GIS in R: Tutorial 2

Marie Auger-Méthé

1 `Spatial` class

We will investigate two new classes of `Spatial` objects: `SpatialLines` and `SpatialPolygons`. Both of these classes are more complex than the `SpatialPoints` class we investigated during the last tutorial. First we will explore the slots associated with these objects. Remember that slots represents the important components of an object. To identify the slots of an object class, you can use the function `getSlots`.

```
# Just like SpatialPoints, SpatialLines and
# SpatialPolygons are object classes associated with
# the sp package
library(sp)
# Get the slots of SpatialPoints
getSlots("SpatialPoints")
```

```
##      coords        bbox proj4string
##    "matrix"    "matrix"       "CRS"
```

```
# Get the slots of SpatialLines
getSlots("SpatialLines")
```

```
##       lines        bbox proj4string
##      "list"    "matrix"       "CRS"
```

```
# Get the slots of SpatialPolygons
getSlots("SpatialPolygons")
```

```
##    polygons   plotOrder        bbox proj4string
##      "list"   "integer"    "matrix"       "CRS"
```

We can note here that a `SpatialPoints` object has three slots: `coords` with the coordinates of each points, `bbox` with the bounding box (extent) of the points, and `proj4sting` with

the coordinate reference system (CRS). While `SpatialLines` and `SpatialPolygons` have the `bbox` and `proj4string` slots just like `SpatialPoints`, they do not have the `coords` slot. Instead, `SpatialLines` and `SpatialPolygons` use the `lines` and `polygons` slots. These two slots are in fact lists of objects of class `Lines` and `Polygons`. Let's investigate the slots of `Lines` and `Polygons`.

```
# Note that both these object classes have slots that
# are lists.
# Get the slots of Lines
getSlots("Lines")
```

```
##       Lines          ID
##      "list" "character"
```

```
# Get the slots of Polygons
getSlots("Polygons")
```

```
##    Polygons   plotOrder       labpt          ID        area
##      "list"   "integer"   "numeric" "character"   "numeric"
```

As we can see, the `Lines` and `Polygons` objects also have a slot for lists. These slots take a list of objects of class `Line` or `Polygon`. `Lines` and `Polygons` also have an `ID` slot. This slot can only take a single character value. In the case of `Lines`, this ID connects all the `Line` objects that are in a `Lines` object with a single indentifier. Let's investigate the slots on `Line` and `Polygon`.

```
# Look at the slots of the fundamental class for
# SpatialLines
getSlots("Line")
```

```
##   coords
## "matrix"
```

```
# Look at the slots of the fundamental class for
# SpatialPolygon
getSlots("Polygon")
```

```
##     labpt      area      hole   ringDir    coords
## "numeric" "numeric" "logical" "integer"  "matrix"
```

```
# Note that both have the coords slot
```

The very basis of the `SpatialLines` and `SpatialPolygons` are the class `Line` and `Polygon`, which as we can see above are the classes that have a slot for coordinates. While `Line` and `Polygon` have the coordinates (`coords`), only `SpatialLines` and `SpatialPolygons` have the `bbox` and the `proj4string` slots.

