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.

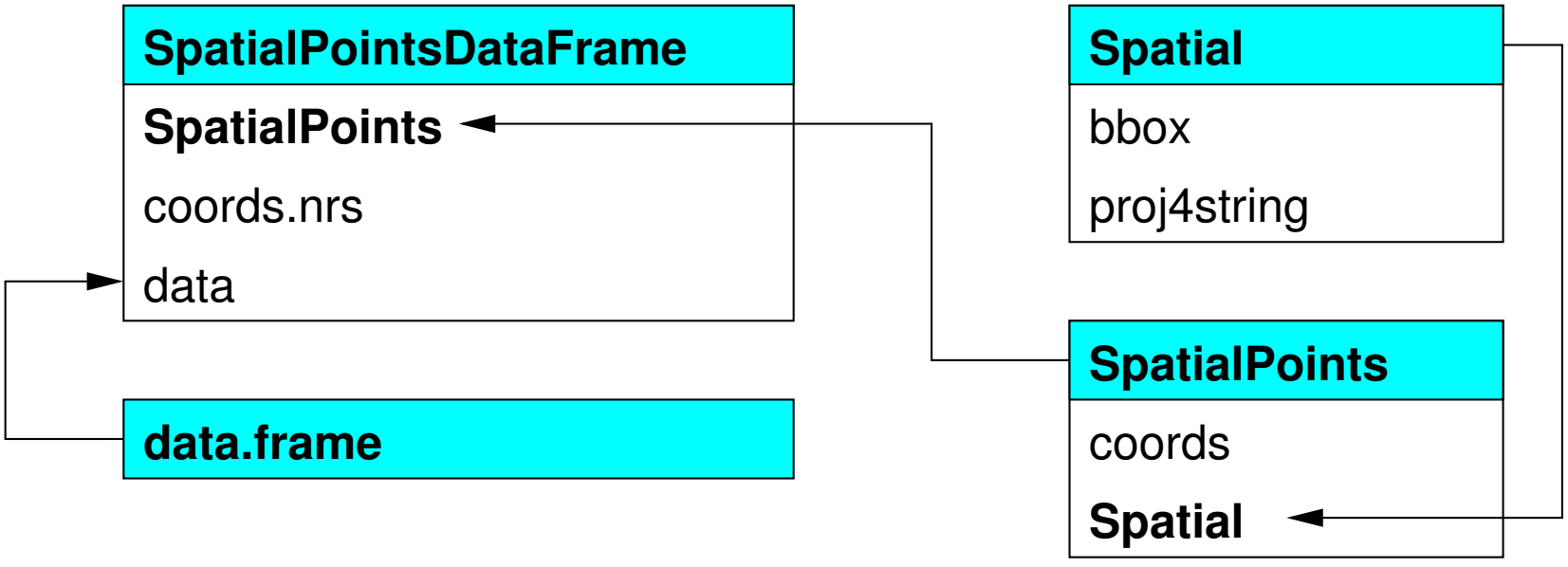
```
> library(sp)
> data(meuse)
> coords <- SpatialPoints(meuse[, c("x", "y")])
> summary(coords)

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.

