Introduction to the North Carolina SIDS data set

Roger Bivand

December 19, 2003

1 Introduction

This data set was presented first in Symons et al. (1983), analysed with reference to the spatial nature of the data in Cressie and Read (1985), expanded in Cressie and Chan (1989), and used in detail in Cressie's monograph on statistics for spatial data (1991, revised 1993). It is for the 100 counties of North Carolina, and includes counts of numbers of live births (also non-white live births) and numbers of sudden infant deaths, for the 1974–1978 and 1979–1984 periods. In Cressie and Read (1985), a listing of county neighbours based on shared boundaries (contiguity) is given, and in Cressie and Chan (1989), and in Cressie (1991, pp. 386–389), a different listing based on the criterion of distance between county seats, with a cutoff at 30 miles. The county seat location coordinates are given in miles in a local (unknown) coordinate reference system. The data are also used to exemplify a range of functions in the `S-PLUS`spatial statistics module user's manual (Kaluzny et al., 1996).

2 Getting the data into R

We will be using the **spdep** package, here version: spdep, version 0.2-7, 2003-12-19, and the **maptools** package. The data from the sources refered to above is collected in the `nc.sids` data set in **spdep**. But to map it, we also need access to data for the county boundaries for North Carolina; this has been made available in the **maptools** package in shapefile format[1]. These data are known to be geographical coordinates (longitude-latitude in decimal degrees) and are assumed to use the NAD83 datum.

```
> library(spdep)
```

```
> library(maptools)
```

The shapefile format presupposes that you have three files with extensions `*.shp`, `*.shx`, and `*.dbf`, where the first contains the geometry data, the second the spatial index, and the third the attribute data. They are required to have the same name apart from the extension, and are read using `read.shape()`. By default, this function reads in the data in all three files, although it is only given the name of the file with the geometry.

```
> sids.shp <- read.shape(system.file("shapes/sids.shp", package = "maptools"))
```

```
Shapefile Type: Polygon   # of Shapes: 100

Field name:  CNTY_  changed to:  CNTY.
```

[1] These data are taken with permission from: http://sal.agecon.uiuc.edu/datasets/sids.zip.

```
Field name:  CNTY_ID  changed to:  CNTY.ID
Field name:  CRESS_ID  changed to:  CRESS.ID
```

The imported object in R has class `Map`, and is a list with two components, `"Shapes"`, which is a list of shapes, and `"att.data"`, which is a data frame with tabular data, one row for each shape in `"Shapes"`. Let us move the attribute data to a separate data frame for convenience:

```
> sids <- sids.shp$att.data
> names(sids)
```

```
 [1] "AREA"      "PERIMETER" "CNTY."     "CNTY.ID"   "NAME"      "FIPS"
 [7] "FIPSNO"    "CRESS.ID"  "BIR74"     "SID74"     "NWBIR74"   "BIR79"
[13] "SID79"     "NWBIR79"
```

```
> sidspolys <- Map2poly(sids.shp)
> sidscents <- get.Pcent(sids.shp)
```

```
> plotpolys(sidspolys)
> points(sidscents)
```

We can examine the names of the columns of the data frame to see what it contains — in fact some of the same columns that we will be examining below, and some others which will be useful in cleaning the data set. We will similarly convert the geometry format of the `Map` object to that of a `polylist` object, which will be easier to handle. Finally, we retreive the centroids of the county polygons to use as label points. Using the `plotpolys()` function from **maptools**, we can display the polygon boundaries and centroids, shown in Figure 1.

