after a device is opened or changed to make this work. Currently, this only sets the colors to bpy.colors.

If the attributes to be plotted are of type factor, spplot tries to create a legend that reflects this. In this case, the color ramp passed needs to be of the same length as the number of factor levels. The factor levels are derived from the first map; subsequent factors with different factor levels result in an error.

Author(s)
Edzer Pebesma, <edzer.pebesma@uni-muenster.de>

References
https://edzer.github.io/sp/

See Also
xyplot, levelplot, panel.identify to identify objects

Examples
library(lattice)
trellis.par.set(sp.theme()) # sets bpy.colors() ramp
demo(meuse, ask = FALSE, echo = FALSE)
l2 = list("SpatialPolygonsRescale", layout.north.arrow(), offset = c(181300,329800), scale = 400)
l3 = list("SpatialPolygonsRescale", layout.scale.bar(), offset = c(180500,329800), scale = 500, fill=c("transparent", "black"))
l4 = list("sp.text", c(180500,329900), "0")
l5 = list("sp.text", c(181000,329900), "500 m")

spplot(meuse, c("ffreq"), sp.layout=list(l2,l3,l4,l5), col.regions= "black", pch=c(1,2,3), key.space=list(x=0.1,y=.95,corner=c(0,1)))
spplot(meuse, c("zinc", "lead"), sp.layout=list(l2,l3,l4,l5, which = 2), key.space=list(x=0.1,y=.95,corner=c(0,1)))

# plotting factors:
meuse$f = factor(sample(letters[6:10], 155, replace=TRUE),levels=letters[1:10])
meuse$g = factor(sample(letters[1:5], 155, replace=TRUE),levels=letters[1:10])
spplot(meuse, c("ffreq", "g"), col.regions=bpy.colors(10))

if (require(RColorBrewer)) {
  spplot(meuse, c("ffreq"), sp.layout=list(l2,l3,l4,l5), col.regions=brewer.pal(3, "Set1"))
}
meuse.grid$g = factor(sample(letters[1:5], 3103, replace=TRUE),
levels=letters[1:10])
meuse.grid$f = factor(sample(letters[6:10], 3103, replace=TRUE),
levels=letters[1:10])
spplot(meuse.grid, c("f","g"), col.regions=bpy.colors(10))

# example modifying colorkey for points:
spplot(meuse["dist"], colorkey = list(
  right = list( # see ?levelplot in package trellis, argument colorkey:
    fun = draw.colorkey,
    args = list(
      key = list(
        at = seq(0, 1, .1), # colour breaks
        col = bpy.colors(11), # colours
        labels = list(
          at = c(0, .2, .4, .6, .8, 1),
          labels = c("0x", "20x", "40x", "60x", "80x", "100x")
        )
      )
    )
  )
))
l6 = list(meuse.grid["dist"], col = grey(seq(.5,.9,length.out=10)))
spplot(meuse, c("zinc", "lead"), sp.layout = l6)
spplot(meuse, c("zinc", "lead"),
sp.layout = list(meuse.grid, meuse.riv, col = 'grey'))

# Custom legend placement, taken from
s <- spplot(meuse.grid[,"dist"], colorkey = list(space = "left", height = 0.4))

args <- s$legend$left$args$key

## Prepare list of arguments needed by "legend=" argument (as described in ?xyplot)
library(lattice) # draw.colorkey
legendArgs <- list(fun = draw.colorkey,
  args = list(key = args),
  corner = c(0.05,.75))

## Call spplot() again, this time passing in to legend the arguments
## needed to print a color key
spplot(meuse.grid[,"dist"], colorkey = FALSE,
  legend = list(inside = legendArgs))

sp.sample = sample point locations in (or on) a spatial object

Sample point locations within a square area, a grid, a polygon, or on a spatial line, using regular
or random sampling methods; the methods used assume that the geometry used is not spherical, so
objects should be in planar coordinates

Usage

spsample(x, n, type, ...)  
makegrid(x, n = 10000, nsig = 2, cellsize, offset = rep(0.5, nrow(bb)),  
pretty = TRUE)

Arguments

x     Spatial object; spsample(x,...) is a generic method for the existing sample.Xxx functions
...
     optional arguments, passed to the appropriate sample.Xxx functions; see NOTES for nclusters and iter
n     (approximate) sample size
type  character; "random" for completely spatial random; "regular" for regular (systematically aligned) sampling; "stratified" for stratified random (one single random location in each "cell"); "nonaligned" for nonaligned systematic sampling (nx random y coordinates, ny random x coordinates); "hexagonal" for sampling on a hexagonal lattice; "clustered" for clustered sampling; "Fibonacci" for Fibonacci sampling on the sphere (see references).
bb    bounding box of the sampled domain; setting this to a smaller value leads to sub-region sampling
offset for square cell-based sampling types (regular, stratified, nonaligned, hexagonal): the offset (position) of the regular grid; the default for spsample methods is a random location in the unit cell [0,1] x [0,1], leading to a different grid after each call; if this is set to c(0.5, 0.5), the returned grid is not random (but, in Ripley's wording, "centric systematic"). For line objects, a single offset value is taken, where the value varies within the [0, 1] interval, and 0 is the beginning of each Line object, and 1 its end

cellsive  if missing, a cell size is derived from the sample size n; otherwise, this cell size is used for all sampling methods except "random"
nsig    for "pretty" cell size; spsample does not result in pretty grids
pretty  logical; if TRUE, choose pretty (rounded) coordinates

Value

an object of class SpatialPoints-class. The number of points is only guaranteed to equal n when sampling is done in a square box, i.e. (sample.Spatial). Otherwise, the obtained number of points will have expected value n.

When x is of a class deriving from Spatial-class for which no spsample-methods exists, sampling is done in the bounding box of the object, using spsample.Spatial. An overlay using over may be necessary to select the features inside the geometry afterwards.

Sampling type "nonaligned" is not implemented for line objects.

Some methods may return NULL if no points could be successfully placed.
makegrid makes a regular grid that covers \( x \); when \texttt{cellsize} is not given it derives one from the number of grid points requested (approximating the number of cells). It tries to choose pretty cell size and grid coordinates.

