The sp Package

July 27, 2006

Version 0.8-18

Date 2006-07-10

Title classes and methods for spatial data

Author Edzer J. Pebesma <e.pebesma@geo.uu.nl>, Roger Bivand <Roger.Bivand@nhh.no> and others

Maintainer Edzer J. Pebesma <e.pebesma@geo.uu.nl>

Depends R (>= 2.2.0), methods

Suggests RColorBrewer

Imports lattice, grid

Description A package that provides 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 version 2 or later

URL http://r-spatial.sourceforge.net/

SaveImage yes

Collate AAA.R Class-CRS.R CRS-methods.R Class-Spatial.R Spatial-methods.R projected.R Class-SpatialPoints.R SpatialPoints-methods.R

R topics documented:

`sp` *A package providing classes and methods for spatial data: points, lines, polygons and grids*

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. The package spproj needs to be available if one wants to work with (non-missing valued) coordinate reference systems; this packages needs, but does not contain the PROJ.4 external library, and is therefore not available for Windows R versions on CRAN[1]

[1] a binary is available from http://www.sourceforge.net/projects/r-spatial/

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, SpatialCell and SpatialGridded can be of arbitray 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, SpatialCellDataFrame, SpatialLinesDataFrame, SpatialPolygonsDataFrame extend the primary spatial information with attribute data.

References

PROJ.4: http://www.remotesensing.org/proj/

GDAL and OGR: http://www.remotesensing.org/gdal/.

Authors

sp is a collaborative effort of Edzer Pebesma, Roger Bivand, Barry Rowlinson and Virgilo Gómez-Rubio.

AttributeList-class

Class "AttributeList"

Description

Attribute list; a kind of data.frame but without row.names

Objects from the Class

Objects can be created by calls of the form `AttributeList(x)`, with `x` a list having elements of equal length.

Slots

att: Object of class `"list"`, containing the attributes variables. The variables are guaranteed to have equal length, and can be of any type that has a length

Methods

[`signature(x = "AttributeList")`: subsets

coerce `signature(from = "AttributeList", to = "data.frame")`: convert the list slot to data.frame

coerce `signature(from = "AttributeList", to = "list")`: retrieve the list slot

coerce `signature(from = "data.frame", to = "AttributeList")`: ...

coerce `signature(from = "list", to = "AttributeList")`: ...

as.list coerce AttributeList object to list

as.data.frame coerce AttributeList object to data.frame

dim retrieve dimensions of object, as if it were a data.frame: `dim(x)[1]` is the length of variables (nr or rows), `dim(x)[2]` the number of variables (nr of columns)

[[, $ retrieve a list element

[[<-, $<- assign a list element

summary `signature(object = "AttributeList")` summarize object; calls `summary.data.frame` on the list

names get names from the list

names<- assign names to the list

Note

In earlier versions (sp <= 0.7-6), objects of class SpatialPointsDataFrame-class, SpatialPixelsDataFrame-class and SpatialGridDataFrame-class had a slot of class data.frame to store the attribute information. It turns out that especially for larger grids, holding the grid attribute information in a data.frame requires up to a factor 11 more memory than needed by the data alone, for the sole reason that R creates unique row.names for each row (i.e., value).

To get around this problem, we created the simple S4 AttributeList class which has a single slot which is a list, without row.names, but with a validator function that checks that all elements in the list have equal length. The class is meant to behave similar to a data.frame, having a similar set of methods. It does not automatically convert character data to factors.

If you have row.names in your data.frame, and you want to keep them when you convert them to e.g. a SpatialPointsDataFrame-class object using coordinates, you should first copy the row.names to a column in the data.frame. If not, they will get lost.

Author(s)

Edzer J. Pebesma, ⟨e.pebesma@geo.uu.nl⟩

See Also

list, data.frame

Examples

```
x = AttributeList(list(a = 1:10, b = sample(1:10), c = runif(10)))
x
summary(x)
x = object.size(AttributeList(list(a = 1:1000000)))
y = object.size(data.frame(a = 1:1000000))
y / x
```

CRS-class *Class "CRS" of coordinate reference system arguments*

Description

Interface class to the PROJ.4 projection system. The class is defined as an empty stub accepting value NA in the sp package. If the rgdal package is available, then the class will permit spatial data to be associated with coordinate reference systems. 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.

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. The initiation function calls the PROJ.4 library 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 argument list used by the PROJ.4 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.

Methods

show `signature(object = "CRS")`: print projection arguments in object

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. These are added to in successive releases, so tracking the website or compiling and installing the most recent revisions will give the greatest choice. On occasion, ellipsoid parameters may be passed through directly. 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).

