GIS in R: Tutorial 2

Marie Auger-Méthé

1 `Spatial` class

Today 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 are the important component of the object. To get the slots of an class, you can use the function `getSlots`.

```
# The are all object associated with the sp package
library(sp)
# Get slots of SpatialPoints
getSlots("SpatialPoints")
```

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

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

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

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

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

We can note here that the SpatialPoints need the slots: `coords` with the coordinates of each points, `bbox` with the bounding box (extent) of the points, and `proj4sting` which has the coordinate reference system (CRS). While both the `SpatialLines` and `SpatialPolygons` use the `bbox` and `proj4string` just like the `SpatialPoints`, they do not use `coords` slots

and instead use `lines` and `polygons`. These two slots are in fact list of objects of class `Line` and `Polygon`.

```
# Note that both these 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"
```

The lists will take a list of object of class Line or Polygon, and each of the Line and Polygon in the list will need to be associated with an ID in the ID slot. The very basis of the `SpatialLines` and `SpatialPolygons` are the class `Line` and `Polygon`, which as we can see below are the class that have a slot for coordinates.

```
# 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
```

1.1 SpatialLines

Here we are going to create a `SpatialLines` object based on the location of a grey seal. The data used here is the a subset of the data published in Lidgard et al. (2014), that

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`). Just to use the simplest example possible, we will make a `SpatialLines` object with only one seal.

```
# Read files with movement data
sealMov <- read.csv("Seal3_169_2_01_1.csv")
# Let's look at the first column 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
```

```
# Get only the points for seal 66486
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 the `DateTime` class, which is an important class in R.

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

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

```
# Make the date time into a POSIXlt class, which is a
# basic class for date and time in R. This class
# understand how time should be ordered. You need to
# specify the format of the data and the time zone
```

```
# (which we are assuming is UTC), see ?POSIXlt
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 location of the seal like we did in the last tutorial.

```
# We can create a SpatialPointsDataFrame by assigning
# the coordinates
coordinates(seal1) <- ~Lon + Lat
# We then 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`.

```
# Creat 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
```

You have noticed that the `Line` object we just created is not useful in itself. All it does is assess that all of the points are associated with one line. In our case all of the locations of this seal is associated with one movement path. The `Line` object needs to be incorporated in a `Lines` object. Note that the `Lines` object is a list of `Line` with an associated ID. The `Lines` object is also of little interest by itself, it's only a building block for the `SpatialLines`.

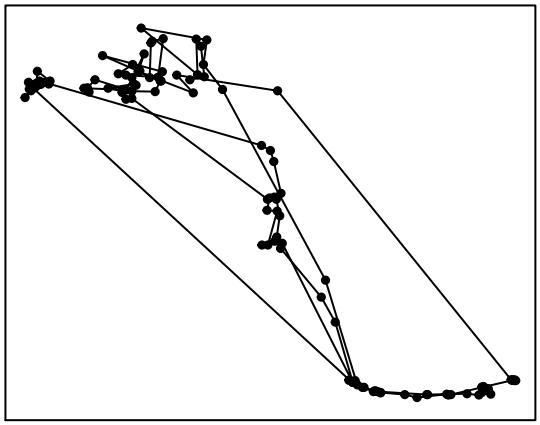
```
# Create a Lines object using a Line
seal1Ls <- Lines(seal1L, ID = "seal1")
# You can't plot it
plot(seal1Ls)
```

```
## 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
```

```
# But let's create a SpatialLines object with this
# Lines object. Note that to make SpatialLines you
# need a list of Lines
seal1SpL <- SpatialLines(list(seal1Ls))
# Now we finnally have a Spatial object that can be
# plotted
plot(seal1SpL)
# Let's add the 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
# just created has the defaut ID 'ID', if we create a
# new SpatialLines but name the ID as 'ID' they are
# going to be identical.
seal1SpL4 <- SpatialLines(list(Lines(Line(seal1), ID = "ID")),
    proj4string = CRS(proj4string(seal1)))
identical(seal1SpL3, seal1SpL4)
```

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

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

```
# Before we create the line objects we would like to
# order the sealMov based on indibidual and then on
# time, so 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 followed by date, use
# order
sealMovD <- sealMov[order(sealMov$Date), ]
# Check the first rows, now date are in order but
# SealID is 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
```

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

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

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