**Methods**

\( x = \) \{ "Spatial" \} sample in the bbox of \( x \)

\( x = \) \{ "Line" \} sample on a line

\( x = \) \{ "Polygon" \} sample in a Polygon

\( x = \) \{ "Polygons" \} sample in a Polygons object, consisting of possibly multiple Polygon objects (holes must be correctly defined, use \texttt{checkPolygonsHoles} if need be)

\( x = \) \{ "SpatialPolygons" \} sample in an SpatialPolygons object; sampling takes place over all Polygons objects present, use subsetting to vary sampling intensity (density); holes must be correctly defined, use \texttt{checkPolygonsHoles} if need be

\( x = \) \{ "SpatialGrid" \} sample in an SpatialGrid object

\( x = \) \{ "SpatialPixels" \} sample in an SpatialPixels object

**Note**

If an \texttt{Polygon-class} object has zero area (i.e. is a line), samples on this line element are returned. If the area is very close to zero, the algorithm taken here (generating points in a square area, selecting those inside the polygon) may be very resource intensive. When numbers of points per polygon are small and \texttt{type="random"}, the number searched for is inflated to ensure hits, and the points returned sampled among these.

The following two arguments can be further specified:

\( \texttt{nclusters} \) Number of clusters (strata) to sample from.

\( \texttt{iter} \) (default = 4) number of times to try to place sample points in a polygon before giving up and returning \texttt{NULL} - this may occur when trying to hit a small and awkwardly shaped polygon in a large bounding box with a small number of points

**Author(s)**

Edzer Pebesma, <edzer.pebesma@uni-muenster.de>

**References**

Chapter 3 in B.D. Ripley, 1981. Spatial Statistics, Wiley


**See Also**

\texttt{over}, \texttt{point.in.polygon}, \texttt{sample}
Examples

data(meuse.riv)
meuse.sr = SpatialPolygons(list(Polygons(list(Polygon(meuse.riv)), "x")))

plot(meuse.sr)
points(spsample(meuse.sr, n = 1000, "regular"), pch = 3)

plot(meuse.sr)
points(spsample(meuse.sr, n = 1000, "random"), pch = 3)

plot(meuse.sr)
points(spsample(meuse.sr, n = 1000, "stratified"), pch = 3)

plot(meuse.sr)
points(spsample(meuse.sr, n = 1000, "nonaligned"), pch = 3)

plot(meuse.sr)
points(spsample(meuse.sr@polygons[[1]], n = 100, "stratified"), pch = 3, cex=.5)

data(meuse.grid)
gridded(meuse.grid) = ~x+y
image(meuse.grid)
points(spsample(meuse.grid,n=1000,type="random"), pch=3, cex=.5)
image(meuse.grid)
points(spsample(meuse.grid,n=1000,type="stratified"), pch=3, cex=.5)
image(meuse.grid)
points(spsample(meuse.grid,n=1000,type="regular"), pch=3, cex=.5)
image(meuse.grid)
points(spsample(meuse.grid,n=1000,type="nonaligned"), pch=3, cex=.5)
!fullgrid(meuse.grid) = TRUE
image(meuse.grid)
points(spsample(meuse.grid,n=1000,type="stratified"), pch=3,cex=.5)

---

spTransform

spTransform for map projection and datum transformation

Description

spTransform for map projection and datum transformation

Usage

spTransform(x, CRSobj, ...)

Arguments

- **x**: object to be transformed
- **CRSobj**: object of class **CRS**, or of class **character** in which case it is converted to **CRS**
- ... further arguments (ignored)

Value

object with coordinates transformed to the new coordinate reference system.

Note

Package **rgdal** provides the methods doing actual transformation, see **spTransform**; when **rgdal** cannot be loaded, an error message follows.

---

**stack**

*rearrange data in SpatialPointsDataFrame or SpatialGridDataFrame for plotting with spplot (levelplot/xyplot wrapper)*

Description

rearrange SpatialPointsDataFrame for plotting with spplot or levelplot

Usage

```r
spmap.to.lev(data, zcol = 1:n, n = 2, names.attr)
## S3 method for class 'SpatialPointsDataFrame'
stack(x, select, ...)
## S3 method for class 'SpatialGridDataFrame'
stack(x, select, ...)
```

Arguments

- **data**: object of class (or extending) SpatialPointsDataFrame or SpatialGridDataFrame
- **zcol**: z-coordinate column name(s), or a column number (range) (after removing the spatial coordinate columns: 1 refers to the first non-coordinate column, etc.)
- **names.attr**: names of the set of z-columns (these names will appear in the plot); if omitted, column names of zcol
- **n**: number of columns to be stacked
- **x**: same as data
- **select**: same as zcol
- ... ignored
**surfaceArea**

*Compute surface area of a digital elevation model.*

**Value**

spmap.to.lev returns a data frame with the following elements:

- **x**: x-coordinate for each row
- **y**: y-coordinate for each row
- **z**: column vector with each of the elements in columns zcol of data stacked
- **name**: factor; name of each of the stacked z columns

**Stack** is an S3 method: it returns a data.frame with a column **values** that has the stacked coordinates and attributes, and a column **ind** that indicates the variable stacked; it also replicates the coordinates.

**See Also**

spplot, levelplot in package lattice, and stack

**Examples**

```r
library(lattice)
data(meuse.grid) # data frame
coordinates(meuse.grid) = c("x", "y") # promotes to SpatialPointsDataFrame
meuse.grid[["idist"]]= 1 - meuse.grid[["dist"]]) # add variable
# the following is made much easier by spplot:
levelplot(z~x+y|name, spmap.to.lev(meuse.grid, z=c("dist","idist"), names.attr =
c("distance", "inverse of distance")), aspect = "iso")
levelplot(values~x+y|ind, as.data.frame(stack(meuse.grid)),aspect = "iso")
gridded(meuse.grid) = TRUE
levelplot(z~x+y|name, spmap.to.lev(meuse.grid, z=c("dist","idist"), names.attr =
c("distance", "inverse of distance")), aspect = "iso")
levelplot(values~x+y|ind, as.data.frame(stack(meuse.grid)), asp = "iso")
```

**Description**

It is often said that if Wales was flattened out it would have an area bigger than England. This function computes the surface area of a grid of heights taking into account the sloping nature of the surface.

**Usage**

```r
surfaceArea(m, ...) 
surfaceArea.matrix(m, cellx = 1, celly = 1, byCell = FALSE)
```
Arguments

m a matrix of height values, or an object of class SpatialPixelsDataFrame or SpatialGridDataFrame.
cellx the size of the grid cells in the x-direction, in the same units as the height values.
celly the size of the grid cells in the y-direction, in the same units as the height values.
byCell return single value or matrix of values
... ignored

Value

Either a single value of the total area if byCell=FALSE, or a matrix the same shape as m of individual cell surface areas if byCell=TRUE. In this case, the sum of the returned matrix should be the same value as that which is returned if byCell=FALSE.