1.1 SpatialLines

1.1.1 One movement path

Here we are going to create a `SpatialLines` object based on the locations of grey seals from Sable Island, Nova Scotia. The data used here is a subset of the data published in Lidgard et al. (2014) and was shared for educational purposes by the researchers of the Ocean Tracking Network (OTN) Canada Sable Island Grey Seal Bioprobes project (`http://members.oceantrack.org/data/discovery/SGS.htm`). To start with the simplest example possible, we will make a `SpatialLines` object with the data from only one of the seals.

```
# Read files with movement data
sealMov <- read.csv("Seal3_169_2_01_1.csv")
# Let's look at the first rows to get a sense of
# what's in the file
head(sealMov)
```

```
##                Date SealID LC    Lat     Lon
## 1 02.10.10 01:48:47  66486  0 44.680 -60.532
## 2 02.10.10 05:52:04  66486  1 44.687 -60.498
## 3 02.10.10 08:10:46  66486  3 44.759 -60.506
## 4 02.10.10 08:54:37  66486  0 44.776 -60.517
## 5 03.10.10 05:38:52  66486  1 44.692 -60.514
## 6 03.10.10 11:35:25  66486  0 44.802 -60.647
```

```
# How many seals do we have and what's their ID
unique(sealMov$SealID)
```

```
## [1] 66486 66506 66548
```

```
# Let's focuss on seal 66486, drop the locations from
# all other seals
seal1 <- subset(sealMov, SealID == 66486, drop = TRUE)
```

Now we want to order the locations of the seal by date, so when we connect the points with lines they are ordered to represent the movement of the animal. To do this, we will use

one of the Date-Time classes, which are important base classes in R.

```
# The current class of the date column is factor,
# which is not great for ordering dates
class(seal1$Date)
```

```
## [1] "factor"
```

```
# We will transform this column to a POSIXlt class,
# which is a base Date-Time class in R. This class
# understand how time should be ordered (e.g., year
# before month). To use POSIXlt, you need to specify
# the date format and the time zone (which we are
# assuming is UTC), see ?POSIXlt
# To see how the date is formated currently, let's
# look at the Date column
seal1$Date[1]
```

```
## [1] 02.10.10 01:48:47
## 455 Levels: 01.10.10 00:20:58 01.10.10 02:01:25 ... 31.10.10 22:35:14
```

```
# Looks like Day.Month.Year Hour:Minute:Second
seal1$Date <- as.POSIXlt(seal1$Date, format = "%d.%m.%y %H:%M:%S",
    zone = "UTC")
# Now the class is POSIXlt
class(seal1$Date)
```

```
## [1] "POSIXlt" "POSIXt"
```

```
# We can now order the seal1 object by date and time
seal1 <- seal1[order(seal1$Date), ]
```

Now, we will create a `SpatialPointsDataFrame` object based on the locations of the seal like. We will use one of the method we discussed in the last tutorial.

```
# Create a SpatialPointsDataFrame by assigning the
# coordinates
coordinates(seal1) <- ~Lon + Lat
# Assign the geographic CRS, which we are assuming is
# WGS84
proj4string(seal1) <- CRS("+proj=longlat +datum=WGS84")
```

```
# Now we have a SpatialPointsDataFrame object
class(seal1)
```

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

Now we will create a `Line` using the `SpatialPointsDataFrame`.

```
# Create a Line object
seal1L <- Line(seal1)
# Now the class is Line
class(seal1L)
```

```
## [1] "Line"
## attr(,"package")
## [1] "sp"
```

```
# Can we plot it?
plot(seal1L)
```

```
## Error in as.double(y):  cannot coerce type 'S4' to vector of type 'double'
```

```
# What's the info
summary(seal1L)
```

```
## Length  Class   Mode
##      1   Line     S4
```

As shown above, the `Line` object we've created is not useful in itself. All it does is assess that all of the coordinates are associated with one continuous `Line`. In our case, all of the locations of seal 66486 are associated with one continuous movement path. The `Line` object gets incorporated in a `Lines` object. Note that the `Lines` object is a list of `Line` objects with one associated ID.

```
# Create a Lines object using a Line
seal1Ls <- Lines(seal1L, ID = "seal1")
# You can't plot it
plot(seal1Ls)
```

