S Classes and Methods for Spatial Data: the `sp` Package

Edzer Pebesma* Roger S. Bivand†

Feb 2005

Contents

*Institute for Geoinformatics, University of Muenster, Weseler Strasse 253, 48151 Münster, Germany. `edzer.pebesma@uni-muenster.de`

†Economic Geography Section, Department of Economics, Norwegian School of Economics and Business Administration, Breiviksveien 40, N-5045 Bergen, Norway; `Roger.Bivand@nhh.no`

1 Introduction

The sp package provides classes and methods for dealing with spatial data in S (R and S-Plus[1]). The spatial data structures implemented include points, lines, polygons and grids; each of them with or without attribute data. We have chosen to use S4 classes and methods style (Chambers, 1998) to allow validation of objects created. Although we mainly aim at using spatial data in the geographical (two-dimensional) domain, the data structures that have a straightforward implementation in higher dimensions (points, grids) do allow this.

The motivation to write this package was born on a pre-conference spatial data workshop during DSC 2003. At that time, the advantage of having multiple R packages for spatial statistics seemed to be hindered by a lack of a uniform interface for handling spatial data. Each package had its own conventions on how spatial data were stored and returned. With this package, and packages supporting the classes provided here, we hope that R will become a more coherent tool for analyzing different types of spatial data.

The package is available, or will be available soon on CRAN. From the package home page, http://r-spatial.sourceforge.net/, a graph gallery with R code, and the development source tree are available.

This vignette describes the classes, methods and functions provided by sp. Instead of manipulating the class slots (components) directly, we provide methods and functions to create the classes from elementary types such as matrices, data.frames or lists and to convert them back to any of these types. Also, coercion (type casting) from one class to the other is provided, where relevant.

Package sp is loaded by

```
> library(sp)
```

2 Spatial data classes

The spatial data classes implemented are points, grids, lines, rings and polygons. Package sp provides classes for the spatial-only information (the topology), e.g. SpatialPoints, and extensions for the case where we attribute information stored in a data.frame is available for each spatial entity (e.g. for points, the SpatialPointsDataFrame). The available data classes are:

[1] our primary efforts target R; depending on the needs, we will address S-Plus as well

data type	class	attributes	contains
points	`SpatialPoints`	No	`Spatial`*
points	`SpatialPointsDataFrame`	`data.frame`	`SpatialPoints`*
pixels	`SpatialPixels`	No	`SpatialPoints`*
pixels	`SpatialPixelsDataFrame`	`data.frame`	`SpatialPixels`* `SpatialPointsDataFrame`**
full grid	`SpatialGrid`	No	`SpatialPixels`*
full grid	`SpatialGridDataFrame`	`data.frame`	`SpatialGrid`*
line	`Line`	No	
lines	`Lines`	No	`Line` list
lines	`SpatialLines`	No	`Spatial`*, `Lines` list
lines	`SpatialLinesDataFrame`	`data.frame`	`SpatialLines`*
rings	`Polygon`	No	`Line`*
rings	`Polygons`	No	`Polygon` list
rings	`SpatialPolygons`	No	`Spatial`*, `Polygons` list
rings	`SpatialPolygonsDataFrame`	`data.frame`	`SpatialPolygons`*

* by direct extension; ** by setIs() relationship;

The class `Spatial` does never hold actual data, it only provides the information common to all derived classes: the spatial coordinates bounding box and information about the coordinate reference system (geographic projection information).

In the following sections we will show how we can create objects of these classes from scratch or from other classes, and which methods and functions are available for them.

3 Manipulating spatial objects

Although entries in spatial objects are in principle accessible through their slot name, e.g. `x@coords` contains the coordinates of an object of class or extending `SpatialPoints`, we strongly encourage users to access the data by using functions and methods, in this case `coordinates(x)` to retrieve the coordinates.