```
> 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 | 12223333344444455666677778888899999999
   2 | 00000001111111222223344455566667888002233445578 8
   4 | 0001223567700145555678 9
   6 | 01144678890012455678889
   8 | 0133113
  10 | 235604469
  12 | 8
  14 | 5357
  16 | 7
  18 | 4
```

Spatial points classes and their slots



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

Spatial polygons

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:

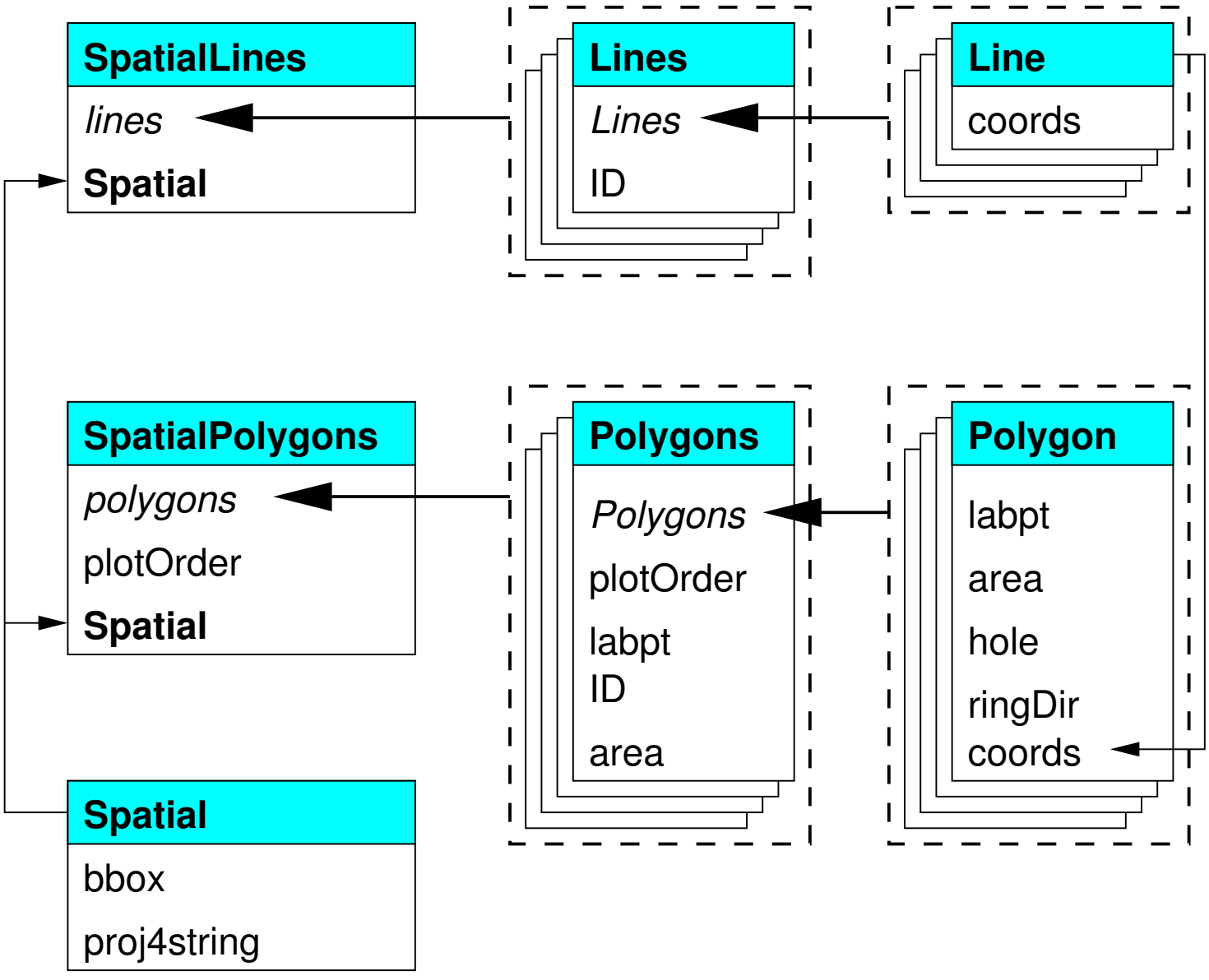
```
> data(meuse.riv)
> str(meuse.riv)

