

Working with **sp** and **aRT**

Pedro Ribeiro de Andrade Neto
Paulo Justiniano Ribeiro Júnior

August 16, 2005

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. **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 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 places in a **TerraLib** database. Geometries are stored directly in layers, while attributes are stored in tables inside layers. They cannot be in the same object because **TerraLib** supports different types of table (static, event, dynamics, ...).

This document shows how to manipulate 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
```

```
-----  
API R-TERRALIB  
aRT version 0.4-0 (2005-08-27) is now loaded  
-----
```

First we establish a connection to a DBMS. The database to be used in the examples is called “sp”. We remove it if exists and then we create a new one.

```

> 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

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

```

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

```

Object of class `aRTlayer`

```

Layer: "points"
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.sp)
```

To read any data from the layer, we need to generate a table, even if the spatial data does not have attributes (it is a **TerraLib** requirement). Geometries with no entry in any table cannot be retrieved from the database. The argument `genid=TRUE` at `createTable()` forces **aRT** to generate a column and fill it with the geometries unique ids.

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

Object of class **aRTtable**

```
Table: "tpoints"
Type: static
Layer: "points"
Rows: 10
Attributes:
  id: string[16] (key)
```

```
> lpoints
```

Object of class **aRTlayer**

```
Layer: "points"
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 `getGeometry`:

```
> getGeometry(lpoints)
```

```
$points  
$points[[1]]  
$points[[1]]$x  
[1] 0.67
```

```
$points[[1]]$y  
[1] 0.3
```

```
$points[[2]]  
$points[[2]]$x  
[1] 0.7
```

```
$points[[2]]$y  
[1] 0.2
```

```
$points[[3]]  
$points[[3]]$x  
[1] 0.37
```

```
$points[[3]]$y  
[1] 0.28
```

```
$points[[4]]  
$points[[4]]$x  
[1] 0.96
```

```
$points[[4]]$y  
[1] 0.62
```

```
$points[[5]]  
$points[[5]]$x  
[1] 0.92
```

```
$points[[5]]$y  
[1] 0.91
```

```
$points[[6]]  
$points[[6]]$x  
[1] 0.77
```

```
$points[[6]]$y  
[1] 0.85
```

```
$points[[7]]  
$points[[7]]$x  
[1] 0.72
```

```
$points[[7]]$y  
[1] 0.46
```

```
$points[[8]]  
$points[[8]]$x  
[1] 0.74
```

```
$points[[8]]$y  
[1] 0.39
```

```
$points[[9]]  
$points[[9]]$x  
[1] 0.63
```

```
$points[[9]]$y  
[1] 0.64
```

```
$points[[10]]  
$points[[10]]$x  
[1] 0.19
```

```
$points[[10]]$y  
[1] 0.72
```

```
$id  
[1] "1" "10" "2" "3" "4" "5" "6" "7" "8" "9"
```

```
attr(,"class")  
[1] "aRTgeometry"
```

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 convert it to a

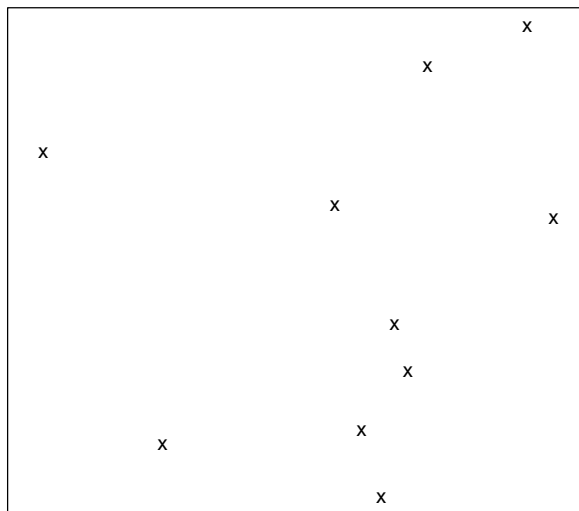


Figure 1: plot of a layer with points

`SpatialPointsDataFrame` we can check what is different:

```
> getGeometry(lpairs)
```

```
$points
$points[[1]]
$points[[1]]$x
[1] 0.67
```

```
$points[[1]]$y
[1] 0.3
```

```
$points[[2]]
$points[[2]]$x
[1] 0.7
```

```
$points[[2]]$y
[1] 0.2
```

```
$points[[3]]  
$points[[3]]$x  
[1] 0.37
```

```
$points[[3]]$y  
[1] 0.28
```

```
$points[[4]]  
$points[[4]]$x  
[1] 0.96
```

```
$points[[4]]$y  
[1] 0.62
```

```
$points[[5]]  
$points[[5]]$x  
[1] 0.92
```

```
$points[[5]]$y  
[1] 0.91
```

```
$points[[6]]  
$points[[6]]$x  
[1] 0.77
```

```
$points[[6]]$y  
[1] 0.85
```

```
$points[[7]]  
$points[[7]]$x  
[1] 0.72
```

```
$points[[7]]$y  
[1] 0.46
```

```
$points[[8]]  
$points[[8]]$x  
[1] 0.74
```

```
$points[[8]]$y  
[1] 0.39
```

```
$points[[9]]  
$points[[9]]$x  
[1] 0.63
```

```
$points[[9]]$y  
[1] 0.64
```

```
$points[[10]]  
$points[[10]]$x  
[1] 0.19
```

```
$points[[10]]$y  
[1] 0.72
```

```
$id  
[1] "1" "10" "2" "3" "4" "5" "6" "7" "8" "9"
```

```
attr("class")  
[1] "aRTgeometry"
```

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