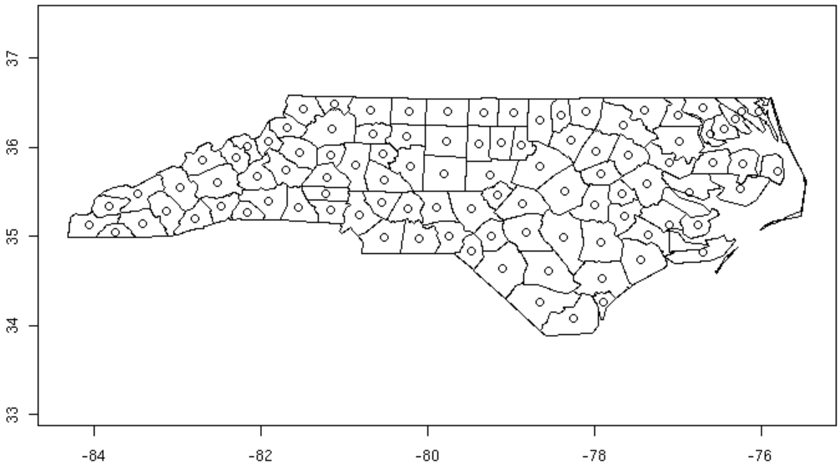

Figure 1: County boundaries and polygon centroids, North Carolina

It may be of interest to look at the structure of a polygon list member. This is made up of a two-column matrix with polygon coordinates. In general, each sub-polygon will have equal first and last coordinates to ensure closure, but this is not absolutely required. Rows in the coordinate matrix set to `NA` represent breaks between sub-polygons, and are respected by the underlying R graphics functions. The attributes contain further information about the polygon: `pstart` is a list with `from` and `to` components, which are vectors of first and last rows in the matrix for each sub-polygon in the object

— there are `nParts` elements in both `from` and `to`. `RingDir` and `ringDir` should be the same (but are not here, `ringDir` is correct, and `RingDir` is wrong!), and are computed in two different ways to determine whether each of the `nParts` sub-polygons runs clockwise or counter-clockwise. Counter-clockwise sub-polygons are "holes" in the surrounding sub-polygon. Finally, `bbox` contains the bounding box of this object. Its appearance is shown in Figure 2.

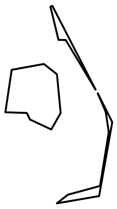

Figure 2: Plot of polygon 56 from the list of polygons.

```
> round(t(sidspolys[[56]]), 3)
```

```
        [,1]    [,2]    [,3]    [,4]    [,5]    [,6] [,7]    [,8]    [,9]
[1,] -75.783 -75.773 -75.545 -75.703 -75.741 -75.783   NA -75.891 -75.908
[2,]  36.225  36.229  35.788  36.050  36.050  36.225   NA  35.631  35.666
       [,10]   [,11]   [,12]   [,13]   [,14]   [,15]   [,16] [,17]   [,18]
[1,] -76.021 -75.988 -75.818 -75.749 -75.729 -75.779 -75.891    NA -75.491
[2,]  35.669  35.893  35.924  35.869  35.665  35.579  35.631    NA  35.670
       [,19]   [,20]   [,21]   [,22]   [,23]   [,24]   [,25]   [,26]
[1,] -75.534 -75.457 -75.526 -75.749 -75.692 -75.521 -75.475 -75.491
[2,]  35.769  35.617  35.228  35.190  35.235  35.281  35.564  35.670
attr(,"pstart")
attr(,"pstart")$from
[1]  1  8 18

attr(,"pstart")$to
[1]  6 16 26

attr(,"bbox")
[1] -76.02121  35.18983 -75.45698  36.22926
attr(,"RingDir")
[1] 1 1 1
attr(,"nParts")
[1] 3
attr(,"ringDir")
[1]  1  1 -1
```

```
> sidscents[56, ]
```

```
[1] -75.80982  35.73548
```

```
> plot(sidspolys[[56]], type = "l", asp = 1, axes = FALSE, xlab = "",
+     ylab = "")
```

3 Getting the data ready to use

We will now access the data set reproduced from Cressie and collaborators, included in **spdep**, and add the neighbour relationships used in Cressie and Chan (1989) to the background map as a graph in blue:

```
> data(nc.sids)

> plotpolys(sidspolys, border = "grey")
> plot(ncCC89.nb, sidscents, add = TRUE, col = "blue")
```

Figure 3 does not show what we wanted — it is obvious that the wrong nodes are being connected to each other. This is specifically included here because it is more the rule than the exception that spatial objects are placed in different order in different data sources (or the numbers of objects may not agree, or some objects are aggregated in one data source and not in another, the possibilities are endless).