Author(s)

Roger Bivand ⟨Roger.Bivand@nhh.no⟩

References

http://www.remotesensing.org/proj/

Examples

```
if (require(rgdal)) {
  print(CRSargs(CRS("+proj=longlat +datum=NAD27")))
  print(CRSargs(CRS("+init=epsg:4267")))
  print(CRSargs(CRS("+init=epsg:26978")))
  print(CRSargs(CRS("+proj=stere +lat_0=52.15616055555555 +lon_0=5.38763888888889 +k=0.99
  print(CRSargs(CRS("+init=epsg:28992")))
}
```

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

Methods

coerce `signature(from = "DMS", to = "numeric")`: convert to decimal degrees

show `signature(object = "DMS")`: print data values

Author(s)

Roger Bivand ⟨Roger.Bivand@nhh.no⟩

See Also

char2dms, dd2dms

Examples

```
data(state)
dd2dms(state.center$x)
dd2dms(state.center$y, NS=TRUE)
as.numeric(dd2dms(state.center$y))
as(dd2dms(state.center$y, NS=TRUE), "numeric")
as.numeric.DMS(dd2dms(state.center$y))
state.center$y
```

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 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

bbox: Object of class "matrix"; bounding box

proj4string: Object of class "CRS"; projection

Extends

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

Note

Author(s)

Edzer J. Pebesma, ⟨e.pebesma@geo.uu.nl⟩

References

See Also

SpatialGridDataFrame-class, SpatialGrid-class

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
```

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 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

Examples

Line *create objects of class Line or Lines*

Description

create objects of class `Line` or `Lines` from coordinates

Usage

```
Line(coords)
Lines(slinelist, ID = as.character(NA))
```

Arguments

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

Value

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

See Also

SpatialLines-class

Lines-class *Class "Lines"*

Description

class for sets of line objects

Usage

```
getLinesLinesSlot(SL)
getLinesIDSlot(Lines)
```

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:` Object of class `"character"`, 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

Examples

`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 area of the polygon

`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.

Note

Author(s)

Roger Bivand

References

See Also

Polygons-class, SpatialPolygons-class

Examples

`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 total area of the Polygon list including the areas of holes; these values are used to make sure that polygons of a smaller area are plotted after polygons of a larger area

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 spgpc wrapper package for gpclib functions, function checkPolygonsHoles() (currently on sourceforge).

Author(s)

Roger Bivand

References

See Also

Examples

`Rlogo` *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(cells
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)
```

`Spatial-class` *Class "Spatial"*

Description

An abstract class from which useful spatial classes are derived

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.

proj4string: Object of class `"CRS"`; holding a valid proj4 string, which can be used for unprojecting or reprojecting coordinates; it is initialised to NA. Other strings are checked for validity in the spproj package, but attempts to assign a string containing "longlat" to data extending beyond longitude [-180, 360] or lattitude [-90, 90] will be stopped.

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

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)

Author(s)

r-spatial team; Edzer J. Pebesma, ⟨e.pebesma@geo.uu.nl⟩ Roger Bivand, Barry Rowlinson, Virgilio Gómez-Rubio

References

See Also

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

Examples

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

grid object of class GridTopology-class, defining the grid topology (offset, cellsize, dim)

grid.index index of points in full grid, or integer(0)

coords coordinates of points, or bbox of grid

bbox: Object of class "matrix"; bounding box

proj4string: Object of class "CRS"; projection

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

Note

Author(s)

Edzer J. Pebesma, ⟨e.pebesma@geo.uu.nl⟩

References

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
```

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$double.eps))
SpatialGrid(grid, proj4string = CRS(as.character(NA)))
coordinatevalues(obj)
points2grid(points, tolerance = sqrt(.Machine$double.eps))
getGridIndex(cc, grid, all.inside = TRUE)
getGridTopology(obj)
areaSpatialGrid(obj)
```

Arguments

cellcentre.offset	numeric; vector with the smallest coordinates for each dimension
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
tolerance	precision, used to which extent points are exactly on a grid
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. instead 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 succees 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 J. Pebesma, ⟨e.pebesma@geo.uu.nl⟩

References

See Also

SpatialGrid-class, SpatialGridDataFrame-class,

Examples

```
x = GridTopology(c(0,0), c(1,1), c(5,4))
class(x)
x
summary(x)
coordinates(x)
coordinates(GridTopology(c(0,0), c(1,1), c(5,4)))
coordinatevalues(x)
```

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;

Slots

points: see SpatialPoints; points slot which is not actually filled with all coordinates (only with min/max)

grid: see GridTopology-class; grid parameters

grid.index: see SpatialPixels-class; this slot is of zero length for this class, as the grid is full

bbox: Object of class `"matrix"`; bounding box

proj4string: Object of class `"CRS"`; projection

data: Object of class AttributeList-class, 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`

[[`signature(x = "SpatialGridDataFrame")`: retrieves an attribute, dropping everything else (topology)

[[<- `signature(x = "SpatialGridDataFrame")`: assigns or replaces an attribute

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

cbind `signature(...)`: if arguments have identical topology, combine their attribute values

Note

Author(s)

Edzer J. Pebesma, ⟨e.pebesma@geo.uu.nl⟩

References

See Also

SpatialGrid-class, which does not contain the attribute data, and SpatialPixelsDataFrame-class which holds possibly incomplete grids

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())
```

SpatialPixelsDataFrame

define spatial grid with attribute data

Description

defines spatial grid by offset, cell size and dimensions

Usage