Missing values (NA) in the input matrix are allowed. They will produce an NA in the output matrix for byCell=TRUE, and contribute zero to the total area. They also have an effect on adjacent cells - see code comments for details.

Methods

obj = "matrix" takes a matrix as input, requires cellx and celly to be set
obj = "SpatialGridDataFrame" takes an object of class SpatialGridDataFrame as input, and retrieves cellx and celly from this
obj = "SpatialPixelsDataFrame" takes an object of class SpatialPixelsDataFrame as input, and retrieves cellx and celly from this

Author(s)

Barry Rowlingson <b.rowlingson@lancaster.ac.uk>, integration in sp Edzer Pebesma.

References


Examples

surfaceArea(volcano)
image(surfaceArea(volcano, byCell=TRUE))

data(meuse.grid)
gridded(meuse.grid) = ~x+y
image(surfaceArea(meuse.grid["dist"], byCell=TRUE))
surfaceArea(meuse.grid["dist"])


zerodist

**find point pairs with equal spatial coordinates**

**Description**
find point pairs with equal spatial coordinates

**Usage**
zerodist(obj, zero = 0.0, unique.ID = FALSE, memcmp = TRUE)
zerodist2(obj1, obj2, zero = 0.0, memcmp = TRUE)
remove.duplicates(obj, zero = 0.0, remove.second = TRUE, memcmp = TRUE)

**Arguments**

- **obj** object of, or extending, class `SpatialPoints`
- **obj1** object of, or extending, class `SpatialPoints`
- **obj2** object of, or extending, class `SpatialPoints`
- **zero** distance values less than or equal to this threshold value are considered to have zero distance (default 0.0); units are those of the coordinates for projected data or unknown projection, or km if coordinates are defined to be longitude/latitude
- **unique.ID** logical; if TRUE, return an ID (integer) for each point that is different only when two points do not share the same location
- **memcmp** use memcmp to find exactly equal coordinates; see NOTE
- **remove.second** logical; if TRUE, the second of each pair of duplicate points is removed, if FALSE remove the first

**Value**
zerodist and zerodist2 return a two-column matrix with in each row pairs of row numbers with identical coordinates; a matrix with zero rows is returned if no such pairs are found. For zerodist, row number pairs refer to row pairs in obj. For zerodist2, row number pairs refer to rows in obj and obj2, respectively. remove.duplicates removes duplicate observations if present, and else returns obj.

**Note**
When using kriging, duplicate observations sharing identical spatial locations result in singular covariance matrices. This function may help identify and remove spatial duplicates. The full matrix with all pair-wise distances is not stored; the double loop is done at the C level.

When unique.ID=TRUE is used, an integer index is returned. sp 1.0-14 returned the highest index, sp 1.0-15 and later return the lowest index.

When zero is 0.0 and memcmp is not FALSE, zerodist uses memcmp to evaluate exact equality of coordinates; there may be cases where this results in a different evaluation compared to doing the double arithmetic of computing distances.
Examples

data(meuse)
summary(meuse)
# pick 10 rows
n <- 10
ran10 <- sample(nrow(meuse), size = n, replace = TRUE)
meusedup <- rbind(meuse, meuse[ran10, ])
coordinates(meusedup) <- c("x", "y")
zd <- zerodist(meusedup)
sum(abs(zd[1:n,1] - sort(ran10))) # 0!
# remove the duplicate rows:
meusedup2 <- meusedup[-zd[,2], ]
summary(meusedup2)
meusedup3 <- subset(meusedup, !(1:nrow(meusedup) %in% zd[,2]))
summary(meusedup3)
coordinates(meuse) <- c("x", "y")
zerodist2(meuse, meuse[c(10:33,1,10),])
## Index

### classes
- CRS-class, 18
- DMS-class, 24
- GridTopology-class, 31
- Line, 37
- Line-class, 38
- Lines-class, 39
- Polygon-class, 52
- Polygons-class, 53
- Spatial-class, 61
- SpatialGrid-class, 63
- SpatialGridDataFrame-class, 64
- SpatialLines-class, 68
- SpatialLinesDataFrame-class, 70
- SpatialMultiPoints-class, 72
- SpatialMultiPointsDataFrame-class, 74
- SpatialPixels-class, 78
- SpatialPixelsDataFrame-class, 80
- SpatialPoints-class, 83
- SpatialPointsDataFrame-class, 85
- SpatialPolygons-class, 88
- SpatialPolygonsDataFrame-class, 91

### color
- bpy.colors, 11

### datasets
- meuse, 42
- meuse.grid, 43
- meuse.grid_11, 44
- meuse.riv, 45
- Rlogo, 57

### dplot
- bubble, 12
- degAxis, 21
- flip, 25
- loadMeuse, 39
- mapasp, 40
- panel.sSplot, 49
- sSplot, 95
- stack, 104
- zerodist, 107

### manip
- coordinates, 16
- gridIndex2nb, 28
- point.in.polygon, 51
- polygons, 53
- sp-deprecated, 60
- SpatialLines, 67
- SpatialMultiPoints, 71
- SpatialPoints, 82
- SpatialPolygons, 87
- spsample, 100

### methods
- addAttrToGeom-methods, 4
- aggregate, 5
- bbox-methods, 10
- coordinates-methods, 17
- coordnames-methods, 18
- dimensions-methods, 22
- disaggregate-methods, 23
- geometry-methods, 26
- gridded-methods, 27
- merge, 41
- over-methods, 46
- polygons-methods, 54
- recenter-methods, 56
- spChFIDs-methods, 92
- spsample, 100
- spTransform, 103