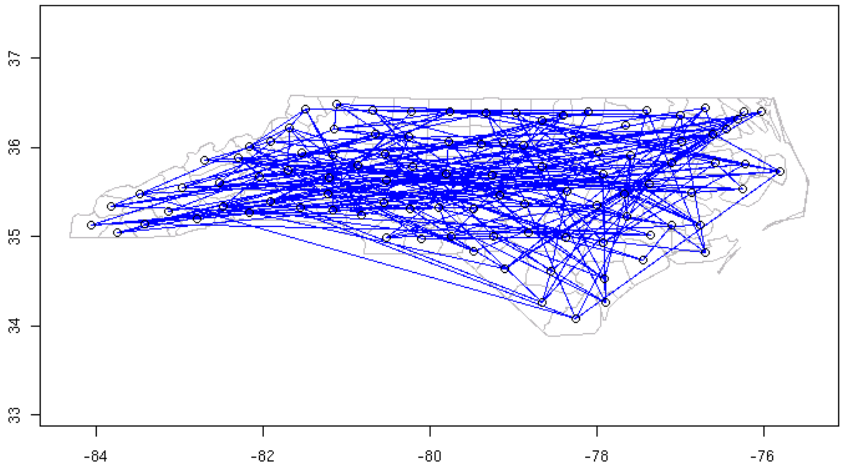

Figure 3: Overplotting shapefile boundaries with 30 mile neighbour relations as a graph (first attempt).

To see what is going on, we can list the first five rows of the `sids` data frame taken from the shapefile, and the first five rows of the `nc.sids` data frame included in **spdep** and taken directly from listings in the sources refered to above. The county ID variable is included in both sources[2], and is also the `region.id` attribute of the Cressie/Chan 30 mile neighbour relations neighbour object.

```
> sids[1:5, c(4, 5, 9)]

  CNTY.ID        NAME BIR74
1    1825        Ashe  1091
2    1827   Alleghany   487
3    1828       Surry  3188
4    1831   Currituck   508
5    1832 Northampton  1421

> nc.sids[1:5, c("CNTY.ID", "BIR74")]
```

[2] prior to R 1.9, the underscore in the DBF file feild name `CNTY_ID` is converted to a point, `CNTY.ID`.

```
          CNTY.ID BIR74
Alamance     1904  4672
Alexander    1950  1333
Alleghany    1827   487
Anson        2096  1570
Ashe         1825  1091
```

```
> attr(ncCC89.nb, "region.id")[1:5]
```

```
[1] 1904 1950 1827 2096 1825
```

We can see that the `nc.sids` data frame is ordered alphabetically by county name (as are the neighbour objects), while the `sids` data frame is ordered by ascending county IDs (as are the polygons). To resolve this, we could swap the rows of the neighbour list objects, but we would also need to change all the identification numbers of the spatial objects too, so it seems best to use a temporary file and functions for reading and writing neighbour lists, and especially for exchanging them with other software (including GeoDa).

```
> tmpGAL <- tempfile(pattern = "GAL")
> write.nb.gal(ncCC89.nb, file = tmpGAL, oldstyle = FALSE, shpfile = "sids",
+     ind = "CNTY.ID")
> CNTY.ID <- sids[, 4]
> ncCC89.2.nb <- read.gal(file = tmpGAL, region.id = CNTY.ID)
```

When `read.gal()` is given a `region.id` argument, it is used to match the incoming data, and to reorder them on-the-fly, so that, as Figure 4 shows, the re-ordered neighbour list object now conforms with the order of the polygon list object.

```
> plotpolys(sidspolys, border = "grey")
> plot(ncCC89.2.nb, sidscents, add = TRUE, col = "blue")
```