```
SpatialPixelsDataFrame(points, data, tolerance = sqrt(.Machine$double.eps),
                proj4string = CRS(as.character(NA)))
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
data	data.frame; contains the attribute (actual grid) data
tolerance	precision up to which extent points should be exactly on a grid
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 J. Pebesma

References

See Also

gridded, gridded<-, SpatialGrid, SpatialGrid-class

Examples

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

SpatialLines-class *a class for spatial lines*

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 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

summary \signature(object = "SpatialLines"): summarize object

Note

Author(s)

Roger Bivand, Edzer Pebesma

References

See Also

Line-class, Lines-class

Examples

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 od Lines from a SpatialLines object

Usage

```
SpatialLines(LinesList, proj4string = CRS(as.character(NA)))
SpatialLinesDataFrame(sl, data, match.ID = TRUE)
as.SpatialLines.SLDF(SLDF)
getSLlinesSlot(SL)
getSLLinesIDSlots(SL)
```

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
SLDF	SpatialLinesDataFrame object

Value

SpatialLines returns object of class SpatialLines; SpatialLinesDataFrame returns object of class SpatialLinesDataFrame

See Also

SpatialLines-class

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:

[`signature(x = "SpatialLinesDataFrame")`: subset rows or columns; in case of row subsetting, the line sets are also subsetted

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

summary `signature(object = "SpatialLinesDataFrame")`: summarize object

Note

Author(s)

Roger Bivand; Edzer Pebesma

References

See Also

SpatialLines-class

Examples

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 grid

bbox: Object of class "matrix"; bounding box

proj4string: 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

"[" signature(x = "SpatialPixels"): select pixel cells; the argument drop=TRUE (default) recalculates grid topology for the selection, if drop=FALSE the grid topology of the parent object is kept.

Note

Author(s)

Edzer J. Pebesma, ⟨e.pebesma@geo.uu.nl⟩

References

See Also

SpatialPixelsDataFrame-class, SpatialGrid-class

Examples

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

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 or unordered non-NA cells;

Slots

bbox: Object of class `"matrix"`; bounding box

proj4string: Object of class `"CRS"`; projection

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 hight except y, which decreases from high to low

data: Object of class AttributeList-class containing the attribute data

Extends

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

Methods

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

summary `signature(object = "SpatialPixelsDataFrame")`: summarize object

[`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

[[`signature(x = "SpatialPixelsDataFrame")`: retrieves an attribute, dropping everything else (topology)

[[<- `signature(x = "SpatialPixelsDataFrame")`: assigns or replaces an attribute

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

Note

Author(s)

Edzer J. Pebesma, ⟨e.pebesma@geo.uu.nl⟩

References

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-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 can be subsetted

coerce signature(from = "SpatialPoints", to = "data.frame"): retrieves the data part

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

Note

Author(s)

Edzer J. Pebesma, ⟨e.pebesma@geo.uu.nl⟩

References

put references to the literature/web site here

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)
```

`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)))
SpatialPointsDataFrame(coords, data, coords.nrs = numeric(0),
                proj4string = CRS(as.character(NA)), match.ID = TRUE)
```

Arguments

`coords`	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
`data`	object of class `data.frame` or of class AttributeList-class; 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; if TRUE AND coords has rownames (i.e., coerced to a matrix, `dimnames(coords)[[2]]` is not NULL), AND data has row.names (i.e. is a data.frame), then the `SpatialPointsDataFram` 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.

Value

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

See Also

coordinates, SpatialPoints-class, SpatialPointsDataFrame-class

`SpatialPointsDataFrame-class`
Class "SpatialPointsDataFrame"

Description

Class for spatial attributes that have spatial point locations

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 AttributeList-class 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.stripped` 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"`.

Methods

[`signature(x = "SpatialPointsDataFrame")`: subset rows or columns; in case of row subsetting, the coordinates are also subsetted

coerce `signature(from = "SpatialPointsDataFrame", to = "data.frame")`: extracts the AttributeList part, and converts it to a data.frame

coerce `signature(from = "SpatialPointsDataFrame", to = "AttributeList")`: extracts the AttributeList part

coordinates `signature(obj = "SpatialPointsDataFrame")`: retrieves the coordinates only

show `signature(object = "SpatialPointsDataFrame")`: print method

summary `signature(object = "SpatialPointsDataFrame")`: summarize object

points `signature(x = "SpatialPointsDataFrame")`: add points to plot

Note

Author(s)

Edzer J. Pebesma, ⟨e.pebesma@geo.uu.nl⟩

References

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-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 Polygons-class

plotOrder: Object of class "integer"; integer array giving the order in which objects should be plotted

bbox: Object of class "matrix"; see Spatial-class

proj4string: Object of class "CRS"; see CRS-class

Extends

Class "Spatial", directly.

