Working with sp and aRT

Pedro Ribeiro de Andrade Neto
Paulo Justiniano Ribeiro Júnior

February 6, 2006

Contents

1 Introduction

sp is an important package for exchanging information between spatial packages. As aRT manipulates all spatial data formats, it must be fully connected to sp. All data in aRT is in sp format, but TerraLib databases can contain data that cannot be directly converted to sp data. For example:

1. TerraLib (and therefore aRT) requires ID in *all* spatial data, different from sp, that requires ID only with lines and polygons.
2. TerraLib layers have support to multigeometry, meaning that each spatial element can have more than one geometry associated. For example, a layer of cities can store their contours and centroids.
3. Geometries and attributes are stored in different objects in a TerraLib database. Geometries are stored directly inside of layers, while attributes are stored in tables inside layers. They cannot be in the same object because TerraLib supports different types of table, for example static, event and dynamic tables.

This document shows how to manipulate spatial data in aRT, showing how to import to and read from TerraLib databases. The data (and also some sentences) used in this document is based on *S Classes and Methods for Spatial Data: the* sp *Package*, by Pebesma and Bivand.

```
> library(aRT)

Loading required package: sp

------------------------------------------------
R-TERRALIB API
aRT version 0.4-15 (2005-12-20) is now loaded
------------------------------------------------
```

First we establish a connection to a DBMS. The database to be used in the exemples is called "sp". We remove it if exists and then we create a new one. We also work with aRT in the silent mode.

```
> aRTsilent(TRUE)

[1] TRUE

> con = openConn()
> if (any(showDbs(con) == "sp")) deleteDb(con, "sp", force = T)
> db = createDb(con, "sp")
```

2 Spatial points

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

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

We cannot work with a `SpatialPoints` object in `aRT` because it does not have `ID`. Therefore we use `SpatialPointsDataFrame`.

```
> xy.spdf = SpatialPointsDataFrame(xy, data.frame(ID = paste(1:10)))
```

To store this data we need to create a *layer*. A layer is a container that can store any geometric type and also other objects. A layer can be created in a database using `createLayer()`:

```
> lpoints = createLayer(db, "points")
> lpoints

Object of class aRTlayer

Layer: "points"
Database: "sp"
Number of polygons: 0
Number of lines: 0
Number of points: 0
Layer does not have raster data
Projection Name: "NoProjection"
Projection Datum: "Spherical"
Projection Longitude: 0
Projection Latitude: 0
Tables: (none)
```

Note that we have two names, `"points"`, the name in the database, and `lpoints`, the R object which can access `"points"`. The function `addPoints()` is used to store the points in a layer.

```
> addPoints(lpoints, xy.spdf)
```

To read any data from the layer, we need to have a table in the layer. It is a TerraLib requirement, then we need it even when the spatial data does not have attributes. Geometries with no entry in any table cannot be retrieved from the database.

```
> tpoints = createTable(lpoints, "tpoints")
> tpoints

Object of class aRTtable

Table: "tpoints"
Type: static
```

```
Layer: "points"
Rows: 10
Attributes:
    id: character[16] (key)
```

```
> lpoints
```

```
Object of class aRTlayer

Layer: "points"
Database: "sp"
Number of polygons: 0
Number of lines: 0
Number of points: 10
Layer does not have raster data
Projection Name: "NoProjection"
Projection Datum: "Spherical"
Projection Longitude: 0
Projection Latitude: 0
Tables:
    "tpoints": static
```

Now the layer has 10 points and one table, and we can read the data using `getPoints`:

```
> points = getPoints(lpoints)
```

We can see the data in the Figure 2. There are two ways for generating this image, plotting the points with `plot(points)`, or directly from the layer, with `plot(lpoints)`. As the points do not have an associated geometry, we do not care about their order, and the order is indeed different from the original. If we see the data we can check what is different:

```
> points
```

```
    coordinates ID
1   (0.67, 0.3)  1
2  (0.37, 0.28) 10
3  (0.96, 0.62)  2
4  (0.92, 0.91)  3
5  (0.77, 0.85)  4
6  (0.72, 0.46)  5
7  (0.74, 0.39)  6
8  (0.63, 0.64)  7
9  (0.19, 0.72)  8
10   (0.7, 0.2)  9
```

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

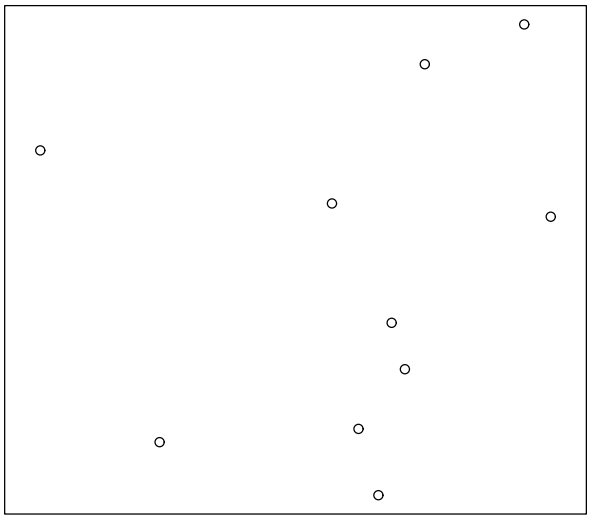

Figure 1: Plot of a layer with points

```
> df = data.frame(z1 = round(5 + rnorm(10), 2), z2 = 0:9, ID = paste(1:10))
> xy.spdf = SpatialPointsDataFrame(xy.sp, df)
> xy.spdf

    coordinates   z1 z2 ID
1   (0.67, 0.3) 3.10  0  1
2  (0.96, 0.62) 4.15  1  2
3  (0.92, 0.91) 3.68  2  3
4  (0.77, 0.85) 4.45  3  4
5  (0.72, 0.46) 6.62  4  5
6  (0.74, 0.39) 5.57  5  6
7  (0.63, 0.64) 3.66  6  7
8  (0.19, 0.72) 3.75  7  8
9    (0.7, 0.2) 5.19  8  9
10 (0.37, 0.28) 5.02  9 10

> lpointsdf = createLayer(db, "lpointsdf")
> addPoints(lpointsdf, xy.spdf)
```

