Handling spatial vector data in R

Sytze de Bruin

October 15, 2013

1 Today's learning objectives

In today's lecture, we will explore the basics of handling spatial vector data in R. There are several R packages for this purpose but we will focus on using `sp`, `rgdal`, `rgeos` and some related packages. At the end of the lecture, you should be able to

1. create point, line and polygon objects from scratch;
2. explore the structure of `sp` classes for spatial data;
3. transform between datums and map projections;
4. apply basic operations on vector data, such as buffering, intersection and area calculation;
5. use a Date-Time class;
6. write spatial data to a kml file.
7. convert spatial data read from a plain text file into a spatial class.

2 Some packages for working with spatial vector data in R

The packages `sp` and `rgdal` are widely used throughout this course. Both packages not only provide functionallity for raster data but also for vector data. For example, rgdal includes bindings to parts of the OGR Simple Feature Library which provides access to a variety of vector file formats such as ESRI Shapefiles and kml. Similarly, rgeos is an interface to the powerful Geometry Engine Open Source (GEOS) library for all kind of operations on geometries (buffering, overlaying, area calculations, etc.). Thus, functionality that you commonly find in expensive GIS software is also available within R, using free but very powerful software libaries. The possiblities are huge; in this course we can only scratch the surface with some essentials which hopefully invite you to experiment further and use them in your research. Details can be found in the book Applied Spatial Data Analysis with R and several vignettes authored by Roger Bivand, Edzer Pebesma and Virgilio Gomez-Rubio (Bivand et al., 2013). Owing to time constraints, this lecture cannot cover the related package spacetime with classes and methods for spatio-temporal data.

3 Creating basic geometries

The package sp provides classes for spatial-only geometries, such as `SpatialPoints` (for points), and combinations of geometries and attribute data, such as a `SpatialPoints-DataFrame`. The following data classes are available for spatial vector data (Pebesma & Bivand, 2005):

data type	class	attributes
points	`SpatialPoints`	No
points	`SpatialPointsDataFrame`	`data.frame`
line	`Line`	No
lines	`Lines`	No
lines	`SpatialLines`	No
lines	`SpatialLinesDataFrame`	`data.frame`
rings	`Polygon`	No
rings	`Polygons`	No
rings	`SpatialPolygons`	No
rings	`SpatialPolygonsDataFrame`	`data.frame`

We will go through a few examples of creating geometries from scratch to familiarize yourself with these classes. First, start Google Earth on your computer and make a note of the longitude and latitude of two points in Wageningen that are relevant to you. Use a decimal degree notation with at least 4 digits after the decimal point. To change the settings in Google Earth click `Tools | Options` and change the Show lat/Long setting on the 3D View Tab.

Points

The example below shows how you can create spatial point objects from these coordinates.

```
R> # load sp package
R> library(sp)
R> # set working directory
R> # setwd("d:/rasta")    # set this to the appropriate directory
R>
R> # coordinates of two points identiefied in Google Earth, for example
R> pnt1_xy <- cbind(5.6660, 51.9872)
R> pnt2_xy <- cbind(5.6643, 51.9668)
R> # combine in single matrix
R> coords <- rbind(pnt1_xy, pnt2_xy)
R> # make spatial points
R> prj_string_WGS <- CRS("+proj=longlat +datum=WGS84")
R> mypoints <- SpatialPoints(coords, proj4string=prj_string_WGS)
R> # inspect object
R> class(mypoints)
```

```
[1] "SpatialPoints"
attr(,"package")
[1] "sp"
```

```
R> str(mypoints)

Formal class 'SpatialPoints' [package "sp"] with 3 slots
  ..@ coords     : num [1:2, 1:2] 5.67 5.66 51.99 51.97
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : NULL
  .. .. ..$ : chr [1:2] "coords.x1" "coords.x2"
  ..@ bbox       : num [1:2, 1:2] 5.66 51.97 5.67 51.99
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "coords.x1" "coords.x2"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
  .. .. ..@ projargs: chr "+proj=longlat +datum=WGS84"

R> # create and display some attribute data and store in a data frame
R> mydata <- data.frame(cbind(id = c(1,2), Name = c("my description 1",
+                                                    "my description 2")))
R> # make spatial points data frame
R> mypointsdf <- SpatialPointsDataFrame(coords, data = mydata,
+                                       proj4string=prj_string_WGS)
R> # inspect and plot object
R> class(mypointsdf)

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

R> names(mypointsdf)

[1] "id"   "Name"

R> str(mypointsdf)

Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
  ..@ data       :'data.frame':        2 obs. of  2 variables:
  .. ..$ id  : Factor w/ 2 levels "1","2": 1 2
  .. ..$ Name: Factor w/ 2 levels "my description 1",..: 1 2
  ..@ coords.nrs : num(0)
  ..@ coords     : num [1:2, 1:2] 5.67 5.66 51.99 51.97
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : NULL
  .. .. ..$ : chr [1:2] "coords.x1" "coords.x2"
  ..@ bbox       : num [1:2, 1:2] 5.66 51.97 5.67 51.99
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "coords.x1" "coords.x2"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
  .. .. ..@ projargs: chr "+proj=longlat +datum=WGS84"
```

```
R> spplot(mypointsdf, zcol="Name", col.regions = c("red", "blue"),
+          xlim = bbox(mypointsdf)[1, ]+c(-0.01,0.01),
+          ylim = bbox(mypointsdf)[2, ]+c(-0.01,0.01),
+          scales= list(draw = TRUE))
```

References

Bivand, R. S., Pebesma, E. J., & Rubio, V. G. (2013). Applied spatial data analysis with R, .

Pebesma, E. J., & Bivand, R. S. (2005). Classes and methods for spatial data in R. *R News*, *5*, 9–13.