Methods

Methods defined with class "SpatialPolygons" in the signature:

[`signature(obj = "SpatialPolygons")`: select subset of (sets of) polygons

plot `\signature(x = "SpatialPolygons", y = "missing")`: plot polygons in SpatialPolygons object

summary `\signature(object = "SpatialPolygons")`: summarize object

Note

Author(s)

Roger Bivand

References

See Also

SpatialPolygons

Examples

```
grd <- GridTopology(c(1,1), c(1,1), c(10,10))
polys <- as.SpatialPolygons.GridTopology(grd)
plot(polys)
text(getSpPPolygonsLabptSlots(polys), labels=getSpPPolygonsIDSlots(polys), cex=0.6)
```

SpatialPolygons *create objects of class SpatialPolygons or SpatialPolygonsDataFrame*

Description

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)
```

Arguments

coords	2-column numeric matrix with coordinates; first point (row) should equal last coordinates (row)
proj4string	projection string of class CRS-class
hole	logical value for setting polygon as hole or not
srl	list with Polygon-class objects
ID	character vector of length one with identifier
Srl	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

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

See Also

SpatialPolygons-class, SpatialPolygonsDataFrame-class

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

summary `\signature(object = "SpatialPolygonsDataFrame")`: summarize object

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.

Author(s)

References

See Also

SpatialPolygons-class

Examples

```
data(ncshp)
grd <- GridTopology(c(1,1), c(1,1), c(10,10))
polys <- as.SpatialPolygons.GridTopology(grd)
centroids <- getSpPPolygonsLabptSlots(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=getSpP
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)])
```

---

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. 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, proj4string=CRS(as.character(NA)))
IDvaluesSpatialPixels(obj)
```

Arguments

grd,obj	GridTopology object or SpatialPixels object
proj4string	projection string of class CRS

Value

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

See Also

GridTopology, SpatialPixels, SpatialPolygons

Examples

```
grd <- GridTopology(cellcentre.offset=c(-175,55), cellsize=c(10,10), cells.dim=c(4,4))
SpP_grd <- as.SpatialPolygons.GridTopology(grd)
plot(SpP_grd)
text(getSpPPolygonsLabptSlots(SpP_grd), getSpPPolygonsIDSlots(SpP_grd), 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=getSpPPol
SpPDF <- SpatialPolygonsDataFrame(SpP_grd, trdata)
spplot(SpPDF)
```

---

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

```
grd <- GridTopology(c(1,1), c(1,1), c(10,10))
polys <- as.SpatialPolygons.GridTopology(grd)
plot(polys)
text(getSpPPolygonsLabptSlots(polys), labels=getSpPPolygonsIDSlots(polys), cex=0.6)
```

---

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 J. Pebesma, e.pebesma@geo.uu.nl

See Also

as.image.SpatialGridDataFrame, image

Examples

```
x <- read.asciigrid(system.file("external/test.ag", package="sp")[1])
class(x)
image(x)
```

---

bbox-methods *retrieve bbox from spatial data*

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

```
# 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)

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

`bpy.colors` *blue-pink-yellow color scheme that prints well on black/white printers*

Description

Create a vector of 'n' "contiguous" colors.

Usage

```
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; R implementation Edzer J. Pebesma, ⟨e.pebesma@geo.uu.nl⟩

References

see http://www.ihe.uni-karlsruhe.de/mitarbeiter/vonhagen/palette.en.html; gnuplot has this color map.

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")
gridded(p) <- TRUE
image(p, col = bpy.colors(100), asp = 1)
# require(lattice)
# trellis.par.set("regions", list(col=bpy.colors())) # make this default pallette
```

bubble *Create 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(2,3), key.entries = quantile(data[,zcol]), main,
        identify = FALSE, labels = row.names(data.frame(obj)), key.space = "righ
        scales = list(draw = FALSE), xlab = NULL, ylab = NULL)
```

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.
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

Value

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

Author(s)

Edzer J. Pebesma

References

See Also

xyplot, mapasp, identify

Examples

```
data(meuse)
coordinates(meuse) <- c("x", "y") # promote to SpatialDataFrame
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 function is also available)

Author(s)

Roger Bivand ⟨Roger.Bivand@nhh.no⟩

See Also

DMS-class

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))
```

---

contourLines2SLDF *Converter functions to build SpatialLinesDataFrame objects*

Description

These functions show how to build converters to SpatialLinesDataFrame objects: contourLines2SLDF from the list returned by the contourLines function in the graphics package (here the data frame is just the contour levels, with one Lines object made up of at least one Line object per level). In addition, Mapgen2SL reads a file in "Mapgen" format into a SpatialLines object.

Usage

```
contourLines2SLDF(cL, proj4string=CRS(as.character(NA)))
Mapgen2SL(file, proj4string=CRS(as.character(NA)))
```

Arguments

cL	a list returned by the contourLines function in the graphics package
proj4string	Object of class "CRS"; see CRS-class
file	filename of a file containing a Mapgen line data set

Value

A SpatialLinesDataFrame object

Note

Coastlines of varying resolution may be chosen online and downloaded in "Mapgen" text format from http://www.ngdc.noaa.gov/mgg/shorelines/shorelines.html, most conveniently using the interactive selection tool, but please note the 500,000 point limit on downloads, which is easy to exceed.