### misc
- compassRose, 15

### models
- select.spatial, 58

### programming
- read.asciigrid, 55

### spatial
- as.SpatialPolygons.GridTopology, 7
- as.SpatialPolygons.PolygonsList, 8
bbox-methods, 10
char2dms, 14
CRS-class, 18
DMS-class, 24
gridded-methods, 27
gridlines, 29
image.SpatialGridDataFrame, 33
is.projected, 35
merge, 41
polygons-methods, 54
sp, 59
SpatialPixels, 75
SpatialPixelsDataFrame, 79
spChFIDs-methods, 92
spDistsN1, 93
spTransform, 103
surfaceArea, 105
[,SpatialGrid-method
  (SpatialGrid-class), 63
[,SpatialGridDataFrame-method
  (SpatialGridDataFrame-class), 64
[,SpatialLines-method
  (SpatialLines-class), 68
[,SpatialLinesDataFrame-method
  (SpatialLinesDataFrame-class), 70
[,SpatialMultiPoints-method
  (SpatialMultiPoints-class), 72
[,SpatialMultiPointsDataFrame-method
  (SpatialMultiPointsDataFrame-class), 74
[,SpatialPixels-method
  (SpatialPixels-class), 78
[,SpatialPixelsDataFrame-method
  (SpatialPixelsDataFrame-class), 80
[,SpatialPoints-method
  (SpatialPoints-class), 83
[,SpatialPointsDataFrame-method
  (SpatialPointsDataFrame-class), 85
[,SpatialPolygons-method
  (SpatialPolygons-class), 88
[,SpatialPolygonsDataFrame-method
  (SpatialPolygonsDataFrame-class), 91
<-,Spatial-method (Spatial-class), 61
<-,Spatial,ANY,missing-method
  (Spatial-class), 61
$.,Spatial-method
  (Spatial-class), 61
$,SpatialMultiPoints-method
  (SpatialMultiPoints-class), 72
$,SpatialPoints-method
  (SpatialPoints-class), 83
$<-,Spatial-method
  (Spatial-class), 61
$<-,SpatialMultiPoints-method
  (SpatialMultiPoints-class), 72
$<-,SpatialPoints-method
  (SpatialPoints-class), 83
%over% (over-methods), 46
addAttrToGeom (addAttrToGeom-methods), 4
addAttrToGeom, SpatialGrid, data.frame-method
  (addAttrToGeom-methods), 4
addAttrToGeom, SpatialLines, data.frame-method
  (addAttrToGeom-methods), 4
addAttrToGeom, SpatialMultiPoints, data.frame-method
  (addAttrToGeom-methods), 4
addAttrToGeom, SpatialPixels, data.frame-method
  (addAttrToGeom-methods), 4
addAttrToGeom, SpatialPoints, data.frame-method
  (addAttrToGeom-methods), 4
addAttrToGeom-methods, 4
aggregate, 5
aggregate.data.frame, 5
areaSpatialGrid (SpatialPixels), 75
as, 74, 85
as.array.SpatialGridDataFrame
  (SpatialGridDataFrame-class), 64
as.character.DMS (char2dms), 14
as.data.frame.SpatialGrid
  (SpatialGrid-class), 63
as.data.frame.SpatialGridDataFrame
  (SpatialGridDataFrame-class), 64
as.data.frame.SpatialMultiPoints
  (SpatialMultiPoints-class), 72
as.data.frame.SpatialMultiPointsDataFrame
  (SpatialMultiPointsDataFrame-class), 74
as.data.frame.SpatialPixels (SpatialPixels-class), 78
as.data.frame.SpatialPixelsDataFrame (SpatialPixelsDataFrame-class), 80
as.data.frame.SpatialPoints (SpatialPoints-class), 83
as.data.frame.SpatialPointsDataFrame (SpatialPointsDataFrame-class), 85
as.data.frame.SpatialPolygons (SpatialPolygons-class), 88
as.data.frame.SpatialPolygonsDataFrame (SpatialPolygonsDataFrame-class), 91
as.double.DMS (DMS-class), 24
as.image.SpatialGridDataFrame, 55
axTicks, 22, 40
bbox (bbox-methods), 10
bbox,ANY-method (bbox-methods), 10
bbox,Line-method (bbox-methods), 10
bbox,Lines-method (bbox-methods), 10
bbox,Polygon-method (bbox-methods), 10
bbox,Polygons-method (bbox-methods), 10
bbox,Spatial-method (bbox-methods), 10
bbox-methods, 10
bpy.colors, 11, 99
bubble, 12
cbind.Spatial (Spatial-class), 61
cbind.SpatialGridDataFrame (SpatialGridDataFrame-class), 64
char2dms, 14, 25
cm.colors, 12
coerce,deldir,SpatialLines-method (SpatialLines-class), 68
coerce,deldir,SpatialPolygons-method (SpatialPolygons-class), 88
coerce,DMS,character-method (char2dms), 14
coerce,DMS,numeric-method (char2dms), 14
coerce,DMS-method (DMS-class), 24
coerce,GridTopology,data.frame-method (GridTopology-class), 31
coerce,GridTopology,SpatialPolygons-method (as.SpatialPolygons.GridTopology), 7
coerce,im, SpatialGridDataFrame-method (SpatialGridDataFrame-class), 64
coerce,Lines, SpatialMultiPoints-method (SpatialMultiPoints-class), 68
coerce,Lines, SpatialPoints-method (SpatialPoints-class), 68
coerce,Polygons,Lines-method (SpatialPolygons-class), 88
coerce,ppp, SpatialGridDataFrame-method (SpatialGridDataFrame-class), 64
coerce,ppp, SpatialPoints-method (SpatialPoints-class), 83
coerce,ppp, SpatialPointsDataFrame-method (SpatialPointsDataFrame-class), 85
coerce,SpatialGrid, data.frame-method (SpatialGrid-class), 63
coerce,SpatialGrid, GridTopology-method (GridTopology-class), 31
coerce,SpatialGrid, SpatialPixels-method (SpatialGrid-class), 63
coerce,SpatialGrid, SpatialPoints-method (SpatialGrid-class), 63
coerce,SpatialGrid, SpatialPolygons-method (SpatialGrid-class), 63
coerce,SpatialGridDataFrame, array-method (SpatialGridDataFrame-class), 64
coerce, SpatialGridDataFrame, data.frame-method
  (SpatialGridDataFrame-class), 64
coerce, SpatialGridDataFrame, matrix-method
  (SpatialGridDataFrame-class), 64
coerce, SpatialGridDataFrame, SpatialPixelsDataFrame-method
  (SpatialGridDataFrame-class), 64
coerce, SpatialGridDataFrame, SpatialPointsDataFrame-method
  (SpatialGridDataFrame-class), 64
coerce, SpatialGridDataFrame, SpatialPolygonsDataFrame-method
  (SpatialGridDataFrame-class), 64
coerce, SpatialLines, SpatialMultiPoints-method
  (SpatialLines-class), 68
coerce, SpatialLines, SpatialPoints-method
  (SpatialLines-class), 68
coerce, SpatialLines, SpatialPointsDataFrame-method
  (SpatialLines-class), 68
coerce, SpatialLinesDataFrame, data.frame-method
  (SpatialLinesDataFrame-class), 70
coerce, SpatialLinesDataFrame, SpatialMultiPointsDataFrame-method
  (SpatialLinesDataFrame-class), 70
coerce, SpatialLinesDataFrame, SpatialPointsDataFrame-method
  (SpatialLinesDataFrame-class), 70
coerce, SpatialMultiPoints, data.frame-method
  (SpatialMultiPoints-class), 72
coerce, SpatialMultiPoints, matrix-method
  (SpatialMultiPoints-class), 72
coerce, SpatialMultiPoints, SpatialPointsDataFrame-method
  (SpatialMultiPoints-class), 72
coerce, SpatialMultiPointsDataFrame, data.frame-method
  (SpatialMultiPointsDataFrame-class), 74
coerce, SpatialMultiPointsDataFrame, SpatialPointsDataFrame-method
  (SpatialMultiPointsDataFrame-class), 74
coerce, SpatialPoints, data.frame-method
  (SpatialPoints-class), 83
coerce, SpatialPoints, SpatialPolygons-method
  (SpatialPoints-class), 83
coerce, SpatialPoints, SpatialPixels-method
  (SpatialPoints-class), 83
coerce, SpatialPointsDataFrame, data.frame-method
  (SpatialPointsDataFrame-class), 85
coerce, SpatialPointsDataFrame, SpatialMultiPointsDataFrame-method
  (SpatialPointsDataFrame-class), 85
coerce, SpatialPointsDataFrame, SpatialPoints-method
  (SpatialPointsDataFrame-class), 85
coerce, SpatialPolygons, SpatialPoints-method
  (SpatialPolygons-class), 88
coerce, SpatialPolygons, SpatialPolygonsDataFrame-method
  (SpatialPolygons-class), 88
coerce, SpatialPolygonsDataFrame, data.frame-method
  (SpatialPolygonsDataFrame-class), 91
coerce, SpatialPolygonsDataFrame, SpatialLinesDataFrame-method (SpatialPolygonsDataFrame-class), 91
coerce, SpatialPolygonsDataFrame, SpatialPolygons-method (SpatialPolygonsDataFrame-class), 91
compassRose, 15
contour, SpatialGridDataFrame (image.SpatialGridDataFrame), 33
contour, SpatialPixelsDataFrame (image.SpatialGridDataFrame), 33
coordinates, 12, 16, 17, 60, 72, 75, 76, 82, 85, 86, 93, 94
coordinates, data.frame-method (coordinates-methods), 17
coordinates, GridTopology-method (coordinates-methods), 17
coordinates, Line-method (coordinates-methods), 17
coordinates, Lines-method (coordinates-methods), 17
coordinates, list-method (coordinates-methods), 17
coordinates, matrix-method (coordinates-methods), 17
coordinates, SpatialGrid-method (coordinates-methods), 17
coordinates, SpatialGridDataFrame-method (coordinates-methods), 17
coordinates, SpatialLines-method (coordinates-methods), 17
coordinates, SpatialMultiPoints-method (coordinates-methods), 17
coordinates, SpatialMultiPointsDataFrame-method (SpatialMultiPointsDataFrame-class), 74
coordinates, SpatialPixels-method (coordinates-methods), 17
coordinates, SpatialPixelsDataFrame-method (coordinates-methods), 17
coordinates, SpatialPoints-method (coordinates-methods), 17
coordinates, SpatialPointsDataFrame-method (SpatialPointsDataFrame-class), 85
coordinates, SpatialPolygons-method (coordinates-methods), 17
coordinates<-, 82
coordinates<-, data.frame-method (coordinates-methods), 17
coordinates<-, Spatial-method (Spatial-class), 61
coordinatevalues (SpatialPixels), 75
coordnames (coordnames-methods), 18
coordnames, Line-method (coordnames-methods), 18
coordnames, Lines-method (coordnames-methods), 18
coordnames, Polygon-method (coordnames-methods), 18
coordnames, Polygons-method (coordnames-methods), 18
coordnames, SpatialGrid-method (SpatialGrid-class), 63
coordnames, SpatialLines-method (coordnames-methods), 18
coordnames, SpatialMultiPoints-method (coordnames-methods), 18
coordnames, SpatialPoints-method (coordnames-methods), 18
coordnames, SpatialPolygons-method (coordnames-methods), 18
coordnames-methods, 18
coordnames<-, (coordnames-methods), 18
coordnames<-, GridTopology, character-method (coordnames-methods), 18
coordnames<-, Line, character-method (coordnames-methods), 18
coordnames<-, Lines, character-method (coordnames-methods), 18
coordnames<-, SpatialGrid, character-method (coordnames-methods), 18
coordnames<-, SpatialLines, character-method (coordnames-methods), 18
coordnames<-, SpatialMultiPoints, character-method (coordnames-methods), 18
coordnames<-, SpatialPixels, character-method (coordnames-methods), 18
coordnames<-, SpatialPoints, character-method (coordnames-methods), 18
INDEX