 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:

```
> 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 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



Spatial grids and pixels

- 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

Spatial pixels

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:

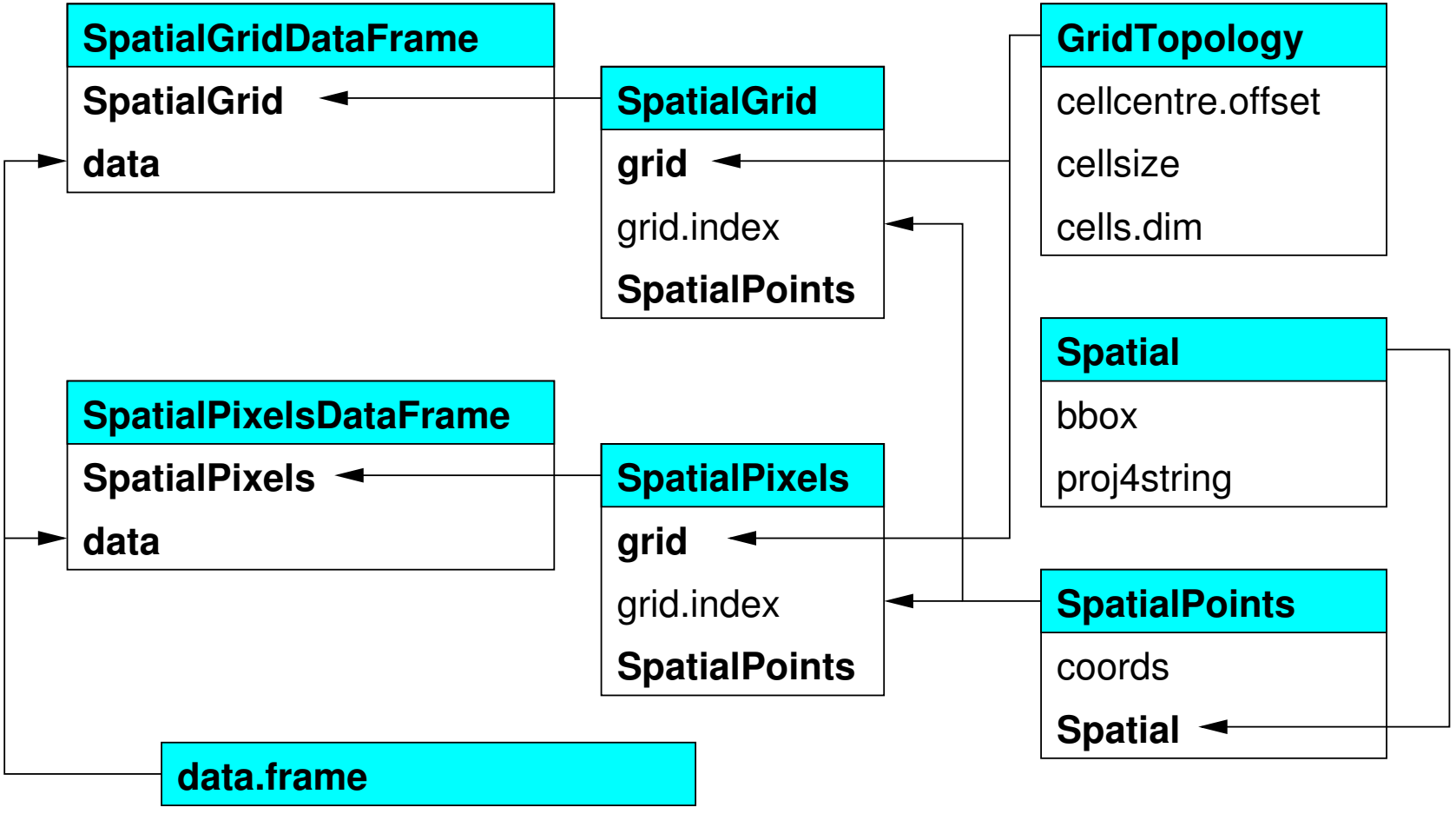
```
> data(meuse.grid)
> coords <- SpatialPixels(SpatialPoints(meuse.grid[, c("x",
+     "y")]))
> meuseg1 <- SpatialPixelsDataFrame(coords, meuse.grid)
> names(meuseg1)

[1] "x"      "y"      "part.a" "part.b" "dist"   "soil"   "ffreq"

> slot(meuseg1, "grid")

                       x      y
cellcentre.offset 178460 329620
cellsize              40     40
cells.dim             78    104

> 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:

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

                       x      y
cellcentre.offset 178460 329620
cellsize              40     40
cells.dim             78    104

> 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



Spatial classes provided by **sp**

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:

data type	class	attributes	extends
points	`SpatialPoints`	none	`Spatial`
points	`SpatialPointsDataFrame`	`data.frame`	`SpatialPoints`
pixels	`SpatialPixels`	none	`SpatialPoints`
pixels	`SpatialPixelsDataFrame`	`data.frame`	`SpatialPixels` `SpatialPointsDataFrame`
full grid	`SpatialGrid`	none	`SpatialPixels`
full grid	`SpatialGridDataFrame`	`data.frame`	`SpatialGrid`
line	`Line`	none	
lines	`Lines`	none	`Line` list
lines	`SpatialLines`	none	`Spatial`, `Lines` list
lines	`SpatialLinesDataFrame`	`data.frame`	`SpatialLines`
polygon	`Polygon`	none	`Line`
polygons	`Polygons`	none	`Polygon` list
polygons	`SpatialPolygons`	none	`Spatial`, `Polygons` list
polygons	`SpatialPolygonsDataFrame`	`data.frame`	`SpatialPolygons`

Methods provided by **sp**

This table summarises the methods provided by **sp**:

method	what it does
`[`	select spatial items (points, lines, polygons, or rows/cols from a grid) and/or attributes variables
`$, $<-, [[, [[<-`	retrieve, set or add attribute table columns
`spsample`	sample points from a set of polygons, on a set of lines or from a gridded area
`bbox`	get the bounding box
`proj4string`	get or set the projection (coordinate reference system)
`coordinates`	set or retrieve coordinates
`coerce`	convert from one class to another
`overlay`	combine two different spatial objects

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