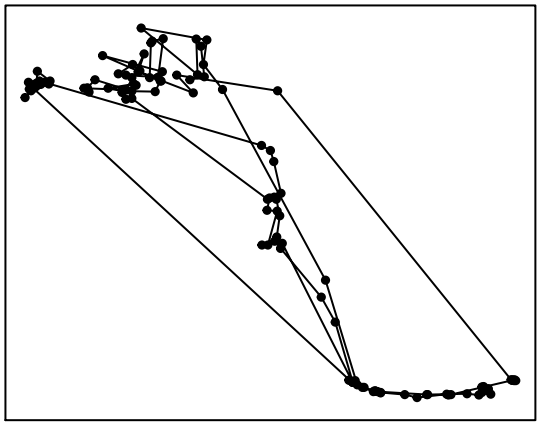
```
## Error in as.double(y):  cannot coerce type 'S4' to vector of type 'double'
```

```
# And the info is not that interesting
summary(seal1Ls)

## Length  Class   Mode
##      1  Lines     S4
```

The `Lines` object is also of little interest by itself, it's only a building block for the `SpatialLines`.

```
# Create a SpatialLines object with the Lines object.
# Note that to make SpatialLines you need a list of
# Lines, not just one Lines object
seal1SpL <- SpatialLines(list(seal1Ls))
# Now we finnally have a Spatial object that can be
# plotted
plot(seal1SpL)
# Let's add the location points
plot(seal1, add = TRUE, pch = 19, cex = 0.5)
# and a box around the plot
box()
```



Because `Line` and `Lines` are not `Spatial` object and only the `SpatialLines` object is a `Spatial` object (with a CRS), you need to specify the CRS. Here we are using the same CRS as the seal locations. Note that the `SpatialLines` assume that the animal is moving straight between the points, which may not make sense in all type of CRS, including in nonprojected CRS like the WGS84.

```
# Check the current CRS
proj4string(seal1SpL)

## [1] NA

# Assign the same CRS as the seal locations which are
# the base of this SpatialLines object
proj4string(seal1SpL) <- proj4string(seal1)
```

So I've shown you the step-by-step way to go from `SpatialPointsDataframe` to `SpatialLines`, but you could combined all of the lines to make one code line.

```
seal1SpL2 <- SpatialLines(list(Lines(Line(seal1), ID = "seal1")),
    proj4string = CRS(proj4string(seal1)))
# Because we are just lumping the functions from
# above, it should give you exactly the same results
identical(seal1SpL, seal1SpL2)
```

```
## [1] TRUE
```

An even quicker way to create a `SpatialLines` from a `SpatialPointsDataFrame` is to use `as`.

```
# Creating a SpatialLines from the seal1
# SpatialPoints object supper quickly.
seal1SpL3 <- as(seal1, "SpatialLines")
# This will not be exactly the same
identical(seal1SpL, seal1SpL3)
```

```
## [1] FALSE
```

```
# The only difference here is that sealSpL3 that
# we've just created has the defaut ID value: 'ID'.
# If we create a new SpatialLines using the Lines and
# Line functions and use 'ID' for the ID, this new
# object will be exactly the same as seal1SpL3.
seal1SpL4 <- SpatialLines(list(Lines(Line(seal1), ID = "ID")),
    proj4string = CRS(proj4string(seal1)))
identical(seal1SpL3, seal1SpL4)
```

```
## [1] TRUE
```