coordnames<-, SpatialPolygons, character-method
(coordnames-methods), 18
createSPComment, 88, 90
CRS, 19, 37, 104
CRS (CRS-class), 18
CRS-class, 7, 18, 68, 70, 71, 76, 79, 82, 87, 89, 91
CRSargs (CRS-class), 18
data.frame, 70
dd2dms, 25
dd2dms (char2dms), 14
degAxis, 21
degreeLabelsEW (mapasp), 40
degreeLabelsNS (mapasp), 40
dim.SpatialGridDataFrame
(SpatialGridDataFrame-class), 64
dim.SpatialLinesDataFrame
(SpatialLinesDataFrame-class), 70
dim.SpatialMultiPointsDataFrame
(SpatialMultiPointsDataFrame-class), 74
dim.SpatialPixelsDataFrame
(SpatialPixelsDataFrame-class), 80
dim.SpatialPointsDataFrame
(SpatialPointsDataFrame-class), 85
dim.SpatialPolygonsDataFrame
(SpatialPolygonsDataFrame-class), 91
dimensions (dimensions-methods), 22
dimensions, Spatial-method
dimensions-methods, 22
dimensions-methods, methods, 22
disaggregate (disaggregate-methods), 23
disaggregate, SpatialLines-method
disaggregate-methods, 23
disaggregate, SpatialLinesDataFrame-method
disaggregate-methods, 23
disaggregate, SpatialPolygons-method
disaggregate-methods, 23
disaggregate, SpatialPolygonsDataFrame-method
disaggregate-methods, 23
dist, 28
DMS-class, 24
filled.contour, 34
flip, 25
flipHorizontal (flip), 25
flipVertical (flip), 25
fullgrid, 77, 79
fullgrid (gridded-methods), 27
fullgrid, Spatial-method
(fullgrid-methods), 27
fullgrid<- (gridded-methods), 27
fullgrid<-, Spatial, ANY-method
(fullgrid-methods), 27
fullgrid<-, SpatialGrid, logical-method
(fullgrid-methods), 27
fullgrid<-, SpatialGridDataFrame, logical-method
(fullgrid-methods), 27
fullgrid<-, SpatialPixels, logical-method
(fullgrid-methods), 27
fullgrid<-, SpatialPixelsDataFrame, logical-method
(fullgrid-methods), 27
geometry (geometry-methods), 26
geometry, Spatial-method
(geometry-methods), 26
geometry, SpatialGridDataFrame-method
(geometry-methods), 26
geometry, SpatialLinesDataFrame-method
(geometry-methods), 26
geometry, SpatialMultiPointsDataFrame-method
(geometry-methods), 26
geometry, SpatialPointsDataFrame-method
(geometry-methods), 26
geometry, SpatialPolygonsDataFrame-method
(geometry-methods), 26
generate-methods, 26
generate<-, geometry-methods, 26
generate<-, data.frame, Spatial-method
(geometry-methods), 26
get_col_regions (spplot), 95
get_ll_TOL (is.projected), 35
get_ll_warn (is.projected), 35
get_Polypath (SpatialPolygons-class), 88
get_PolypathRule
(SpatialPolygons-class), 88
get_ReplCRS_warn (is.projected), 35
getGridIndex (SpatialPixels), 75
INDEX