3.1 Standard methods

Selecting, retrieving or replacing certain attributes in spatial objects with attributes is done using standard methods

- `[` select "rows" (items) and/or columns in the data attribute table; e.g. `meuse[1:2, "zinc"]` returns a `SpatialPointsDataFrame` with the first two points and an attribute table with only variable "zinc".
- `[[` select a column from the data attribute table
- `[[<-` assign or replace values to a column in the data attribute table.

Other methods available are: `plot`, `summary`, `print`, `dim` and `names` (operate on the data.frame part), `as.data.frame`, `as.matrix` and `image` (for gridded data), `lines` (for line data), `points` (for point data), `subset` (points and grids), `stack` (point and grid data.frames), and `length` (number of features).

3.2 Spatial methods

A number of spatial methods are available for the classes in `sp`:

- `dimensions(x)` returns number of spatial dimensions
- `y = spTransform(x, CRS("+proj=longlat +datum=WGS84"))` transform from one coordinate reference system (geographic projection) to another (requires package rgdal)
- `bbox(x)` returns a matrix with the coordinates bounding box; the dimensions form rows, min/max form the columns
- `coordinates(x)` returns a matrix with the spatial coordinates
- `gridded(x)` tells whether `x` derives from SpatialPixels
- `spplot(x)` plot attributes, possibly in combination with other types of data (points, lines, grids, polygons), and possibly in as a conditioning plot for multiple attributes
- `overlay(x, y)` combine two spatial layers of different type, e.g. retrieve the polygon or grid indexes on a set of points.
- `spsample(x)` sampling of spatial points in continuous space within a polygon, a gridded area, or on a spatial line. Subsetting and `sample` can be used to subsample full spatial entities.

4 Spatial points

4.1 Points without attributes

We can generate a set of 10 points on the unit square $[0, 1] \times [0, 1]$ by

```
> xc = round(runif(10), 2)
> yc = round(runif(10), 2)
> xy = cbind(xc, yc)
> xy
```

```
       xc   yc
 [1,] 0.67 0.30
 [2,] 0.96 0.62
 [3,] 0.92 0.91
 [4,] 0.77 0.85
```

```
 [5,] 0.72 0.46
 [6,] 0.74 0.39
 [7,] 0.63 0.64
 [8,] 0.19 0.72
 [9,] 0.70 0.20
[10,] 0.37 0.28
```

this 10×2 matrix can be converted into a `SpatialPoints` object by

```
> xy.sp = SpatialPoints(xy)
> xy.sp

SpatialPoints:
        xc   yc
 [1,] 0.67 0.30
 [2,] 0.96 0.62
 [3,] 0.92 0.91
 [4,] 0.77 0.85
 [5,] 0.72 0.46
 [6,] 0.74 0.39
 [7,] 0.63 0.64
 [8,] 0.19 0.72
 [9,] 0.70 0.20
[10,] 0.37 0.28
Coordinate Reference System (CRS) arguments: NA

> plot(xy.sp, pch = 2)
```

The plot is shown in figure 1.

We can retrieve the coordinates from `xy.sp` by

```
> xy.cc = coordinates(xy.sp)
> class(xy.cc)

[1] "matrix"

> dim(xy.cc)

[1] 10  2
```

and other methods retrieve the bounding box, the dimensions, select points (not dimensions or columns), coerce to a data.frame, or print a summary:

```
> bbox(xy.sp)

    min  max
xc 0.19 0.96
yc 0.20 0.91
```

Figure 1: plot of `SpatialPoints` object; aspect ratio of x and y axis units is 1

```
> dimensions(xy.sp)

[1] 2

> xy.sp[1:2]

SpatialPoints:
       xc   yc
[1,] 0.67 0.30
[2,] 0.96 0.62
Coordinate Reference System (CRS) arguments: NA

> xy.df = as.data.frame(xy.sp)
> class(xy.df)

[1] "data.frame"

> dim(xy.df)

[1] 10  2

> summary(xy.sp)

Object of class SpatialPoints
Coordinates:
    min  max
xc 0.19 0.96
yc 0.20 0.91
Is projected: NA
proj4string : [NA]
Number of points: 10
```