Author(s)

Roger Bivand; Edzer Pebesma

See Also

SpatialLines-class

Examples

```
#data(lnsshp) # retrieved as:
# library(maptools)
# lns <- read.shape(system.file("shapes/fylk-val.shp", package="maptools")[1])
#res <- shp2SLDF(lns)
#plot(res)
#invisible(title(""))
#data(co37_d90_arc) # retrieved as:
# library(RArcInfo)
# fl <- "http://www.census.gov/geo/cob/bdy/co/co90e00/co37_d90_e00.zip"
# download.file(fl, "co37_d90_e00.zip")
# e00 <- zip.file.extract("co37_d90.e00", "co37_d90_e00.zip")
# e00toavc(e00, "ncar")
# arc <- get.arcdata(".", "ncar")
#res <- arcobj2SLDF(arc)
#plot(res)
#invisible(title(""))
res <- contourLines2SLDF(contourLines(volcano))
plot(res, col=terrain.colors(nrow(res@data)))
invisible(title(""))
```

---

coordinates-methods

retrieve (or set) spatial coordinates

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 = "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

---

`coordinates` *sets spatial coordinates to create spatial data, or retrieves spatial coordinates*

Description

sets spatial coordinates to create spatial data, or retrieves spatial coordinates

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.

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)
```

---

coordnames-methods *retrieve or assign coordinate names for classes in sp*

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" assigns coordinate names

x = "SpatialLines", value = "character" assigns coordinate names

x = "Lines", value = "character" assigns coordinate names

x = "Line", value = "character" assigns coordinate names

x = "SpatialPolygons", value = "character" assigns coordinate names

x = "Polygons", value = "character" assigns coordinate names

x = "Polygon", value = "character" assigns coordinate names

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
...	passed to the actual axis call

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")),xlim=c(-1,1),ylim=c(-1,1))
degAxis(1)
degAxis(2, at = c(-1,-0.5,0,0.5,1))
#
```

dimensions-methods *retrieve spatial dimensions from spatial data*

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 deriviving 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)
dimensions(S)

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

---

gridded-methods *specify spatial data as being gridded, or find out whether they are*

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

```
gridded(obj)
gridded(obj) <- TRUE
fullgrid(obj)
fullgrid(obj) <- TRUE
gridparameters(obj)
```

Arguments

obj	object deriviving from class "Spatial" (for gridded), or

object of class SpatialGridDataFrame-class (for fullgrid and gridparameters)

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 SpatialDataFrameGrid 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.

Methods

obj = "Spatial" object deriviving 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)
```

`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; `gridat` permits the construction of points and labels for non-projected grid annotation

Usage

```
gridlines(x, easts = pretty(bbox(x)[1,]), norths = pretty(bbox(x)[2,]), ndiscr =
gridat(x, easts = pretty(bbox(x)[1,]), norths = pretty(bbox(x)[2,]), offset=0.5)
```

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

Details

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

Author(s)

Edzer J. Pebesma, ⟨e.pebesma@geo.uu.nl⟩, using example code of Roger Bivand.

See Also

Function transform in package spproj

Examples

```
data(meuse)
coordinates(meuse) = ~x+y
plot(meuse)
plot(gridlines(meuse), add=TRUE)
title("default gridlines within Meuse bounding box")
```

---

image.SpatialGridDataFrame

image gridded spatial data, or convert to format for image

Description

Convert gridded data in SpatialGridDataFrame to image format; call image on data in SpatialGridDataFrame format

Usage

```
image.SpatialGridDataFrame(x, attr = 1, xcol = 1, ycol = 2,
        red=NULL, green=NULL, blue=NULL, asp = 1, axes = FALSE, xlim = NULL,
        ylim = NULL, add = FALSE, ...)
as.image.SpatialGridDataFrame(x, xcol = 1, ycol = 2)
```

Arguments

x	object of class SpatialGridDataFrame
attr	column number 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
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
asp	aspect ratio of unit x and unit y axis
axes	logical; should coordinate axes be drawn?
xlim	x-axis limits
ylim	y-axis limits
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

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.

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

Author(s)

Edzer J. Pebesma

References

See Also

image.default, SpatialGridDataFrame-class, levelplot in package lattice

Examples

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

Usage

```
is.projected(sd)
proj4string(sd)
proj4string(sd) <- value
```

Arguments

sd	An object of class or extending Spatial-class
value	CRS object, containing a valid proj4 string; attempts to assign an object containing "longlat" to data extending beyond longitude [-180, 360] or lattitude [-90, 90] will be stopped

Details

proj4 strings are operative through non-CRAN package spproj, see http://r-spatial.sourceforge.net

Value

is.projected returns a logical; proj4string returns a character vector of length 1; spatial.dimension returns the number of spatial dimensions (2 or 3).

Author(s)

Edzer J. Pebesma, ⟨e.pebesma@geo.uu.nl⟩

See Also

CRS

Examples

`mapasp` *Calculate aspect ratio for plotting geographic maps*

Description

Calculate aspect ratio for plotting geographic maps

Usage

```
mapasp(data, xlim, ylim)
```

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)

Value

`mapasp` is used for the aspect argument in lattice plots and spplot;

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".

See Also

levelplot in package `lattice`

`meuse` *Meuse river data set*

Description

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

Usage