getGridTopology (SpatialPixels), 75
getLinesIDSlot (sp-deprecated), 60
getLinesLineSlot (sp-deprecated), 60
getParUsrBB (Spatial-class), 61
getPolygonAreaSlot (sp-deprecated), 60
getPolygonHoleSlot (sp-deprecated), 60
getPolygonLaaptSlot (sp-deprecated), 60
getPolygonsIDSlot (sp-deprecated), 60
getPolygonsLaaptSlot (sp-deprecated), 60
getPolygonsplotOrderSlot (sp-deprecated), 60
getPolygonsPlotOrderSlot (sp-deprecated), 60
getSLLinesIDSlots (sp-deprecated), 60
getSLLinesSlot (sp-deprecated), 60
getSpatialLinesMidPoints (SpatialLines), 67
getSpatialPolygonsLabelPoints (SpatialPolygons), 87
gSpPnHoles (sp-deprecated), 60
gSpPnParts (sp-deprecated), 60
gsPPlotOrderSlot (sp-deprecated), 60
gsPPolygonIDSlots (sp-deprecated), 60
gsPPolygonsLabptSlots (sp-deprecated), 60
gsPPolygonsSlot (sp-deprecated), 60
gridIndex2nb, 28
gridlines, 29
gridparameters (gridded-methods), 27
GridsDatums, 20
GridTopology, 8
GridTopology (SpatialPixels), 75
GridTopology-class, 31, 63, 65, 76, 78–80
gt (Rlogo), 57

head.Spatial (Spatial-class), 61
HexPoints2SpatialPolygons (as.SpatialPolygons.GridTopology), 7

identicalCRS (CRS-class), 18
identify, 13
IDvaluesGridTopology (as.SpatialPolygons.GridTopology), 7
IDvaluesSpatialPixels (as.SpatialPolygons.GridTopology), 7

im-class (Spatial-class), 61
image, 34, 55
image.default, 34
image.SpatialGridDataFrame, 33, 65
image.SpatialPixels (image.SpatialGridDataFrame), 33
image.SpatialPixelsDataFrame (image.SpatialGridDataFrame), 33
image2Grid (image.SpatialGridDataFrame), 33
imageScale
(image.SpatialGridDataFrame), 33
is.projected, 35, 94
is.projected, CRS-method (CRS-class), 18
is.projected, Spatial-method (is.projected), 35

labels (gridlines), 29
layout.north.arrow (spplot), 95
layout.scale.bar (spplot), 95
lcm, 66
levelplot, 34, 40, 96–99, 105
Line, 23, 37, 38, 39
Line-class, 37, 38, 39, 50, 69
LineLength (SpatialLines), 67
Lines, 23
Lines (Line), 37
Lines-class, 37–39, 39, 50, 67–69
LinesLength (SpatialLines), 67
loadMeuse, 39
locator, 58
longlat.scales (panel.spplot), 49

make_EPSG, 20
makegrid (spsample), 100
mapasp, 13, 40
mapLegendGrob (spplot), 95
match, 41
mean, 5
merge, 41, 41
merge, Spatial, ANY-method (merge), 41
merge, Spatial, data.frame-method (merge), 41
meuse, 40, 42, 43–45
meuse.area (meuse.riv), 45
meuse.grid, 40, 42, 43, 45
meuse.grid_11, 44
meuse.riv, 45

over, 4, 5, 60, 101, 102
over (over-methods), 46
over, Spatial, Spatial-method (over-methods), 46
over, SpatialGrid, SpatialGrid-method (over-methods), 46
over, SpatialGrid, SpatialGridDataFrame-method (over-methods), 46
over, SpatialGrid, SpatialPoints-method (over-methods), 46
over, SpatialGrid, SpatialPointsDataFrame-method (over-methods), 46
over, SpatialGrid, SpatialPolygons-method (over-methods), 46
over, SpatialGrid, SpatialPolygonsDataFrame-method (over-methods), 46
over, SpatialGridDataFrame, SpatialPoints-method (over-methods), 46
over, SpatialGridDataFrame, SpatialPolygons-method (over-methods), 46
over, SpatialPoints, SpatialGrid-method (over-methods), 46
over, SpatialPoints, SpatialGridDataFrame-method (over-methods), 46
over, SpatialPoints, SpatialPixels-method (over-methods), 46
over, SpatialPointsDataFrame, SpatialPoints-method (over-methods), 46
over, SpatialPointsDataFrame, SpatialPolygons-method (over-methods), 46
over, SpatialPolygons, SpatialGrid-method (over-methods), 46
over, SpatialPolygons, SpatialGridDataFrame-method (over-methods), 46
over, SpatialPolygons, SpatialPoints-method (over-methods), 46
over, SpatialPolygons, SpatialPointsDataFrame-method (over-methods), 46
over-methods, 46
overDF_for_rgeos (over-methods), 46
owin-class (Spatial-class), 61
panel.ggmap (spplot), 95
INDEX

