Analysing Spatial Data in R: Representing Spatial Data

Roger Bivand

Department of Economics
Norwegian School of Economics and Business Administration
Bergen, Norway

31 August 2007
Object framework

- To begin with, all contributed packages for handling spatial data in R had different representations of the data.
- This made it difficult to exchange data both within R between packages, and between R and external file formats and applications.
- The result has been an attempt to develop shared classes to represent spatial data in R, allowing some shared methods and many-to-one, one-to-many conversions.
- We chose to use new-style classes to represent spatial data, and are confident that this choice was justified.
Spatial objects

- The foundation object is the `Spatial` class, with just two slots (new-style class objects have pre-defined components called slots)
- The first is a bounding box, and is mostly used for setting up plots
- The second is a `CRS` class object defining the coordinate reference system, and may be set to `CRS(as.character(NA))`, its default value.
- Operations on `Spatial*` objects should update or copy these values to the new `Spatial*` objects being created
Spatial points

- The most basic spatial data object is a point, which may have 2 or 3 dimensions.
- A single coordinate, or a set of such coordinates, may be used to define a SpatialPoints object; coordinates should be of mode double and will be promoted if not.
- The points in a SpatialPoints object may be associated with a row of attributes to create a SpatialPointsDataFrame object.
- The coordinates and attributes may, but do not have to be keyed to each other using ID values.
Spatial points

Using the Meuse bank data set of soil samples and measurements of heavy metal pollution provided with sp, we’ll make a SpatialPoints object.

```r
> library(sp)
> data(meuse)
> coords <- SpatialPoints(meuse[, c("x", "y")])
> summary(coords)

Object of class SpatialPoints
Coordinates:
   min     max
x 178605 181390
y 329714 333611
Is projected: NA
proj4string : [NA]
Number of points: 155
```
Spatial points

Now we’ll add the original data frame to make a SpatialPointsDataFrame object. Many methods for standard data frames just work with SpatialPointsDataFrame objects.

```r
> meuse1 <- SpatialPointsDataFrame(coords, meuse)
> names(meuse1)

[1] "x"    "y"    "cadmium" "copper" "lead"    "zinc"
[7] "elev" "dist" "om"    "ffreq" "soil"    "lime"
[13] "landuse" "dist.m"

> summary(meuse1$zinc)

    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.     
      113.0       198.0       326.0      469.7       674.5      1839.0

> stem(meuse1$zinc, scale = 1/2)

The decimal point is 2 digit(s) to the right of the |

    0 | 122233334444455556666777788888999999999
    2 | 00000001111111222222334445555666678888800022334455788
    4 | 000122356770014555556789
    6 | 011446788900124556788889
    8 | 01331113
   10 | 235604469
   12 | 8
   14 | 5357
   16 | 7
   18 | 4
```
Spatial points classes and their slots

SpatialPointsDataFrame

SpatialPoints

coords

cords.nrs

data

data.frame

Spatial

bbox

proj4string

SpatialPoints

coords

Spatial
Spatial lines and polygons

- A Line object is just a spaghetti collection of 2D coordinates; a Polygon object is a Line object with equal first and last coordinates
- A Lines object is a list of Line objects, such as all the contours at a single elevation; the same relationship holds between a Polygons object and a list of Polygon objects, such as islands belonging to the same county
- SpatialLines and SpatialPolygons objects are made using lists of Lines or Polygons objects respectively
- SpatialLinesDataFrame and SpatialPolygonsDataFrame objects are defined using SpatialLines and SpatialPolygons objects and standard data frames, and the ID fields are here required to match the data frame row names
The Meuse bank data set also includes the coordinates of the edge of the river, linked together at the edge of the study area to form a polygon. We can make these coordinates into a SpatialPolygons object:

```r
> data(meuse.riv)
> str(meuse.riv)

num [1:176, 1:2] 182004 182137 182252 182314 182332 ...

> river_polygon <- Polygons(list(Polygon(meuse.riv)), ID = "meuse")
> rivers <- SpatialPolygons(list(river_polygon))
> summary(rivers)

Object of class SpatialPolygons
Coordinates:
    min       max
r1 178304.0 182331.5
r2 325698.5 337684.8
Is projected: NA
proj4string : [NA]
```
Spatial lines