```
data(meuse)
```

Format

This data frame contains the following columns:

x a numeric vector; x-coordinate (m) in RDM (Dutch topographical map coordinates)

y a numeric vector; y-coordinate (m) in RDM (Dutch topographical map coordinates)

cadmium topsoil cadmium concentration, ppm.; note that zero cadmium values in the original data set have been shifted to 0.2 (half the lowest non-zero value)

copper topsoil copper concentration, ppm.

lead topsoil lead concentration, ppm.

zinc topsoil zinc concentration, ppm.

elev relative elevation

dist distance to river Meuse; obtained from the nearest cell in meuse.grid, which in turn was derived by a spread (spatial distance) GIS operation, therefore it is accurate up to 20 metres; normalized $[0, 1]$

om organic matter, as percentage

ffreq flooding frequency class

soil soil type

lime lime class

landuse landuse class

dist.m distance to river Meuse (metres), as obtained during the field survey

Note

row.names refer to the original sample number

Author(s)

The actual field data were collected by Ruud van Rijn and Mathieu Rikken; compiled for R by Edzer J. Pebesma

References

P.A. Burrough, R.A. McDonnell, 1998. Principles of Geographical Information Systems. Oxford University Press.

http:/www.gstat.org/

Examples

```
data(meuse)
summary(meuse)
```

---

meuse.grid *Prediction Grid for Meuse Data Set*

Description

The `meuse.grid` data frame has 3103 rows and 2 columns; a grid with 40 m x 40 m spacing that covers the Meuse Study area

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 flood frequency; the lower the value, the larger the flood frequency; the origin of this item is questionable

part.a arbitrary division of the area in two areas, a and b

part.b see `part.a`

soil soil type; it is questionable whether these data come from a real soil map

Details

`x` and `y` are in RDM, 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

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

meuse.riv *River Meuse outline*

Description

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

Usage

```
data(meuse.riv)
```

Format

This data frame contains a 176 x 2 matrix with coordinates.

Details

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

References

See the meuse documentation

Examples

```
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")
)
```

nowrapSpatialLines *Split SpatialLines components at offset*

Description

When recentering a world map, most often from the "Atlantic" view with longitudes with range -180 to 180, to the "pacific" view with longitudes with range 0 to 360, lines crossing the offset (0 for this conversion) get stretched horizonally. This function breaks Line objects at the offset (usually Greenwich), inserting a very small gap, and reassembling the Line objects created as Lines. A similar function for polygons is found in the **spgpc** package.

Usage

```
nowrapSpatialLines(obj, offset = 0, eps = rep(.Machine$double.eps, 2))
```

Arguments

obj	A Spatial Lines object
offset	default 0, untried for other values
eps	vector of two fuzz values, both default double.eps

Value

A Spatial Lines object

Author(s)

Roger Bivand

Examples

```
Sl <- SpatialLines(list(Lines(list(Line(cbind(sin(seq(-4,4,0.4)), seq(1,21,1)))))), proj4
summary(Sl)
#lapply(sapply(getSLlinesSlot(Sl), getLinesLinesSlot), bbox)
#lapply(sapply(getSLlinesSlot(recenter(Sl)), getLinesLinesSlot), bbox)
nwSL <- nowrapSpatialLines(Sl)
summary(nwSL)
#lapply(sapply(getSLlinesSlot(nwSL), getLinesLinesSlot), bbox)
#lapply(sapply(getSLlinesSlot(recenter(nwSL)), getLinesLinesSlot), bbox)
```

overlay-methods *Methods for spatially overlay-ing points (grids) and polygons layers*

Description

overlay combines points (or grids) and polygons by performing point-in-polygon operation on all point-polygons combinations.

Methods

x = "SpatialPoints", y = "SpatialPolygons" returns a numeric vector of length equal to the number of points; the number is the id (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 values of the `x` points fallin in each polygon (set) of `y`

x = "SpatialPolygons", y = "SpatialPoints" returns the polygon id of points in `y`; if `x` is a `SpatialRingDataFram` a data.frame with rows from `x` corresponding to ponits 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

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

Note

points on a polygon boundary and points corresponding to a polygon vertex are considered to be inside the polygon

Author(s)

Edzer J. Pebesma, ⟨e.pebesma@geo.uu.nl⟩

See Also

overlay, point.in.polygon

`overlay` *spatial overlay for points, grids and polygons*

Description

Usage

```
overlay(x, y, ...)
```

Arguments

`x`	first layer
`y`	second layer, put on top of x
`...`	optional arguments; see example below

Value

a numerical array of indices of `x` on locations of `y`, or a data.frame with (possibly aggregate) properties of `x` in units of `y`.

Note

points on a polygon boundary and points corresponding to a polygon vertex are considered to be inside the polygon

See Also

overlay-methods, point.in.polygon

Examples