1.1.2 Three movement paths

In our first example, we only had one continous movement path. However, we might want a `SpatialLines` object with multiple movement paths. For example, we might want to have one `SpatialLines` object with the movement path of the three seals in our sealMov `data.frame`. To start, we would like to have each path to be a different `Lines` object with a different ID representing the seal ID. For this, we will need to create a `Lines` object with the coordinates of each seal.

```
# Before we create the line objects we would like to
# order the sealMov based on individual and then on
# time. Thus, we need to make the date column a
# POSIXlt object. See above for further explanation.
sealMov$Date <- as.POSIXlt(sealMov$Date, format = "%d.%m.%y %H:%M:%S",
    zone = "UTC")
# We will select the row by sealID below, so we only
# need to order it by dates
sealMovD <- sealMov[order(sealMov$Date), ]
# Check the first rows, now dates are in order but
# seal IDs are mixed
head(sealMovD)
```

```
##                    Date SealID LC    Lat     Lon
## 133 2010-10-01 00:20:58  66506  1 43.979 -60.045
## 134 2010-10-01 02:01:25  66506  1 44.019 -60.042
## 135 2010-10-02 00:02:21  66506  0 44.061 -59.982
## 1   2010-10-02 01:48:47  66486  0 44.680 -60.532
## 2   2010-10-02 05:52:04  66486  1 44.687 -60.498
## 3   2010-10-02 08:10:46  66486  3 44.759 -60.506
```

```
# We want to create a Lines object with the data
# points associated with each seal.
# If we did it for one seal, e.g.: SealID: 66506, we
# could do it as follow:
seal2Ls <- Lines(Line(sealMovD[sealMovD$SealID == 66506,
    c("Lon", "Lat")]), ID = 66506)
```

To most efficient way to do a `Lines` object for each seal, is through `lapply`. `lapply` is a base function in `R` that applies a function repeatly to an object. There are many such functions in `R`. `lapply` returns a list. Here is a few very easy examples of how to use `lapply`.

```
# Apply lapply to a simple function
# Create a matrix
oo <- matrix(1:3, nrow = 3)
# Just a column with value 1-3
oo
```

```
##      [,1]
## [1,]    1
```

```
## [2,]    2
## [3,]    3

# Use lapply with a function that adds 1 to the value
# of each row of oo
lapply(oo, function(x) {
    x + 1
})

## [[1]]
## [1] 2
##
## [[2]]
## [1] 3
##
## [[3]]
## [1] 4

# You could do something more complicated that use
# two different object. E.g., let's a create new
# matrix
aa <- matrix(1:6, nrow = 3)
# It's a matrix with value from 1-6 in two columns
aa

##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6

# Now we want to do the mean of each row using
# lapply. We use oo as the row index, as in:
# if oo is 1, do mean(aa[1,])
lapply(oo, function(x) {
    mean(aa[x, ])
})

## [[1]]
## [1] 2.5
##
```

```
## [[2]]
## [1] 3.5
##
## [[3]]
## [1] 4.5
```

Now we will use `lapply` to do a list of `Lines`. We are going to use the SealID as our index and create one `Lines` object per seal.

```
# Get the ID of the 3 seals
sealIndex <- unique(sealMovD$SealID)
sealIndex

## [1] 66506 66486 66548

# now use the lapply on the code we described above:
# Lines(Line(sealMovD[sealMovD£SealID == 66506,
# c('Lon','Lat')]), ID = 66506)
# Note that the ID of each Lines object is the seal
# ID
sealsLs <- lapply(sealIndex, function(x) {
    Lines(Line(sealMovD[sealMovD$SealID == x, c("Lon",
        "Lat")]), ID = x)
})
# We should get a list back
class(sealsLs)

## [1] "list"

# And the elements of this list should be a Lines
# object
class(sealsLs[[1]])

## [1] "Lines"
## attr(,"package")
## [1] "sp"
```

We can now use this list to create a `SpatialLines`.

```
# Create a SpatialLines based on the list and assign
# the CRS
sealsSpL <- SpatialLines(sealsLs, proj4string = CRS(proj4string(seal1)))
# We can plot it and assign different coulors to the
# different Lines
plot(sealsSpL, col = c("red", "purple", "orange"))
# with a box around the plot
box()
```