As our object now has attributes, we can import the table data using `importTable()`.

```
> tpointsdf = importTable(lpointsdf, "tpointsdf", id = "ID", xy.spdf)
> tpointsdf

Object of class aRTtable

Table: "tpointsdf"
Type: static
Layer: "lpointsdf"
Rows: 10
Attributes:
    ID: character[16] (key)
    z1: numeric
    z2: integer
```

To read the data from the layer and the table together we can use a second argument of `getPoints` with the table to be read.

```
> getPoints(lpointsdf, tpointsdf)

    coordinates ID   z1 z2
1   (0.67, 0.3)  1 3.10  0
2  (0.37, 0.28) 10 5.02  9
3  (0.96, 0.62)  2 4.15  1
4  (0.92, 0.91)  3 3.68  2
5  (0.77, 0.85)  4 4.45  3
6  (0.72, 0.46)  5 6.62  4
7  (0.74, 0.39)  6 5.57  5
8  (0.63, 0.64)  7 3.66  6
9  (0.19, 0.72)  8 3.75  7
10   (0.7, 0.2)  9 5.19  8
```

3 Grids

(not supported yet)

4 Lines

4.1 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. As objects from this class already stores `ID`, they are pushed in the layer directly using `addLines()`.

```
> l1 = cbind(c(1, 2, 3), c(3, 2, 2))
> l1a = cbind(l1[, 1] + 0.05, l1[, 2] + 0.05)
> l2 = cbind(c(1, 2, 3), c(1, 1.5, 1))
> Sl1 = Line(l1)
> Sl1a = Line(l1a)
> Sl2 = Line(l2)
> S1 = Lines(list(Sl1), ID = "a")
> S2 = Lines(list(Sl2), ID = "b")
> S3 = Lines(list(Sl1a), ID = "c")
> Sl = SpatialLines(list(S1, S2, S3))
> llines = createLayer(db, "llines")
> addLines(llines, Sl)
> createTable(llines, "llines")

Object of class aRTtable

Table: "llines"
Type: static
Layer: "llines"
Rows: 3
Attributes:
    id: character[16] (key)
```

4.2 Building line objects with attributes

The same as polygons

5 Polygons

5.1 Building from scratch

The following example shows how a set of polygons are built from scratch. Note that `Sr4` has the opposite direction (right) as the other three; it is meant to represent a hole in the `Sr3` polygon.

```
> 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")
> SR = SpatialPolygons(list(Srs1, Srs2, Srs3), 1:3)
> lrings = createLayer(db, "lrings")
> addPolygons(lrings, SR)
```

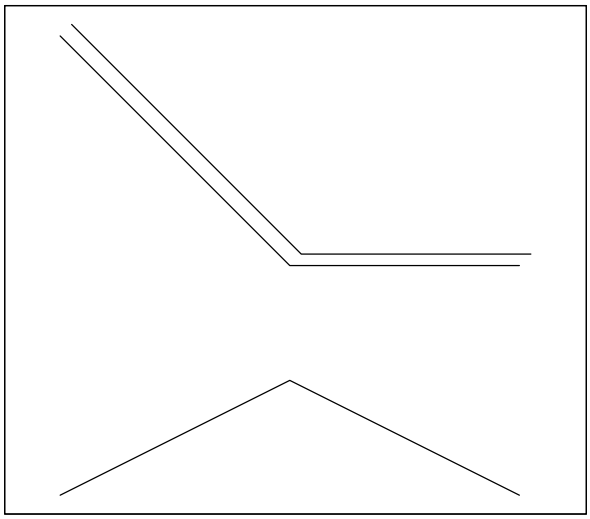

Figure 2: Plot of a layer with lines

```
> trings = createTable(lrings, "trings")
> lrings

Object of class aRTlayer

Layer: "lrings"
Database: "sp"
Number of polygons: 4
Number of lines: 0
Number of points: 0
Layer does not have raster data
Projection Name: "NoProjection"
Projection Datum: "Spherical"
Projection Longitude: 0
Projection Latitude: 0
Tables:
    "trings": static

> pols = getPolygons(lrings)
```

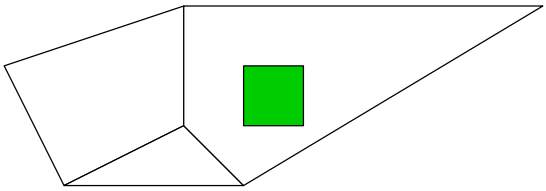

Figure 3: plot of a layer with polygons

5.2 Polygons with attributes

Polygons with attributes, objects of class `SpatialPolygonsDataFrame`, are built from the `SpatialPolygons` object (topology) and the attributes (data.frame):

To import the attributes, we need to create a table, but, due to the internal differences of `sp` data storage we need to insert `SrDf` manually, creating both table and the two integer columns before inserting the data:

References

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