```
# To familiarize yourself with lapply, we going to
# apply it 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
```

```
# Here we make an lapply function that adds 1 to the
# value each 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 create new matrix
aa <- matrix(1:6, nrow = 3)
# Just 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 we 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` 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)
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
```

```r
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()
```

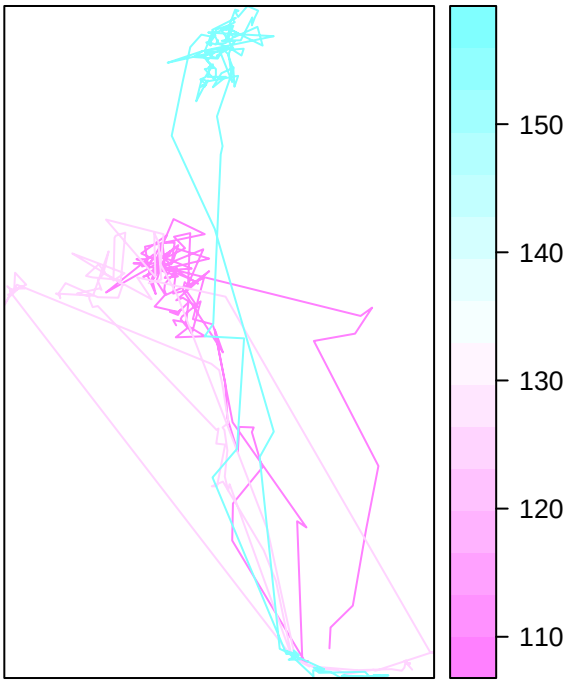
```r
# Add a bit of canada to give you a reference point
map("world", region = "Canada", add = TRUE)

## Error in eval(expr, envir, enclos):  could not find function "map"

# These animals are moving from Sable Island at sea
# and back.
```

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.

```r
# We are going to create a data.frame with some
# attribute to add to each seal. This is a completely
# inveted example and I'm going to assign arbitrary
# value to each seal that represents, let say, their
# let's say age and one with weight
sealAtt <- as.data.frame(cbind(age = c(1.2, 3, 3.2),
    weight = c(110, 126, 156)))
# We match the attribute based on ID and we need to
# assign the ID as the row name of our datae 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
```

```r
# Now you could plot the Lines based on their value,
# 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`, 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 was 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 3 seal as one `Lines` rather than one `Lines` per individual.

```
# This is a bit of an artificial example The big
# difference here is that we are using one Lines for
# all 3 seals, so here we use sapply only to create a
# list of Line object not Lines object
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"
```

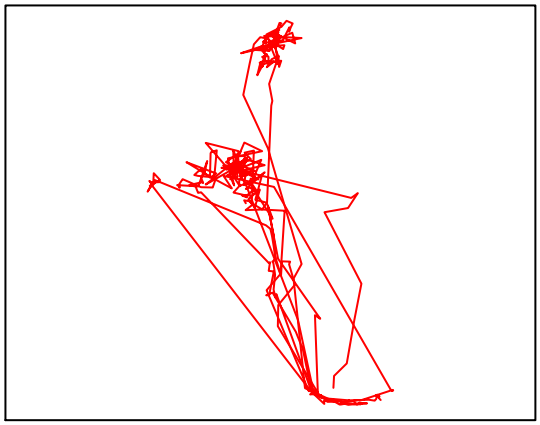
```r
# compare to the sealsLs that we created above
class(sealsLs[[1]])
```

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

```r
# If we create a Lines object with the 3 Line, 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 color is not
# going to work teh same, only the first color is
# going to be used because all of these 3 movement
# path are group together under one ID.
plot(sealsSpL2, col = c("red", "purple", "orange"))
box()
```



However, the disadvantage here is that we can only assign a set of attribute for the three seals at the same time. We can't differentiate between them. That's becasue you link the attributes to the `SpatialLines` based on the ID slot, which is only attributed to the `Lines`, not the `Line`. the example below demonstrate the points.

```
# We can 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 can retain here is that you should do as many `Lines` as you want to have separated entities.

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 inseconds since Jan 1 1970 00:00 UTC. So 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