117

panel.gridplot, 96
panel.gridplot (panel.spplot), 49
panel.identify, 99
panel.pointsplot, 96
panel.pointsplot (panel.spplot), 49
panel.polygonsplot, 96
panel.polygonsplot (panel.spplot), 49
panel.RgoogleMaps (panel.spplot), 95
panel.spplot, 49
par, 65, 89
plot, Spatial, missing-method (Spatial-class), 61
plot, SpatialGrid, missing-method (SpatialGrid-class), 63
plot, SpatialGridDataFrame, missing-method (SpatialGridDataFrame-class), 64
plot, SpatialLines, missing-method (SpatialLines-class), 68
plot, SpatialMultiPoints, missing-method (SpatialMultiPoints-class), 72
plot, SpatialPixels, missing-method (SpatialPixels-class), 78
plot, SpatialPixelsDataFrame, missing-method (SpatialPixelsDataFrame-class), 80
plot, SpatialPoints, missing-method (SpatialPoints-class), 83
plot, SpatialPolygons, missing-method (SpatialPolygons-class), 88
plot, SpatialPolygonsDataFrame, SpatialPolygons-class), 75
plot, SpatialPointsDataFrame, SpatialPolygons-class), 84
plot, SpatialPolygonsDataFrame, SpatialPolygons-class), 75
plot, SpatialPolygonsDataFrame, SpatialPolygons-class), 84
point.in.polygon, 48, 51, 58, 102
points, SpatialMultiPointsDataFrame-method (SpatialMultiPointsDataFrame-class), 74
points, SpatialPointsDataFrame-method (SpatialPointsDataFrame-class), 85
points2grid (SpatialPixels), 75
Polygon, 23
Polygon (SpatialPolygons), 87
polygon, 89

Polygon-class, 52, 54, 87, 102
Polygons, 23
Polygons (SpatialPolygons), 87
polygons, 53
polygons, Spatial-method (polygons-methods), 54
polygons, SpatialPolygons-method (polygons-methods), 54
Polygons-class, 52, 53, 87, 89
polygons-methods, 54
polygons<-, (polygons), 53
polygons<-, data.frame, SpatialPolygons-method (polygons-methods), 54
polypath, 89
ppp-class (Spatial-class), 61
pretty, 65
print.CRS (CRS-class), 18
print.DMS (DMS-class), 24
print.SpatialMultiPoints (SpatialMultiPoints-class), 72
print.SpatialMultiPointsDataFrame (SpatialMultiPointsDataFrame-class), 74
print.SpatialPoints (SpatialPoints-class), 83
print.SpatialPointsDataFrame (SpatialPointsDataFrame-class), 85
print.summary.GridTopology (GridTopology-class), 31
print.summary.Spatial (Spatial-class), 61
print.summary.SpatialGrid (SpatialGrid-class), 63
print.summary.SpatialGridDataFrame (SpatialGridDataFrame-class), 64
print.summary.SpatialPixels (SpatialPixels-class), 78
print.summary.SpatialPixelsDataFrame (SpatialPixelsDataFrame-class), 80
proj4string, 19
proj4string (is.projected), 35
proj4string, Spatial-method (is.projected), 35
proj4string<-, (is.projected), 35
proj4string<-, Spatial, character-method
index