4.2 Points with attributes

One way of creating a `SpatialPointsDataFrame` object is by building it from a a `SpatialPoints` object and a data.frame containing the attributes:

```
> df = data.frame(z1 = round(5 + rnorm(10), 2), z2 = 20:29)
> df

     z1 z2
1  3.10 20
2  4.15 21
3  3.68 22
4  4.45 23
5  6.62 24
6  5.57 25
7  3.66 26
8  3.75 27
9  5.19 28
10 5.02 29
```

```
> xy.spdf = SpatialPointsDataFrame(xy.sp, df)
> xy.spdf

    coordinates   z1 z2
1   (0.67, 0.3) 3.10 20
2  (0.96, 0.62) 4.15 21
3  (0.92, 0.91) 3.68 22
4  (0.77, 0.85) 4.45 23
5  (0.72, 0.46) 6.62 24
6  (0.74, 0.39) 5.57 25
7  (0.63, 0.64) 3.66 26
8  (0.19, 0.72) 3.75 27
9    (0.7, 0.2) 5.19 28
10 (0.37, 0.28) 5.02 29

> summary(xy.spdf)

Object of class SpatialPointsDataFrame
Coordinates:
    min  max
xc 0.19 0.96
yc 0.20 0.91
Is projected: NA
proj4string : [NA]
Number of points: 10
Data attributes:
       z1              z2
 Min.   :3.100   Min.   :20.00
 1st Qu.:3.697   1st Qu.:22.25
 Median :4.300   Median :24.50
 Mean   :4.519   Mean   :24.50
 3rd Qu.:5.147   3rd Qu.:26.75
 Max.   :6.620   Max.   :29.00

> dimensions(xy.spdf)

[1] 2

> xy.spdf[1:2, ] # selects row 1 and 2

   coordinates   z1 z2
1  (0.67, 0.3) 3.10 20
2 (0.96, 0.62) 4.15 21

> xy.spdf[1] # selects attribute column 1, along with the coordinates

    coordinates   z1
1   (0.67, 0.3) 3.10
```

```
2  (0.96, 0.62) 4.15
3  (0.92, 0.91) 3.68
4  (0.77, 0.85) 4.45
5  (0.72, 0.46) 6.62
6  (0.74, 0.39) 5.57
7  (0.63, 0.64) 3.66
8  (0.19, 0.72) 3.75
9    (0.7, 0.2) 5.19
10 (0.37, 0.28) 5.02
```

```
> xy.spdf[1:2, "z2"] # select row 1,2 and attribute "z2"
```

```
   coordinates z2
1  (0.67, 0.3) 20
2 (0.96, 0.62) 21
```

```
> xy.df = as.data.frame(xy.spdf)
> xy.df[1:2,]
```

```
    z1 z2   xc   yc
1 3.10 20 0.67 0.30
2 4.15 21 0.96 0.62
```

```
> xy.cc = coordinates(xy.spdf)
> class(xy.cc)
```

```
[1] "matrix"
```

```
> dim(xy.cc)
```

```
[1] 10  2
```

A note on selection with `[`: the behaviour is as much as possible copied from that of `data.frame`s, but coordinates are always sticky and allways a `SpatialPointsDataFrame` is returned; `drop=FALSE` is not allowed. If coordinates should be dropped, use the `as.data.frame` method and select the non-coordinate data, or use `[[` to select a single attribute column (example below).

`SpatialPointsDataFrame` objects can be created directly from data.frames by specifying which columns contain the coordinates:

```
> df1 = data.frame(xy, df)
> coordinates(df1) = c("xc", "yc")
> df1
```

```
    coordinates   z1 z2
1   (0.67, 0.3) 3.10 20
2  (0.96, 0.62) 4.15 21
3  (0.92, 0.91) 3.68 22
```

```
4  (0.77, 0.85) 4.45 23
5  (0.72, 0.46) 6.62 24
6  (0.74, 0.39) 5.57 25
7  (0.63, 0.64) 3.66 26
8  (0.19, 0.72) 3.75 27
9    (0.7, 0.2) 5.19 28
10 (0.37, 0.28) 5.02 29
```

or

```
> df2 = data.frame(xy, df)
> coordinates(df2) = ~xc+yc
> df2[1:2,]

   coordinates   z1 z2
1  (0.67, 0.3) 3.10 20
2 (0.96, 0.62) 4.15 21

> as.data.frame(df2)[1:2,]

    xc   yc   z1 z2
1 0.67 0.30 3.10 20
2 0.96 0.62 4.15 21
```