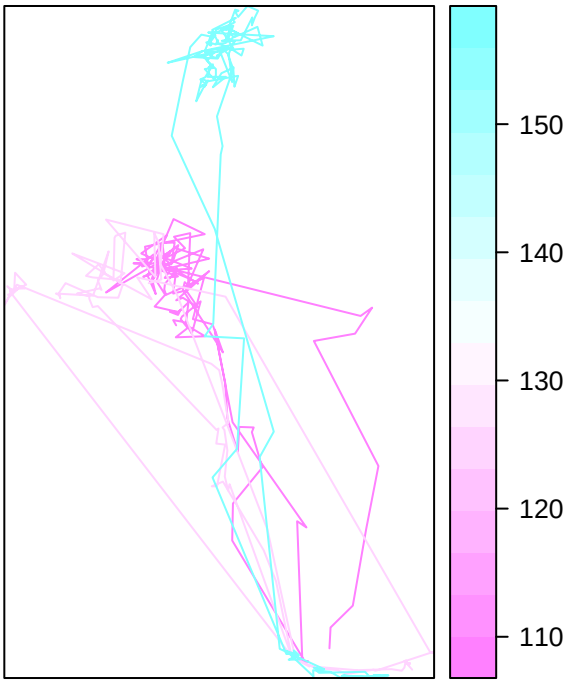
In the last tutorial, I showed how `SpatialPoints` had an analogue with attributes called `SpatialPointsDataFrame`. We can also create a `SpatialLinesDataFrame` from a `SpatialLines` object. Here we are going to add attributes to the seal movement paths.

```
# We are going to create a data.frame with 2 columns
# to add as attributes to each seal. This is invented
# data; I'm going to assign arbitrary values to each
# seal that represents their age and weight
sealAtt <- as.data.frame(cbind(age = c(1.2, 3, 3.2),
```

```
    weight = c(110, 126, 156)))
# We match the attributes based on ID and we need to
# assign the ID as the row name of our data.frame
rownames(sealAtt) <- sealIndex
# Now we can create the SpatialLinesDataFrame
sealsSpLdf <- SpatialLinesDataFrame(sealsSpL, data = sealAtt)
# Info
summary(sealsSpLdf)
```

```
## Object of class SpatialLinesDataFrame
## Coordinates:
##       min     max
## x -60.921 -59.763
## y  43.922  45.212
## Is projected: FALSE
## proj4string : [+proj=longlat +datum=WGS84]
## Data attributes:
##       age            weight
##  Min.   :1.200   Min.   :110.0
##  1st Qu.:2.100   1st Qu.:118.0
##  Median :3.000   Median :126.0
##  Mean   :2.467   Mean   :130.7
##  3rd Qu.:3.100   3rd Qu.:141.0
##  Max.   :3.200   Max.   :156.0
```

```
# Now you could plot the Lines and color them based
# on the values of the attributes, e.g. based on
# their weight
spplot(sealsSpLdf, zcol = "weight")
```



Note that we have been putting one `Line` object per `Lines` object. However, a `Lines` object can contain multiple `Line` objects, as long as these can be associated with a single ID. I'm not sure when this would be useful, but one example I could think of, is if you had a movement path that was disconnected. For example, if you had multiple foraging trips for one individual. Here I'm only going to show you how this work by putting all three seals as one `Lines` objects rather than as one `Lines` object per individual.

```
# The big difference here is that we are using 1
# Lines for all 3 seals. Thus, we use sapply to
# create a list of Line objects (as opposed to a list
# of Lines objects)
sealsL <- lapply(sealIndex, function(x) {
    Line(sealMovD[sealMovD$SealID == x, c("Lon", "Lat")])
})
# We should get a list back
class(sealsL)

## [1] "list"
```

```r
# And now the elements of this list should be a Line
# object
class(sealsL[[1]])
```

```
## [1] "Line"
## attr(,"package")
## [1] "sp"
```