```
r1 = cbind(c(180114, 180553, 181127, 181477, 181294, 181007, 180409,
180162, 180114), c(332349, 332057, 332342, 333250, 333558, 333676,
332618, 332413, 332349))
r2 = cbind(c(180042, 180545, 180553, 180314, 179955, 179142, 179437,
179524, 179979, 180042), c(332373, 332026, 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))

sr1=Polygons(list(Polygon(r1)),"r1")
sr2=Polygons(list(Polygon(r2)),"r2")
sr3=Polygons(list(Polygon(r3)),"r3")
sr=SpatialPolygons(list(sr1,sr2,sr3))
srdf=SpatialPolygonsDataFrame(sr, data.frame(cbind(1:3,5:3), row.names=c("r1","r2","r3")))

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

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

overlay(srdf, meuse)
overlay(sr, meuse)

overlay(meuse, srdf, fn = mean)
overlay(meuse, srdf)
overlay(as(meuse, "SpatialPoints"), srdf)
overlay(as(meuse, "SpatialPoints"), sr)

# same thing, with grid:
overlay(srdf, meuse.grid)
overlay(sr, meuse.grid)

overlay(meuse.grid, srdf, fn = mean)
```

```
overlay(meuse.grid, srdf)
overlay(as(meuse.grid, "SpatialPoints"), srdf)
overlay(as(meuse.grid, "SpatialPoints"), sr)
```

`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, ...)
sp.grid(obj, col = 1, alpha = 1,...)
sp.text(loc, txt, ...)
```

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
`loc`	numeric vector of two elements
`txt`	text to be plotted
`alpha`	alpha (transparency) level
`...`	arguments passed to the underlying 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.

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

`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 J. Pebesma, ⟨e.pebesma@geo.uu.nl⟩

References

http://r-spatial.sourceforge.net/ has a graph gallery with examples with R code.

See Also

spplot, spplot-methods

Examples

`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)
```

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

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(), in gstat file polygon.c; 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))
```

---

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

---

polygons *sets spatial coordinates to create spatial data, or retrieves spatial coordinates*

Description

sets spatial coordinates to create spatial data, or retrieves spatial coordinates

Usage

```
polygons(obj)
polygons(object) <- value
```

Arguments

obj	object of class "SpatialPolygons" or "SpatialPolygonsDataFrame"
object	object of class "data.frame"
value	object of class "SpatialPolygons"

Value

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

Examples

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

---

recenter-methods *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 prefered, 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

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

---

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)
```

---

spDistsN1 *Euclidean or Great Circle distance between points*

Description

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

Usage

```
spDistsN1(pts, pt, longlat = FALSE)
```

Arguments

pts	A matrix of 2D points
pt	A single 2D point
longlat	if FALSE, Euclidean distance, if TRUE Great Circle distance

Value

A numeric vector of distances in the metric of the points if longlat=FALSE, or in kilometers if longlat=TRUE

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

References

http://home.att.net/~srschmitt/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]
```

---

spplot *Plot methods for spatial data with attributes*

Description

Lattice (trellis) plot methods for spatial data with attributes

Usage

```
spplot(obj, ...)
spplot.grid(obj, zcol = names(obj), ..., names.attr,
        scales = list(draw = FALSE), xlab = NULL, ylab = NULL, aspect = mapasp(o
        panel = panel.gridplot, sp.layout = NULL, formula, xlim = bbox(obj)[1, ]
        ylim = bbox(obj)[2, ], checkEmptyRC = TRUE)
spplot.polygons(obj, zcol = names(obj), ..., names.attr,
        scales = list(draw = FALSE), xlab = NULL, ylab = NULL, aspect = mapasp(o
        panel = panel.polygonsplot, sp.layout = NULL, formula, xlim = bbox(obj)[
        ylim = bbox(obj)[2, ])
spplot.points(obj, zcol = names(obj), ..., names.attr,
        scales = list(draw = FALSE), xlab = NULL, ylab = NULL, aspect = mapasp(o
        panel = panel.pointsplot, sp.layout = NULL, identify = FALSE, formula,
        xlim = bbexpand(bbox(obj)[1, ], 0.04), ylim = bbexpand(bbox(obj)[2, ], 0
mapLegendGrob(obj, widths = unit(1, "cm"), heights = unit(1, "cm"),
        fill = "black", just = "right")
sp.theme()
layout.north.arrow()
layout.scale.bar(height = 0.05)
spplot.locator(n = 512, type = "n", ...)
```

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 xyplot (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 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(ol 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
xlim	numeric; x-axis limits
ylim	numeric; y-axis limits
widths	width of grob
heights	heights of grob
fill	fill color of grob
just	grob placement justification
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.

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.

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` for the layout of panels

`col.regions` to specify fill colours

`pretty` for colour breaks at pretty numbers

`at` to specify at which values colours change

`as.table` to 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 and levelplot.

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

`key.space` character: "bottom", "right", "left" or "right" to denote key location, or list: see argument key in the help for xyplot what the options are

`legendEntries` character; array with key legend (text) entries; suitable defaults obtained from data

`cuts` number of cuts or 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 if `cuts` is only number of cuts

`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 a layout items. A layout item is a list with its first argument 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.

A special layout list item is `which` (integer), to control to which panel a layout item should be added. If `which` is present in the main, top-level list it applies for all layout items; in sub-lists with layout items it denotes the (set of) panels in which the layout item should be drawn. Without a `which` item, layout items are drawn in each panel.