Note that in this form, `coordinates` by setting (specifying) the coordinates promotes its argument, an object of class `data.frame` to an object of class `SpatialPointsDataFrame`. The method `as.data.frame` coerces back to the original `data.frame`. When used on a right-hand side of an equation, `coordinates` *retrieves* the matrix with coordinates:

```
> coordinates(df2)[1:2,]

       xc   yc
[1,] 0.67 0.30
[2,] 0.96 0.62
```

Elements (columns) in the data.frame part of an object can be manipulated (retrieved, assigned) directly:

```
> df2[["z2"]]

 [1] 20 21 22 23 24 25 26 27 28 29

> df2[["z2"]][10] = 20
> df2[["z3"]] = 1:10
> summary(df2)
```

```
Object of class SpatialPointsDataFrame
Coordinates:
    min  max
xc 0.19 0.96
yc 0.20 0.91
Is projected: NA
proj4string : [NA]
Number of points: 10
Data attributes:
       z1              z2              z3
 Min.   :3.100   Min.   :20.00   Min.   : 1.00
 1st Qu.:3.697   1st Qu.:21.25   1st Qu.: 3.25
 Median :4.300   Median :23.50   Median : 5.50
 Mean   :4.519   Mean   :23.60   Mean   : 5.50
 3rd Qu.:5.147   3rd Qu.:25.75   3rd Qu.: 7.75
 Max.   :6.620   Max.   :28.00   Max.   :10.00
```

Plotting attribute data can be done by using either `spplot` to colour symbols, or `bubble` which uses symbol size:

```
> bubble(df2, "z1", key.space = "bottom")
> spplot(df2, "z1", key.space = "bottom")
```

the resulting plots are shown in figure 2.

5 Grids

Package `sp` has two classes for grid topology: `SpatialPixels` and `SpatialGrid`. The pixels form stores coordinates and is for partial grids, or unordered points; the `SpatialGrid` form does not store coordinates but holds full grids (i.e., `SpatialGridDataFrame` holds attribute values for each grid cell). Objects can be coerced from one representation to the other.

5.1 Creating grids from topology

When we know the offset, the cell sizes and the dimensions of a grid, we can specify this by using the function `GridTopology`:

```
> gt = GridTopology(cellcentre.offset = c(1,1,2), cellsize=c(1,1,1), cells.dim = c(3,4,6))
> grd = SpatialGrid(gt)
> summary(grd)
```

```
Object of class SpatialGrid
Coordinates:
     min max
[1,] 0.5 3.5
[2,] 0.5 4.5
```

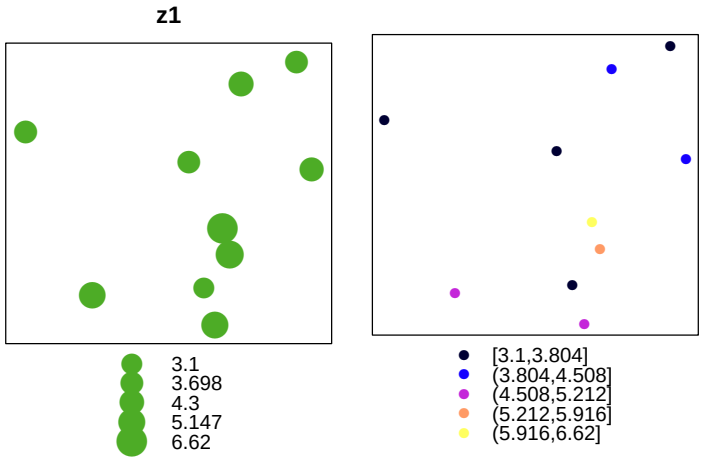


Figure 2: plot of `SpatialPointsDataFrame` object, using symbol size (`bubble`, top) or colour (`spplot`, bottom)

```
[3,] 1.5 7.5
Is projected: NA
proj4string : [NA]
Grid attributes:
  cellcentre.offset cellsize cells.dim
1                 1        1         3
2                 1        1         4
3                 2        1         6
```