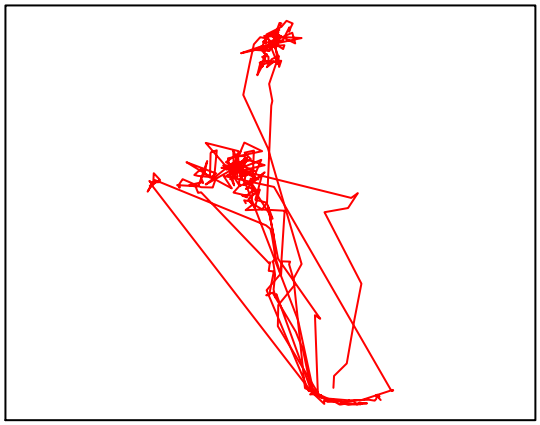
```r
# compare to Lines associated with the sealsLs that
# we created above
class(sealsLs[[1]])
```

```
## [1] "Lines"
## attr(,"package")
## [1] "sp"
```

```r
# If we create a Lines object with the 3 Line
# objects, we won't be able to assign to each Line
# the seal ID, we will need to give it only one ID
# which groups the 3 Line, here just 'OTN seals'.
sealsLs2 <- Lines(sealsL, ID = "OTN seals")
# You can try to put 3 IDs put it won't work
sealsLs3 <- Lines(sealsL, ID = sealIndex)
```

```
## Error in Lines(sealsL, ID = sealIndex):  Single ID required
```

```r
# Now we can make a SpatialLines from the new Lines
# object we have created, remember that SpatialLines
# needs a list of Lines
sealsSpL2 <- SpatialLines(list(sealsLs2), proj4string = CRS(proj4string(seal1)))
# We can plot this object, but assigning colour is
# not going to work in the same way. Only the first
# colour is going to be used because all of these 3
# movement paths are group together under one ID.
plot(sealsSpL2, col = c("red", "purple", "orange"))
box()
```



The disadvantage here is that we can only assign one set of attributes for the three seals at the same time. We can't differentiate between them. That's because you link the attributes to the `SpatialLines` based on the ID slot, which is only attributed to the `Lines`, not the `Line`. This is demonstrated in the example below.

```
# We can't link data.frame we have created above
sealsSpLdf2 <- SpatialLinesDataFrame(sealsSpL2, data = sealAtt)

## Error in SpatialLinesDataFrame(sealsSpL2, data = sealAtt):  row.names of data
and Lines IDs do not match

# Even if we say to match it without the ID names
sealsSpLdf2 <- SpatialLinesDataFrame(sealsSpL2, data = sealAtt,
    match.ID = FALSE)

## Error in SpatialLinesDataFrame(sealsSpL2, data = sealAtt, match.ID = FALSE):
length of data.frame does not match number of Lines elements

# You could do it if your data.frame has only one row
sealsSpLdf2 <- SpatialLinesDataFrame(sealsSpL2, data = sealAtt[1,
```

```
    ], match.ID = FALSE)
# And that's becase the sealsSpL2 only has one Lines
# object
length(slot(sealsSpLdf2, "lines"))

## [1] 1

# Compare to the previous one that had 3
length(slot(sealsSpLdf, "lines"))

## [1] 3
```

So what you should gather from this, is that you should create a `Lines` object for all the entities you're interested in separating or that you'll need to associate with attributes.

Exercise 1

Import the waveglider.csv file, which is a modified version of the wg_m42_waveglider.csv found on the OTN Ocean Glidders and Marine Observation website (`http://gliders.oceantrack.org/ajax/waveglider/`), see main page for more information (`http://gliders.oceantrack.org`). Create a `SpatialLinesDataFrame` with this file. Make one `Lines` per day. You can use the day column to help you with this. Don't forget to order the rows by time. Note that in this case the time is in seconds since Jan 1 1970 00:00 UTC. To create a correct `R` date time object, use: `as.POSIXct(waveglider$time, origin="1970-01-01")`.

Exercise 2

Plot this `SpatialLinesDataFrame`. Use different colours for each day.

2 References

Lidgard DC, Bowen WD, Jonsen ID, Iverson SJ (2014) Predator-borne acoustic transceivers and GPS tracking reveal spatial and temporal patterns of encounters with acoustically-tagged fish in the open ocean. Mar Ecol Prog Ser 501:157-168