There is a helper function `contourLines2SLDF` to convert the list of contours returned by `contourLines` into a `SpatialLinesDataFrame` object. This example shows how the data slot row names match the ID slot values of the set of `Lines` objects making up the `SpatialLinesDataFrame`, note that some `Lines` objects include multiple `Line` objects:

```r
> library(maptools)
> volcano_sl <- ContourLines2SLDF(contourLines(volcano))
> row.names(slot(volcano_sl, "data"))
[1] "C_1"  "C_2"  "C_3"  "C_4"  "C_5"  "C_6"  "C_7"  "C_8"  "C_9"
[10] "C_10"
> sapply(slot(volcano_sl, "lines"), function(x) slot(x, "+"ID"))
[1] "C_1"  "C_2"  "C_3"  "C_4"  "C_5"  "C_6"  "C_7"  "C_8"  "C_9"
[10] "C_10"
> sapply(slot(volcano_sl, "lines"), function(x) length(slot(x, "+"Lines")))
[1] 3 4 1 1 2 2 3 2 1
> volcano_sl$level
[1] 100 110 120 130 140 150 160 170 180 190
Levels: 100 110 120 130 140 150 160 170 180 190
Spatial Polygons classes and slots

SpatialPolygons
- polygons
- plotOrder
- Spatial

SpatialLines
- lines
- Spatial

Lines
- Lines
- ID

Line
- coords

Polygon
- labpt
- area
- hole
- ringDir
- coords

Polygons
- Polygons
- plotOrder
- labpt
- ID
- area

Spatial
- bbox
- proj4string
There are two representations for data on regular rectangular grids (oriented N-S, E-W): SpatialPixels and SpatialGrid.

SpatialPixels are like SpatialPoints objects, but the coordinates have to be regularly spaced; the coordinates are stored, as are grid indices.

SpatialPixelsDataFrame objects only store attribute data where it is present, but need to store the coordinates and grid indices of those grid cells.

SpatialGridDataFrame objects do not need to store coordinates, because they fill the entire defined grid, but they need to store NA values where attribute values are missing.
Let's make a `SpatialPixelsDataFrame` object for the Meuse bank grid data provided, with regular points at a 40m spacing. The data include soil types, flood frequency classes and distance from the river bank:

```r
> data(meuse.grid)
> coords <- SpatialPixels(SpatialPoints(meuse.grid[, c("x", "y")]))
> meuseg1 <- SpatialPixelsDataFrame(coords, meuse.grid)
> names(meuseg1)
[1] "x"   "y"   "part.a" "part.b" "dist" "soil" "ffreq"
> slot(meuseg1, "grid")

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>cellcentre.offset</td>
<td>178460</td>
<td>329620</td>
</tr>
<tr>
<td>cellsize</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>cells.dim</td>
<td>78</td>
<td>104</td>
</tr>
</tbody>
</table>

> object.size(meuseg1)
[1] 339036
> dim(slot(meuseg1, "data"))
[1] 3103   7
```
Spatial grids

In this case we convert the SpatialPixelsDataFrame object to a SpatialGridDataFrame by making a change in-place. In other contexts, it is much more usual to create the GridTopology object in the grid slot directly, and populate the grid from there, as we’ll see later:

```r
> meuseg2 <- meuseg1
> fullgrid(meuseg2) <- TRUE
> slot(meuseg2, "grid")

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>cellcentre.offset</td>
<td>178460 329620</td>
</tr>
<tr>
<td>cellsize</td>
<td>40</td>
</tr>
<tr>
<td>cells.dim</td>
<td>78</td>
</tr>
</tbody>
</table>

> class(slot(meuseg2, "grid"))

[1] "GridTopology"
attr(,"package")
[1] "sp"

> object.size(meuseg2)

[1] 425684

> dim(slot(meuseg2, "data"))

[1] 8112 7
Spatial grid and pixels classes and their slots

**SpatialGridDataFrame**
- **SpatialGrid**
  - data

**SpatialPixelsDataFrame**
- **SpatialPixels**
  - data

**SpatialGridDataframe**
- **SpatialGrid**
  - grid
  - grid.index
  - **SpatialPoints**

**SpatialPixels**
- **SpatialPoints**
  - grid
  - grid.index
  - **SpatialPoints**

**GridTopology**
- cellcentre.offset
- cellsize
- cells.dim

**Spatial**
- bbox
- proj4string

**SpatialPoints**
- coords
- **Spatial**
This table summarises the classes provided by **sp**, and shows how they build up to the objects of most practical use, the **Spatial*DataFrame** family objects:

<table>
<thead>
<tr>
<th>data type</th>
<th>class</th>
<th>attributes</th>
<th>extends</th>
</tr>
</thead>
<tbody>
<tr>
<td>points</td>
<td>SpatialPoints</td>
<td>none</td>
<td>Spatial</td>
</tr>
<tr>
<td>points</td>
<td>SpatialPointsDataFrame</td>
<td>data.frame</td>
<td>SpatialPoints</td>
</tr>
<tr>
<td>pixels</td>
<td>SpatialPixels</td>
<td>none</td>
<td>SpatialPoints</td>
</tr>
<tr>
<td>pixels</td>
<td>SpatialPixelsDataFrame</td>
<td>data.frame</td>
<td>SpatialPixels</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>full grid</td>
<td>SpatialGrid</td>
<td>none</td>
<td>SpatialPixels</td>
</tr>
<tr>
<td>full grid</td>
<td>SpatialGridDataFrame</td>
<td>data.frame</td>
<td>SpatialGrid</td>
</tr>
<tr>
<td>line</td>
<td>Line</td>
<td>none</td>
<td>Line list</td>
</tr>
<tr>
<td>lines</td>
<td>Lines</td>
<td>none</td>
<td>Line list</td>
</tr>
<tr>
<td>lines</td>
<td>SpatialLines</td>
<td>none</td>
<td>Spatial, Lines list</td>
</tr>
<tr>
<td>lines</td>
<td>SpatialLinesDataFrame</td>
<td>data.frame</td>
<td>SpatialLines</td>
</tr>
<tr>
<td>polygon</td>
<td>Polygon</td>
<td>none</td>
<td>Line</td>
</tr>
<tr>
<td>polygons</td>
<td>Polygons</td>
<td>none</td>
<td>Polygon list</td>
</tr>
<tr>
<td>polygons</td>
<td>SpatialPolygons</td>
<td>none</td>
<td>Spatial, Polygons list</td>
</tr>
<tr>
<td>polygons</td>
<td>SpatialPolygonsDataFrame</td>
<td>data.frame</td>
<td>SpatialPolygons</td>
</tr>
</tbody>
</table>
This table summarises the methods provided by \texttt{sp}:

<table>
<thead>
<tr>
<th>method</th>
<th>what it does</th>
</tr>
</thead>
<tbody>
<tr>
<td>[</td>
<td>select spatial items (points, lines, polygons, or rows/cols from a grid) and/or attributes variables</td>
</tr>
<tr>
<td>$, $&lt;-, [], [[]&lt;-</td>
<td>retrieve, set or add attribute table columns</td>
</tr>
<tr>
<td>\texttt{spsample}</td>
<td>sample points from a set of polygons, on a set of lines or from a gridded area</td>
</tr>
<tr>
<td>\texttt{bbox}</td>
<td>get the bounding box</td>
</tr>
<tr>
<td>\texttt{proj4string}</td>
<td>get or set the projection (coordinate reference system)</td>
</tr>
<tr>
<td>\texttt{coordinates}</td>
<td>set or retrieve coordinates</td>
</tr>
<tr>
<td>\texttt{coerce}</td>
<td>convert from one class to another</td>
</tr>
<tr>
<td>\texttt{overlay}</td>
<td>combine two different spatial objects</td>
</tr>
</tbody>
</table>
Using *Spatial* family objects

- Very often, the user never has to manipulate *Spatial* family objects directly, as we have been doing here, because methods to create them from external data are also provided.

- Because the *Spatial*DataFrame family objects behave in most cases like data frames, most of what we are used to doing with standard data frames just works — like [ or $ (but no merge, etc., yet).

- These objects are very similar to typical representations of the same kinds of objects in geographical information systems, so they do not suit spatial data that is not geographical (like medical imaging) as such.

- They provide a standard base for analysis packages on the one hand, and import and export of data on the other, as well as shared methods, like those for visualisation we turn to now.