The order of items in `sp.layout` matters; objects are drawn in the order they appear. Plot order and prevalence of sp.layout items: for points and lines, sp.layout items are drawn before the points (to allow for grids and polygons); for grids and polygons sp.layout is drawn afterwards (so the item will not be overdrawn by the grid and/or polygon). Although a matter of taste, transparency may help when combining things.

`sp.theme` returns a lattice theme; use `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.

Author(s)

Edzer J. Pebesma, ⟨e.pebesma@geo.uu.nl⟩

References

http://r-spatial.sourceforge.net/

See Also

Examples

```
library(lattice)
trellis.par.set(sp.theme()) # sets bpy.colors() ramp
data(meuse)
coordinates(meuse) <- ~x+y
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)))

if (require(RColorBrewer)) {
        spplot(meuse, c("ffreq"), sp.layout=list(l2,l3,l4,l5),
                col.regions=brewer.pal(3, "Set1"))
}
```

`spsample` *sample point locations in (or on) a spatial object*

Description

sample point locations within a square area, a grid, a polygon, or on a spatial line, using regular or random sampling methods

Usage

```
spsample(x, n, type, ...)
sample.Spatial(x, n, type, bb = bbox(x), offset = runif(nrow(bb)), cellsize, ...
sample.Line(x, n, type, offset = runif(1), proj4string=CRS(as.character(NA)), ..
sample.Polygon(x, n, type = "random", bb = bbox(x), offset = runif(2), proj4stri
sample.Polygons(x, n, type = "random", bb = bbox(x), offset = runif(2), proj4str
sample.Sgrid(x, n, type = "random", bb = bbox(x), offset = runif(nrow(bb)), ...)
makegrid(x, n = 10000, nsig = 2, cellsize, offset = rep(0.5, nrow(bb)))
```

Arguments

x	Spatial object; `spsample(x,...)` is a generic method for the existing `sample.Xxx` fumctions
...	optional arguments, passed to the appropriate `sample.Xxx` functions
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"); or `"nonaligned"` for nonaligned systematic sampling (nx random y coordinates, ny random x coordinates)
bb	bounding box of the sampled domain; setting this to a smaller value leads to sub-region sampling
offset	for regular sampling only: the offset (position) of the regular grid; the default for `spsample` methods is a random location in the unit cell $[0,1] \times [0,1]$, 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")
cellsize	if missing, a cell size is derived from the sample size `n`; otherwise, this cell size is used for all sampling methods except `"random"`
proj4string	Object of class `"CRS"`; holding a valid proj4 string
nsig	for "pretty" coordinates; `spsample` does not result in pretty grids
iter	default = 4: number of times to try to place sample points in a polygon before giving up and returning 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

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 may be necessary to select 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, deriving cell size from the number of grid points requested (approximating the number of cells).

Methods

x = "Spatial" sample in the bbox of x

x = "Line" sample on a line

x = "Polygon" sample in an Polygon

x = "Polygons" sample in an Polygons object, consisting of possibly multiple Polygon objects (and holes!)

x = "SpatialPolygons" sample in an SpatialPolygons object; sampling takes place over all Polygons objects present, use subsetting to vary sampling intensity (density)

x = "SpatialGrid" sample in an SpatialGrid object

x = "SpatialPixels" sample in an SpatialPixels object

Note

If an 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 type="random", the number searched for is inflated to ensure hits, and the points returned sampled among these.

Author(s)

Edzer J. Pebesma, ⟨e.pebesma@geo.uu.nl⟩

References

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

See Also

overlay-methods, point.in.polygon, 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)
```

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

Description

rearrange SpatialPointsDataFrame for plotting with spplot or levelplot

Usage

```
spmap.to.lev(data, zcol = 1:n, n = 2, names.attr)
stack.SpatialPointsDataFrame(x, select, ...)
```

Arguments

`data`	object of class (or extending) SpatialDataFrame
`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

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 return 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

```
library(lattice)
data(meuse.grid) # data frame
coordinates(meuse.grid) = c("x", "y") # promotes to SpatialDataFrame
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")
```

`transform-methods` *place holder for transform methods in library spproj*

Description

provide useful error message if library spproj is not loaded

Methods

obj = "Spatial" tries to load library spproj and call the appropriate transform method present there; if this fails, an error message results.

`zerodist` *find point pairs with equal spatial coordinates*

Description

find point pairs with equal spatial coordinates

Usage

```
zerodist(obj, zero = 0.0)
remove.duplicates(obj, zero = 0.0)
```

Arguments

`obj`	object of, or extending, class SpatialPoints
`zero`	value to be compared to for establishing when a distance is considered zero (default 0.0)

Value

pairs of row numbers with identical coordinates, numeric(0) if no such pairs are found

Note

When using kriging, duplicate observations sharing identical spatial locations result in singular covariance matrices in kriging situations. This function may help identifying spatial duplications, so they can be removed. A matrix with all pair-wise distances is calculated, so if x, y and z are large this function is slow

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)
```

Index