Handling shapefiles in the `spatstat` package

Adrian Baddeley

27 January 2009

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

This information applies to `spatstat` version `1.14-10`.

1 Shapefiles

A shapefile represents a list of spatial objects — a list of points, a list of lines, or a list of polygons — 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 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 `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`.

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 line segments, 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.

SpatialPoints: if the object `x` is of class `SpatialPoints`, use `as(x, "ppp")` or `as.ppp(x)` to convert it to a spatial point pattern.

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]
```

SpatialLines: if the object `x` is of class `SpatialLines`, use `as(x, "psp")` or `as.psp(x)` to convert it to a spatial line segment pattern.

The window for the pattern can be specified as an argument `window` to these functions. Information about the connectivity of the lines is lost: a `SpatialLines` object is a list, each element of which is a sequence of line segments forming a connected curve. The connectivity is ignored when this is converted to a `psp` object.

SpatialPolygons: if the object `x` is of class `SpatialPolygons`, use `as(x, "owin")` or `as.owin(x)` to convert it to a window (object of class `"owin"`) in the `spatstat` package.

This will generate an error if the polygons in `x` intersect each other, if they are self-intersecting, or if they violate other geometrical conditions. An object of class `SpatialPolygons` is just a list 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 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.

An object x of class SpatialPointsDataFrame, SpatialLinesDataFrame or SpatialPolygonsDataFrame is effectively a list of spatial objects together with a data frame containing additional variables attached to the objects. The data frame of auxiliary data is extracted by x@data or slot(x, "data").

SpatialLinesDataFrame: if the object x is of class SpatialLinesDataFrame, type something like

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

to extract the points.

SpatialLinesDataFrame: if the object x is of class SpatialLinesDataFrame, type something like

```
> y <- as(x, "SpatialLines")
> z <- as(y, "psp")
```

to extract the line segments.

SpatialPolygonsDataFrame: if the object x is of class SpatialPolygonsDataFrame, type something like

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

to extract the polygonal region(s).