(is.projected), 35
proj4string<-, Spatial, CRS-method
(is.projected), 35
psp-class (Spatial-class), 61
rainbow, 12
rasterImage, 34
rbind.SpatialLines
 (SpatialLines-class), 68
rbind.SpatialLinesDataFrame
 (SpatialLinesDataFrame-class), 70
rbind.SpatialMultiPoints
 (SpatialMultiPoints-class), 72
rbind.SpatialMultiPointsDataFrame
 (SpatialMultiPointsDataFrame-class), 74
rbind.SpatialPixels
 (SpatialPixels-class), 78
rbind.SpatialPixelsDataFrame
 (SpatialPixelsDataFrame-class), 80
rbind.SpatialPoints
 (SpatialPoints-class), 83
rbind.SpatialPointsDataFrame
 (SpatialPointsDataFrame-class), 85
rbind.SpatialPolygons
 (SpatialPolygons-class), 88
rbind.SpatialPolygonsDataFrame
 (SpatialPolygonsDataFrame-class), 91
read.asciigrid, 55
readOGR, 88, 90
rebuild_CRS (CRS-class), 18
rebuild_CRS, CRS-method (CRS-class), 18
rebuild_CRS, Spatial-method
 (Spatial-class), 61
recenter (recenter-methods), 56
recenter, Line-method
 (recenter-methods), 56
recenter, Lines-method
 (recenter-methods), 56
recenter, Polygon-method
 (recenter-methods), 56
recenter, Polygons-method
 (recenter-methods), 56
recenter, SpatialLines-method
 (recenter-methods), 56
recenter, SpatialPolygons-method
 (recenter-methods), 56
remove.duplicates (zerodist), 107
Rlogo, 57
row.names.SpatialLines
 (SpatialLines-class), 68
row.names.SpatialLinesDataFrame
 (SpatialLinesDataFrame-class), 70
row.names.SpatialMultiPoints
 (SpatialMultiPoints-class), 72
row.names.SpatialMultiPointsDataFrame
 (SpatialMultiPointsDataFrame-class), 74
row.names.SpatialPoints
 (SpatialPoints-class), 83
row.names.SpatialPointsDataFrame
 (SpatialPointsDataFrame-class), 85
row.names.SpatialPolygons
 (SpatialPolygons-class), 88
row.names.SpatialPolygonsDataFrame
 (SpatialPolygonsDataFrame-class), 91
sample, 102
select.spatial, 58
set_col_regions (spplot), 95
set_ll_TOL, 61
set_ll_TOL (is.projected), 35
set_ll_warn, 61
set_ll_warn (is.projected), 35
set_Polypath (SpatialPolygons-class), 88
set_PolypathRule
 (SpatialPolygons-class), 88
set_ReplCRS_warn (is.projected), 35
setParUsrBB (Spatial-class), 61
show, CRS-method (CRS-class), 18
show, DMS-method (DMS-class), 24
show, GridTopology-method
 (GridTopology-class), 31
show, SpatialGrid-method
 (SpatialGrid-class), 63
show, SpatialGridDataFrame-method
 (SpatialGridDataFrame-class), 64
show, SpatialMultiPoints-method
 (SpatialMultiPoints-class), 72
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sppanel.list-method</code></td>
<td>95</td>
</tr>
<tr>
<td><code>sppanel.NULL-method</code></td>
<td>95</td>
</tr>
<tr>
<td><code>sppanel.SpatialGrid-method</code></td>
<td>95</td>
</tr>
<tr>
<td><code>sppanel.SpatialLines-method</code></td>
<td>95</td>
</tr>
<tr>
<td><code>sppanel.SpatialPoints-method</code></td>
<td>95</td>
</tr>
<tr>
<td><code>sppanel.SpatialPolygons-method</code></td>
<td>95</td>
</tr>
<tr>
<td><code>sppanel-methods</code></td>
<td>95</td>
</tr>
<tr>
<td><code>spplot</code></td>
<td>13, 50, 51, 95, 97, 105</td>
</tr>
<tr>
<td><code>spplot</code>, <code>SpatialGridDataFrame-method</code></td>
<td>95</td>
</tr>
<tr>
<td><code>spplot</code>, <code>SpatialLinesDataFrame-method</code></td>
<td>95</td>
</tr>
<tr>
<td><code>spplot</code>, <code>SpatialMultiPointsDataFrame-method</code></td>
<td>95</td>
</tr>
<tr>
<td><code>spplot</code>, <code>SpatialPointsDataFrame-method</code></td>
<td>95</td>
</tr>
<tr>
<td><code>spplot</code>, <code>SpatialPolygonsDataFrame-method</code></td>
<td>95</td>
</tr>
<tr>
<td><code>spplot-methods</code></td>
<td>51, 98</td>
</tr>
<tr>
<td><code>spplot-methods</code></td>
<td>95</td>
</tr>
<tr>
<td><code>spplot.grid</code></td>
<td>95</td>
</tr>
<tr>
<td><code>spplot.key</code></td>
<td>49</td>
</tr>
<tr>
<td><code>spplot.locator</code></td>
<td>95</td>
</tr>
<tr>
<td><code>spplot.polygons</code></td>
<td>95</td>
</tr>
<tr>
<td><code>spsample</code></td>
<td>8, 100</td>
</tr>
<tr>
<td><code>spsample</code>, <code>Line-method</code></td>
<td>100</td>
</tr>
<tr>
<td><code>spsample</code>, <code>Lines-method</code></td>
<td>100</td>
</tr>
<tr>
<td><code>spsample</code>, <code>Polygon-method</code></td>
<td>100</td>
</tr>
<tr>
<td><code>spsample</code>, <code>Spatial-method</code></td>
<td>100</td>
</tr>
<tr>
<td><code>spsample</code>, <code>SpatialGrid-method</code></td>
<td>100</td>
</tr>
<tr>
<td><code>spsample</code>, <code>SpatialLines-method</code></td>
<td>100</td>
</tr>
<tr>
<td><code>spsample</code>, <code>SpatialPolygons-method</code></td>
<td>100</td>
</tr>
<tr>
<td><code>spsample-methods</code></td>
<td>101</td>
</tr>
<tr>
<td><code>spsample-methods</code></td>
<td>100</td>
</tr>
<tr>
<td><code>spTransform</code></td>
<td>30, 103, 104</td>
</tr>
<tr>
<td><code>spTransform</code>, <code>Spatial, ANY-method</code></td>
<td>103</td>
</tr>
<tr>
<td><code>spTransform</code>, <code>Spatial, character-method</code></td>
<td>103</td>
</tr>
<tr>
<td><code>spTransform</code>, <code>Spatial, CRS-method</code></td>
<td>103</td>
</tr>
<tr>
<td><code>spTransform-methods</code></td>
<td>103</td>
</tr>
<tr>
<td><code>stack</code></td>
<td>104, 105</td>
</tr>
<tr>
<td><code>stack.SpatialGridDataFrame</code></td>
<td>104</td>
</tr>
<tr>
<td><code>subset.data.frame</code></td>
<td>61</td>
</tr>
<tr>
<td><code>subset.Spatial</code></td>
<td>61</td>
</tr>
<tr>
<td><code>sum</code></td>
<td>5</td>
</tr>
<tr>
<td><code>summary</code>, <code>GridTopology-method</code></td>
<td>31</td>
</tr>
<tr>
<td><code>summary</code>, <code>Spatial-method</code></td>
<td>61</td>
</tr>
<tr>
<td><code>summary</code>, <code>SpatialGrid-method</code></td>
<td>63</td>
</tr>
<tr>
<td><code>summary</code>, <code>SpatialLines-method</code></td>
<td>68</td>
</tr>
<tr>
<td><code>summary</code>, <code>SpatialMultiPoints-method</code></td>
<td>72</td>
</tr>
<tr>
<td><code>summary</code>, <code>SpatialPoints-method</code></td>
<td>78</td>
</tr>
<tr>
<td><code>summary</code>, <code>SpatialPoints-method</code></td>
<td>83</td>
</tr>
<tr>
<td><code>summary</code>, <code>SpatialPolygons-method</code></td>
<td>88</td>
</tr>
<tr>
<td><code>summary</code>, <code>SpatialMultiPoints</code></td>
<td>72</td>
</tr>
<tr>
<td><code>summary</code>, <code>SpatialPoints</code></td>
<td>83</td>
</tr>
<tr>
<td><code>surfaceArea</code></td>
<td>105</td>
</tr>
<tr>
<td><code>surfaceArea</code>, <code>matrix-method</code></td>
<td>105</td>
</tr>
<tr>
<td><code>surfaceArea</code>, <code>SpatialGridDataFrame-method</code></td>
<td>105</td>
</tr>
<tr>
<td><code>surfaceArea</code>, <code>SpatialPixelsDataFrame-method</code></td>
<td>105</td>
</tr>
<tr>
<td><code>surfaceArea</code>, <code>SpatialPointsDataframe</code></td>
<td>105</td>
</tr>
<tr>
<td><code>tail.Spatial</code></td>
<td>61</td>
</tr>
<tr>
<td><code>text</code></td>
<td>29, 30</td>
</tr>
<tr>
<td><code>text(gridlines)</code></td>
<td>29</td>
</tr>
<tr>
<td><code>wkt</code></td>
<td>35</td>
</tr>
<tr>
<td><code>wkt</code>, <code>CRS-method</code></td>
<td>18</td>
</tr>
</tbody>
</table>
wkt, Spatial-method (is.projected), 35
write.asciigrid (read.asciigrid), 55
write.table, 55

xyplot, 13, 96, 98, 99
zerodist, 107
zerodist2 (zerodist), 107