The grid parameters can be retrieved by the function

```
> gridparameters(grd)
```

```
  cellcentre.offset cellsize cells.dim
1                 1        1         3
2                 1        1         4
3                 2        1         6
```

5.2 Creating grids from points

In the following example a three-dimensional grid is constructed from a set of point coordinates:

```
> pts = expand.grid(x = 1:3, y = 1:4, z=2:7)
> grd.pts = SpatialPixels(SpatialPoints(pts))
> summary(grd.pts)
```

```
Object of class SpatialPixels
Coordinates:
  min max
x 0.5 3.5
y 0.5 4.5
z 1.5 7.5
Is projected: NA
proj4string : [NA]
Number of points: 72
Grid attributes:
  cellcentre.offset cellsize cells.dim
x                 1        1         3
y                 1        1         4
z                 2        1         6
```

```
> grd = as(grd.pts, "SpatialGrid")
> summary(grd)
```

```
Object of class SpatialGrid
Coordinates:
  min max
```

```
x 0.5 3.5
y 0.5 4.5
z 1.5 7.5
Is projected: NA
proj4string : [NA]
Grid attributes:
  cellcentre.offset cellsize cells.dim
x                 1        1         3
y                 1        1         4
z                 2        1         6
```

Note that when passed a points argument, SpatialPixels accepts a tolerance (default 10 * .Machine$double.eps) to specify how close the points have to be to being exactly on a grid. For very large coordinates, this value may have to be increased. A warning is issued if full rows and/or columns are missing.

5.3 Gridded data with attributes

Spatial, gridded data are data with coordinates on a regular lattice. To form such a grid we can go from coordinates:

```
> attr = expand.grid(xc = 1:3, yc = 1:3)
> grd.attr = data.frame(attr, z1 = 1:9, z2 = 9:1)
> coordinates(grd.attr) = ~xc+yc
> gridded(grd.attr)

[1] FALSE

> gridded(grd.attr) = TRUE
> gridded(grd.attr)

[1] TRUE

> summary(grd.attr)

Object of class SpatialPixelsDataFrame
Coordinates:
   min max
xc 0.5 3.5
yc 0.5 3.5
Is projected: NA
proj4string : [NA]
Number of points: 9
Grid attributes:
   cellcentre.offset cellsize cells.dim
xc                 1        1         3
yc                 1        1         3
Data attributes:
```

```
       z1          z2
 Min.   :1   Min.   :1
 1st Qu.:3   1st Qu.:3
 Median :5   Median :5
 Mean   :5   Mean   :5
 3rd Qu.:7   3rd Qu.:7
 Max.   :9   Max.   :9
```

5.4 Are grids stored as points or as matrix/array?

The form in which gridded data comes depends on whether the grid was created from a set of points or from a matrix or external grid format (e.g. read through rgdal). Retrieving the form, or conversion to another can be done by `as(x, "Class")`, or by using the function `fullgrid`:

```
> fullgrid(grd)

[1] TRUE

> fullgrid(grd.pts)

[1] FALSE

> fullgrid(grd.attr)

[1] FALSE

> fullgrid(grd.pts) = TRUE
> fullgrid(grd.attr) = TRUE
> fullgrid(grd.pts)

[1] TRUE

> fullgrid(grd.attr)

[1] TRUE
```

The advantage of having grids in cell form is that when a large part of the grid contains missing values, these cells do not have to be stored; also, no ordering of grid cells is required. For plotting by a grid with `levelplot`, this form is required and `spplot` (for grids a front-end to `levelplot`) will convert grids that are not in this form. In contrast, `image` requires a slightly altered version of the the full grid form. A disadvantage of the cell form is that the coordinates for each point have to be stored, which may be prohibitive for large grids. Grids in cell form do have an index to allow for fast transformation to the full grid form.

