Handling shapefiles in the `spatstat` package

Adrian Baddeley

2011-11-11
`spatstat` version `1.24-2`

This vignette explains how to read data into the `spatstat` package from files in the popular 'shapefile' format.

This vignette is part of the documentation included in `spatstat` version `1.24-2`. The information applies to `spatstat` versions `1.18-0` and above.

1 Shapefiles

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.

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 three 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 sp package supports a standard set of spatial data types in R. These standard data types can be handled by many other packages, so it is useful to convert your spatial data into one of the data types supported by sp.

The gpclib package supports geometrical operations on polygons — such as union and intersection. However, gpclib has a restrictive licence which forbids its use for commercial activities. Type licence.polygons() for more information. **By default, gpclib is disabled. It must be enabled** by typing spatstat.options(gpclib=TRUE) assuming you are covered by the licence.

3 How to read shapefiles into spatstat

To read shapefile data into spatstat, you follow two steps:

1. using the facilities of maptools, read the shapefiles and store the data in one of the standard formats supported by sp.

2. convert the sp data type into one of the data types 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 an object in the sp package 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)
```

The class may be either `SpatialPoints` indicating a point pattern, `SpatialLines` indicating a list of polygonal lines, or `SpatialPolygons` indicating a list of polygons. It may also be `SpatialPointsDataFrame`, `SpatialLinesDataFrame` or `SpatialPolygonsDataFrame` indicating that, in addition to the spatial objects, there is a data frame of additional variables.

Here are some examples, using the example shapefiles supplied in the `maptools` package itself.

```
> setwd(system.file("shapes", package="maptools"))
> baltim   <- readShapeSpatial("baltim.shp")
> columbus <- readShapeSpatial("columbus.shp")
> fylk     <- readShapeSpatial("fylk-val.shp")
```

```
> class(baltim)
```

```
[1] "SpatialPointsDataFrame"
```

```
> class(columbus)
```

```
[1] "SpatialPolygonsDataFrame"
```

```
> class(fylk)
```

```
[1] "SpatialLinesDataFrame"
```

3.2 Convert data to `spatstat` format

To convert the dataset to an object in the `spatstat` package, the procedure depends on the type of data, as explained below.

3.2.1 Objects of class `SpatialPoints`

An object `x` of class `SpatialPoints` represents a spatial point pattern. Use `as(x, "ppp")` or `as.ppp(x)` to convert it to a spatial point pattern in `spatstat`.

The window for the point pattern will be taken from the bounding box of the points. You will probably wish to change this window, usually by taking another dataset to provide the window information. Use `[.ppp` to change the window: if `X` is a point pattern object of class `"ppp"` and `W` is a window object of class `"owin"`, type

```
> X <- X[W]
```

3.2.2 Objects of class SpatialPointsDataFrame

An object x of class SpatialPointsDataFrame represents a pattern of points with additional variables ('marks') attached to each point. It includes an object of class SpatialPoints giving the point locations, and a data frame containing the additional variables attached to the points.

Use as(x, "ppp") or as.ppp(x) to convert an object x of class SpatialPointsDataFrame to a spatial point pattern in spatstat. In this conversion, the data frame of additional variables in x will become the marks of the point pattern z.

```
> y <- as(x, "ppp")
```

Before the conversion you can extract the data frame of auxiliary data by df <- x@data or df <- slot(x, "data"). After the conversion you can extract these data by df <- marks(y).

For example:

```
> balt <- as(baltim, "ppp")
> bdata <- slot(baltim, "data")
```

3.2.3 Objects of class SpatialLines

A "line segment" is the straight line between two points in the plane.

In the spatstat package, an object of class psp ("planar segment pattern") represents a pattern of line segments, which may or may not be connected to each other (like matches which have fallen at random on the ground).

In the sp package, an object of class SpatialLines represents a **list of lists** of **connected curves**, each curve consisting of a sequence of straight line segments that are joined together (like several pieces of a broken bicycle chain.)

So these two data types do not correspond exactly.

The list-of-lists hierarchy in a SpatialLines object is useful when representing internal divisions in a country. For example, if USA is an object of class SpatialLines representing the borders of the United States of America, then USA@lines might be a list of length 52, with USA@lines[[i]] representing the borders of the i-th State. The borders of each State consist of several different curved lines. Thus USA@lines[[i]]@Lines[[j]] would represent the jth piece of the boundary of the i-th State.

If x is an object of class SpatialLines, there are several things that you might want to do:

1. collect together all the line segments (all the segments that make up all the connected curves) and store them as a single object of class `psp`.

   To do this, use `as(x, "psp")` or `as.psp(x)` to convert it to a spatial line segment pattern.

2. convert each connected curve to an object of class `psp`, keeping different connected curves separate.

   To do this, type something like the following:

   ```
   > out <- lapply(x@lines, function(z) { lapply(z@Lines, as.psp) })
   ```

   The result will be a **list of lists** of objects of class `psp`. Each one of these objects represents a connected curve, although the `spatstat` package does not know that. The list structure will reflect the list structure of the original `SpatialLines` object `x`. If that's not what you want, then use `curvelist <- do.call("c", out)` or

   ```
   > curvegroup <- lapply(out, function(z) { do.call("superimposePSP", z)})
   ```

   to collapse the list-of-lists-of-`psp`'s into a list-of-`psp`'s. In the first case, `curvelist[[i]]` is a `psp` object representing the `i`-th connected curve. In the second case, `curvegroup[[i]]` is a `psp` object containing all the line segments in the `i`-th group of connected curves (for example the `i`-th State in the `USA` example).

The window for the spatial line segment pattern can be specified as an argument `window` to the function `as.psp`.

3.2.4 Objects of class `SpatialLinesDataFrame`

An object `x` of class `SpatialLinesDataFrame` is a `SpatialLines` object with additional data. The additional data is stored as a data frame `x@data` with one row for each entry in `x@lines`, that is, one row for each group of connected curves.

In the `spatstat` package, an object of class `psp` (representing a collection of line segments) may have a data frame of marks. Note that each *line segment* in a `psp` object may have different mark values.

If `x` is an object of class `SpatialLinesDataFrame`, there are two things that you might want to do:

1. collect together all the line segments that make up all the connected lines, and store them as a single object of class psp.

   To do this, use as(x, "psp") or as.psp(x) to convert it to a marked spatial line segment pattern.

2. keep each connected curve separate, and convert each connected curve to an object of class psp. To do this, type something like the following:

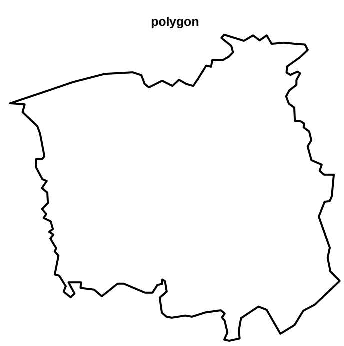
```
> out <- lapply(x@lines, function(z) { lapply(z@Lines, as.psp) })
> dat <- x@data
> for(i in seq(nrow(dat)))
+   out[[i]] <- lapply(out[[i]], "marks<-", value=dat[i, , drop=FALSE])
```

   The result is a list-of-lists-of-psp's. See the previous subsection for explanation on how to change this using c() or superimposePSP.

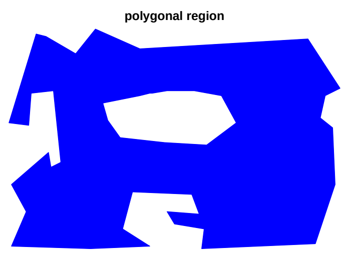
In either case, the mark variables attached to a particular *group of connected lines* in the SpatialLinesDataFrame object, will be duplicated and attached to each *line segment* in the resulting psp object.

3.2.5 Objects of class SpatialPolygons

First, so that we don't go completely crazy, let's introduce some terminology. A *polygon* is a closed curve that is composed of straight line segments. You can draw a polygon without lifting your pen from the paper.



A *polygonal region* is a region in space whose boundary is composed of straight line segments. A polygonal region may consist of several unconnected pieces, and each piece may have holes. The boundary of a polygonal region consists of one or more polygons. To draw the boundary of a polygonal region, you may need to lift and drop the pen several times.



An object of class `owin` in `spatstat` represents a polygonal region. It is a region of space that is delimited by boundaries made of lines.

An object `x` of class `SpatialPolygons` represents a **list of polygonal regions**. For example, a single object of class `SpatialPolygons` could store information about every State in the United States of America (or the United States of Malaysia). Each State would be a separate polygonal region (and it might contain holes such as lakes).

There are two things that you might want to do with an object of class `SpatialPolygons`:

1. combine all the polygonal regions together into a single polygonal region, and convert this to a single object of class `owin`.

   For example, you could combine all the States of the USA together and obtain a single object that represents the territory of the USA.

   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.

2. keep the different polygonal regions separate; convert each one of the polygonal regions to an object of class `owin`.

   For example, you could keep the States of the USA separate, and convert each State to an object of class `owin`.

   To do this, type the following:

```
> regions <- slot(x, "polygons")
> regions <- lapply(regions, function(x) { SpatialPolygons(list(x)) })
> windows <- lapply(regions, as.owin)
```

The result is a list of objects of class `owin`. Often it would make sense to convert this to a tessellation object, by typing

```
> te <- tess(tiles=windows)
```

The conversion process may generate an error message, saying that some of the polygons intersect each other, or are self-intersecting, or violate other geometrical conditions. This happens because an object of class `SpatialPolygons` is just a list of lists of polygons, possibly self-intersecting or mutually intersecting, but an object of class `"owin"` is intended to specify a well-defined region of space.

If you chose option 1, the conversion process will check whether any of the polygons in `x` intersect each other. This often generates an error with a shapefile representing a division of space into states or counties or administrative regions, like the Départements of France, because two adjacent regions have boundaries that intersect (even though the intersection has zero area). If you chose option 2, the conversion process will only check whether, for each polygonal region in `x`, the component polygons intersect each other. This will *usually* avoid the checking problem.

If an error occurs, the error message will usually specify which component polygons fail the test. The best strategy is usually just to plot the object `x` (using the plot facilities in `sp`) to identify the problem.

It is possible to suppress the stringent checking of polygons in `spatstat` during the conversion:

```
> spatstat.options(checkpolygons=FALSE)
> y <- as(x, "owin")
> spatstat.options(checkpolygons=TRUE)
```

The resulting object `y` should be inspected carefully and used circumspectly; it has not passed the stringent tests required for many algorithms in `spatstat`.

3.2.6 Objects of class `SpatialPolygonsDataFrame`

What a mouthful!

An object `x` of class `SpatialPolygonsDataFrame` represents a list of polygonal regions, with additional variables attached to each region. It includes an object of class `SpatialPolygons` giving the spatial regions, and a data frame containing the additional variables attached to the regions. The regions are extracted by

```
> y <- as(x, "SpatialPolygons")
```

and you then proceed as above to convert the curves to `spatstat` format.

The data frame of auxiliary data is extracted by `df <- x@data` or `df <- slot(x, "data")`.

For example:

```
> cp       <- as(columbus, "SpatialPolygons")
> cregions <- slot(cp, "polygons")
> cregions <- lapply(cregions, function(x) { SpatialPolygons(list(x)) })
> cwindows <- lapply(cregions, as.owin)
```

There is currently no facility in `spatstat` for attaching marks to an `owin` object directly.

However, `spatstat` supports objects called **hyperframes**, which are like data frames except that the entries can be any type of object. Thus we can represent the `columbus` data in `spatstat` as follows:

```
> ch <- hyperframe(window=cwindows)
> ch <- cbind.hyperframe(ch, columbus@data)
```

Then `ch` is a hyperframe containing a column of `owin` objects followed by the columns of auxiliary data.