```
> df = data.frame(z1 = round(5 + rnorm(10), 2), z2 = 0:9, ID = 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)
```



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

Object of class aRTtable

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

```
> getGeometry(lpointsdf)
```

```
$points
$points[[1]]
$points[[1]]$x
[1] 0.67
```

```
$points[[1]]$y
[1] 0.3
```

```
$points[[2]]
$points[[2]]$x
[1] 0.37
```

```
$points[[2]]$y
[1] 0.28
```

```
$points[[3]]
$points[[3]]$x
[1] 0.96
```

```
$points[[3]]$y
[1] 0.62
```

```
$points[[4]]
$points[[4]]$x
[1] 0.92
```

```
$points[[4]]$y
[1] 0.91
```

```
$points[[5]]  
$points[[5]]$x  
[1] 0.77
```

```
$points[[5]]$y  
[1] 0.85
```

```
$points[[6]]  
$points[[6]]$x  
[1] 0.72
```

```
$points[[6]]$y  
[1] 0.46
```

```
$points[[7]]  
$points[[7]]$x  
[1] 0.74
```

```
$points[[7]]$y  
[1] 0.39
```

```
$points[[8]]  
$points[[8]]$x  
[1] 0.63
```

```
$points[[8]]$y  
[1] 0.64
```

```
$points[[9]]  
$points[[9]]$x  
[1] 0.19
```

```
$points[[9]]$y  
[1] 0.72
```

```
$points[[10]]  
$points[[10]]$x  
[1] 0.7
```

```
$points[[10]]$y
[1] 0.2
```

```
$id
[1] "1" "10" "2" "3" "4" "5" "6" "7" "8" "9"
```

```
attr("class")
[1] "aRTgeometry"
```

```
> getData(tpointsdf)
```

	ID	z1	z2
1	1	3.1	0
2	2	4.15	1
3	3	3.68	2
4	4	4.45	3
5	5	6.62	4
6	6	5.57	5
7	7	3.66	6
8	8	3.75	7
9	9	5.19	8
10	10	5.02	9

As `SpatialPointsDataFrame` does not have ID, a simple merge of the layer with the table will not work¹.

3 Grids

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.

```
> 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))
> S11 = Line(l1)
> S11a = Line(l1a)
> S12 = Line(l2)
```

¹E-mail para eles, as outras classes tem id e fazem o merge, entao acho que nao é tarefa do aRT.

```

> S1 = Lines(list(S11, S11a), ID = "a")
> S2 = Lines(list(S12), ID = "b")
> S1 = SpatialLines(list(S1, S2))
> llines = createLayer(db, "llines")
> addLines(llines, S1)
> createTable(llines, "llines", gen = T)

```

Object of class aRTtable

```

Table: "llines"
Type: static
Layer: "llines"
Rows: 2
Attributes:
  id: string[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)
> trings = createTable(lrings, "trings", genid = T)
> lrings

```

Object of class aRTlayer

```

Layer: "lrings"
Number of polygons: 4
Number of lines: 0

```

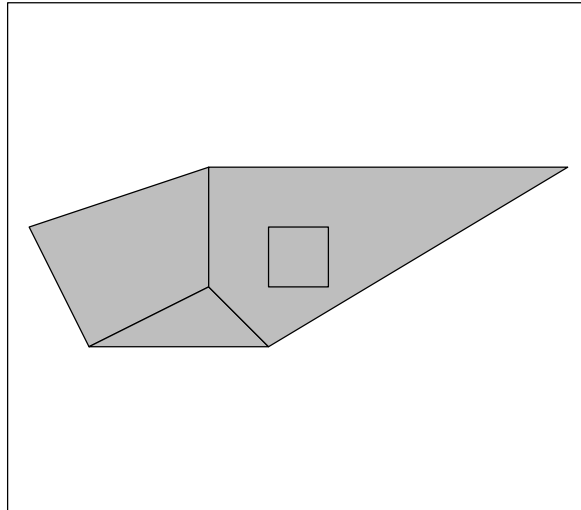


Figure 2: plot of a layer with polygons

```

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

```

5.2 Polygons with attributes

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

```

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

```

```
> lringsdf = createLayer(db, "lringsdf")
> addPolygons(lringsdf, SrDf)
```

To import the attributes, we need to create a table, but, due to the internal differences of `sp` data storage

```
> xy.spdf@data
```

An object of class "AttributeList"

Slot "att":

\$z1

```
[1] 3.10 4.15 3.68 4.45 6.62 5.57 3.66 3.75 5.19 5.02
```

\$z2

```
[1] 0 1 2 3 4 5 6 7 8 9
```

\$ID

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
> SrDf@data
```

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

```
> class(SrDf@data)
```

```
[1] "data.frame"
```

we need to insert `SrDf` manually, creating both table and the two integer columns before inserting the data:

```
> tringsdf = createTable(lringsdf, "tringsdf", id = "ID", gen = F)
> createColumn(tringsdf, "a", "i")
> createColumn(tringsdf, "b", "i")
> addRows(tringsdf, SrDf@data)
```

Using rownames as id

```
> tringsdf
```

Object of class `aRTtable`

Table: "tringsdf"

Type: static

Layer: "lringsdf"

Rows: 3

Attributes:

ID: string[16] (key)

a: integer

b: integer

```
> getData(tringsdf)
```

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

References

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