Besides `print`, `summary`, `plot`, objects of class `SpatialGridDataFrame` have methods for

- `[` select rows (points) or columns (variables)
- `[[` retrieve a column from the attribute table (data.frame part)
- `[[<-` assign or replace a column in the attribute table (data.frame part)
- `coordinates` retrieve the coordinates of grid cells
- `as.matrix` retrieve the data as a matrix. The first index (rows) is the x-column, the second index (columns) the y-coordinate. Row index 1 is the smallest x-coordinate; column index 1 is the larges y-coordinate (top-to-bottom).
- `as` coercion methods for `data.frame`, `SpatialPointsDataFrame`
- `image` plot an image of the grid

Finally, `spplot`, a front-end to `levelplot` allows the plotting of a single grid plot or a lattice of grid plots.

5.5 Row and column selection of a region

Rows/columns selection can be done when gridded data is in the full grid form (as `SpatialGridDataFrame`). In this form also rows and/or columns can be de-selected (in which case a warning is issued):

```
> fullgrid(grd.attr) = FALSE
> grd.attr[1:5, "z1"]

Object of class SpatialPixelsDataFrame
Object of class SpatialPixels
Grid topology:
   cellcentre.offset cellsize cells.dim
xc                 1        1         3
yc                 1        1         3
SpatialPoints:
     xc yc
[1,]  1  3
[2,]  2  3
[3,]  3  3
[4,]  1  2
[5,]  2  2
Coordinate Reference System (CRS) arguments: NA

Data summary:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    4.0     5.0     7.0     6.6     8.0     9.0

> fullgrid(grd.attr) = TRUE
> grd.attr[1:2,-2, c("z2","z1")]
```

```
Object of class SpatialGridDataFrame
Object of class SpatialGrid
Grid topology:
   cellcentre.offset cellsize cells.dim
xc                 1        2         2
yc                 2        1         2
SpatialPoints:
     xc yc
[1,]  1  3
[2,]  3  3
[3,]  1  2
[4,]  3  2
Coordinate Reference System (CRS) arguments: NA

Data summary:
       z2            z1
 Min.   :1.0   Min.   :4.0
 1st Qu.:2.5   1st Qu.:5.5
 Median :3.5   Median :6.5
 Mean   :3.5   Mean   :6.5
 3rd Qu.:4.5   3rd Qu.:7.5
 Max.   :6.0   Max.   :9.0
```

6 Lines

6.1 Building line objects from scratch

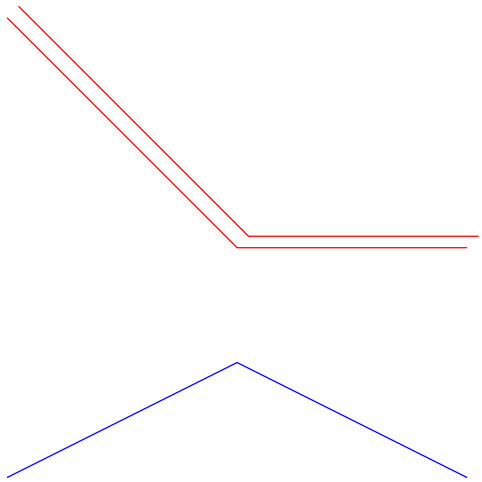
In many instances, line coordinates will be retrieved from external sources. The following example shows how to build an object of class `SpatialLines` from scratch. Note that the `Lines` objects have to be given character ID values, and that these values must be unique for `Lines` objects combined in a `SpatialLines` object.

