Reading shapefiles into R for use with the `overlapptest` package

Marcelino de la Cruz

2018-04-03
`overlapptest` version `1.0`

This vignette explains how to read shapefile data into R for use the `overlapptest` package. It is based on the vignette about handling shape files in the package `spatstat` (Baddeley et al. 2015).

This vignette is part of the documentation included in `overlapptest` version `1.0`. The information applies to `overlapptest` versions `1.0-0` and above.

1 Shapefiles

As explained by Baddeley et al. (2015), a shapefile represents a list of spatial objects — a list of points, a list of lines, or a list of polygonal regions — and each object in the list may have additional variables attached to it. The `overlapptest` package deals only with polygonal objects.

A dataset stored in shapefile format is actually stored in a collection of text files, for example

```
mydata.shp
mydata.prj
mydata.sbn
mydata.dbf
```

which all have the same base name `mydata` but different file extensions. To refer to this collection you will always use the filename with the extension `shp`, for example `mydata.shp`.

2 Helper packages

We'll use two other packages to handle shapefile data.

The `maptools` package is designed specifically for handling file formats for spatial data. It contains facilities for reading and writing files in shapefile format.

The `spatstat` package contains functions for handling polygons (as `owin` objects) which allow computing areas, centroids, intersections and rotations.

3 How to read shapefiles into R for use with `overlapptest`

To read shapefile data into R, for use with `overlapptest`, two steps must be followed:

1. using the facilities of `maptools`, read the shapefiles as a Spatial*DataFrame object.

2. convert the Spatial*DataFrame object into a multi-polygonal owin object supported by `spatstat`.

3.1 Read shapefiles using `maptools`

Here's how to read shapefile data.

1. ensure that the package `maptools` is installed. You will need version `0.7-16` or later.
2. start R and load the package:

   ```
   > library(maptools)
   ```

3. read the shapefile into `R` using `readShapeSpatial`, for example

   ```
   > x <- readShapeSpatial("mydata.shp")
   ```

4. To find out what kind of spatial objects are represented by the dataset, inspect its class:

   ```
   > class(x)
   ```

   For applications of the `overlapptest` package, the class should typicaly be a `SpatialPolygonsDataFrame`.

For example, to read in the shapefile data supplied in the `overlapptest` package, we just should set the working directory to the folder where there are the sapefile data and use the `readShapeSpatial()` function of `maptools`.

```
> setwd(system.file("shapes", package="overlapptest"))
> Androsace <- readShapeSpatial("Androsace.shp")

> class(Androsace)

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

3.2 Convert data to `spatstat` format

Both packages `maptools` and `spatstat` must be loaded in order to convert the data.

```
> library(maptools)
> library(spatstat)
```

In addition, for applications of the `overlapptest` package, it is fundamental to avoid the automatic correction of polygons implemented in `spatstat` which, by default, will try to "repare" overlapping pieces (it would also try repairing polygon self-intersections, so the geometry of the shapefiles should be reliable). For this, just type,

```
> spatstat.options(fixpolygons=FALSE)
```

There are different ways to convert the dataset to an object in the `spatstat` package, as explained in the corresponding vignette in `spatstat`. For applications of the `overlapptest`, the most convenient way is combining all the polygonal elements of the same type (ussually present in a unique shapefile) into a single "polygonal region", and convert this to a single object of class `owin`. To do this, use `as(x, "owin")` or `as.owin(x)`. The result is a single window (object of class `"owin"`) in the `spatstat` package. In our example,

```
> Androsace<- as(Androsace, "owin")
```

3.3 Checking the reliability of the `owin` object

For this, we should load the `overlapptest` package.

```
> library(overlapptest)
```

The function `check.ventana()` will check that the vertices of all polygons are listed anti-clockwise (to ensure that `spatstat` considers them as "solid" polygons and not "holes", something necessary to be able to compute intersections among them). If it finds some clockwise listed vertices, it would try to reorder them, and will return the corrected `owin` object. If it succeded, the order number of the corrected polygon(s) would be listed as the attribute `corrected` of the `owin` object. If it would not, the order number of the wrong polygon(s) would be listed as the attribute `not.corrected`. These polygons should be corrected manually before using the other functions in the `overlapptest` package.

```
> Androsace <- check.ventana(Androsace)

57 problematic polygon(s) detected

all problematic polygons have been repared
```

In this case the message warns that 57 polygons have been corrected so no manual correction is necessary. To re-check this, we can examine the attributes of the `owin` object.

```
> attributes (Androsace)

$names
[1] "type"   "xrange" "yrange" "bdry"   "units"

$class
[1] "owin"

$corrected
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 51
[51] 52 53 54 55 56 57 58
```

In case that running `check.ventana()` would not produce any warnings that would mean that all the polygons were correct. Once the polygons have been checked, the `owin` object could be used with the other functions in the `overlapptest` package.