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

Object of class SpatialLines
Coordinates:
```

```
  min  max
x   1 3.05
y   1 3.05
Is projected: NA
proj4string : [NA]
```

```
> plot(Sl, col = c("red", "blue"))
```



6.2 Building line objects with attributes

The class `SpatialLinesDataFrame` is designed for holding lines data that have an attribute table (data.frame) attached to it:

```
> df = data.frame(z = c(1,2), row.names=sapply(slot(Sl, "lines"), function(x) slot(x, "ID"))
> Sldf = SpatialLinesDataFrame(Sl, data = df)
> summary(Sldf)
```

```
Object of class SpatialLinesDataFrame
Coordinates:
  min  max
x   1 3.05
y   1 3.05
Is projected: NA
```

```
proj4string : [NA]
Data attributes:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    1.25    1.50    1.50    1.75    2.00
```

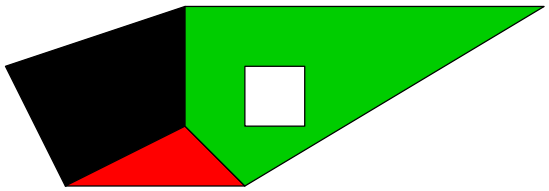
Not many useful methods for it are available yet. The `plot` method only plots the lines, ignoring attribute table values. Suggestions for useful methods are welcome.

7 Polygons

7.1 Building from scratch

The following example shows how a set of polygons are built from scratch. Note that `Sr4` has the opposite direction (anti-clockwise) as the other three (clockwise); it is meant to represent a hole in the `Sr3` polygon. The default value for the hole colour `pbg` is `"transparent`, which will not show, but which often does not matter, because another polygon fills the hole — here it is set to `"white"`. Note that the `Polygons` objects have to be given character ID values, and that these values must be unique for `Polygons` objects combined in a `SpatialPolygons` object.

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



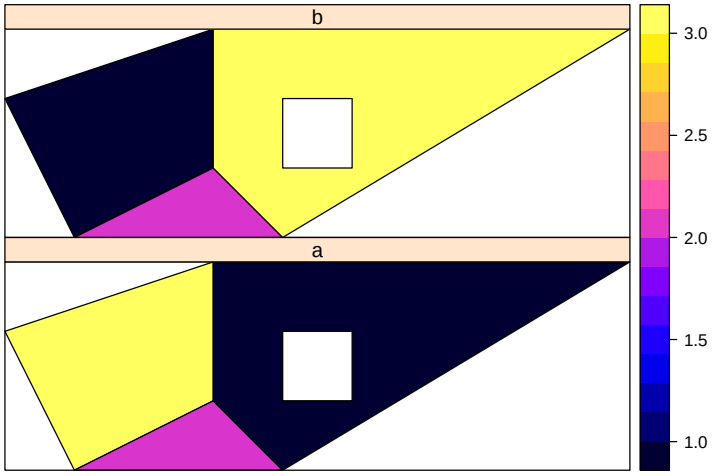
7.2 Polygons with attributes

Polygons with attributes, objects of class `SpatialPolygonsDataFrame`, are built from the `SpatialPolygons` object (topology) and the attributes (data.frame). The `row.names` of the attributes data frame are matched with the ID slots of the `SpatialPolygons` object, and the rows of the data frame will be re-ordered if necessary.

```
> attr = data.frame(a=1:3, b=3:1, row.names=c("s3/4", "s2", "s1"))
> SrDf = SpatialPolygonsDataFrame(SpP, attr)
> as(SrDf, "data.frame")

     a b
s1   3 1
s2   2 2
s3/4 1 3

> spplot(SrDf)
```



or, as another way to create the `SpatialPolygonsDataFrame` object:

```
> SrDf = attr
> polygons(SrDf) = SpP
```

8 Importing and exporting data

Data import and export from external data sources and file formats is handled inthe **rgdal** package in the first instance, using the available OGR/GDAL drivers for vector and raster data. This keeps the range of drivers up to date, and secures code maintenance through working closely with the open source geospatial community. Mac OSX users unable or unwilling to install **rgdal** from source after installing its external dependencies will find some functions in the **maptools** package to import and export a limited range of formats.

References

Chambers, J.M., 1998, Programming with data, a guide to the S language. Springer, New York.