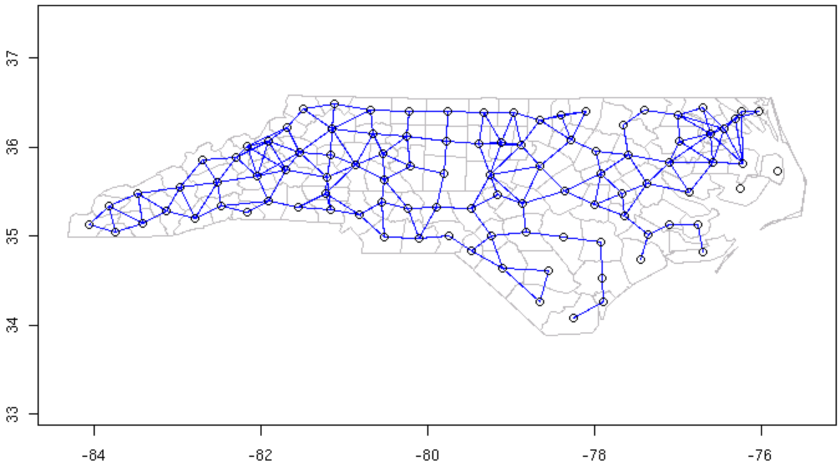

Figure 4: Overplotting shapefile boundaries with 30 mile neighbour relations as a graph (re-ordered neighbour list)

Printing the new object shows that it is a neighbour list object, with a very sparse structure — if displayed as a matrix, only 3.94% of cells would be filled. Objects of class `nb` contain a list as long as the number of counties; each component of the list is a vector with the index numbers of the neighbours of the county in question, so that the neighbours of the county with `region.id` of "1825" can be retreived by matching against the indices. More information can be obtained by using `summary()` on an `nb` object. Finally, we associate a vector of names with the neighbour list, through the `row.names` argument. The names should be unique, as with data frame row names.

```
> ncCC89.2.nb

Neighbour list object:
Number of regions: 100
Number of nonzero links: 394
Percentage nonzero weights: 3.94
Average number of links: 3.94
2 regions with no links:
2000 2099

> r.id <- attr(ncCC89.2.nb, "region.id")
> ncCC89.2.nb[[match("1825", r.id)]]

[1]  2 18 19

> r.id[ncCC89.2.nb[[match("1825", r.id)]]]

[1] 1827 1874 1880
```

The neighbour list object records neighbours by their order in relation to the list itself, so the neighbours list for the county with `region.id` "1825" are the second, eighteenth, and nineteenth in the list. We can retreive their `CNTY.ID` codes by looking them up in the `region.id` attribute.

```
> sids[card(ncCC89.2.nb) == 0, ]

    AREA PERIMETER CNTY. CNTY.ID NAME  FIPS FIPSNO CRESS.ID BIR74 SID74 NWBIR74
56 0.094     3.640  2000    2000 Dare 37055  37055       28   521     0      43
87 0.167     2.709  2099    2099 Hyde 37095  37095       48   338     0     134
   BIR79 SID79 NWBIR79
56  1059     1      73
87   427     0     169
```

We should also note that this neighbour criterion generates two counties with no neighbours, Dare and Hyde, whose county seats were more than 30 miles from their nearest neighbours. The `card()` function returns the cardinality of the neighbour set. We need to return to methods for handling no-neighbour objects later on. We will also show how new neighbours lists may be constructed in R, and compare these with those from the literature.

4 Preliminary exploration of the data (incomplete)

One of the first steps taken by Cressie and Read (1985) is to try to bring out spatial trends by dividing North Carolina up into 4 × 4 rough rectangles. Just to see how this works, let us map these rough rectangles before proceeding further (see Figure 5). We need to recall that the `nc.sids` data frame is not in the same order as the polygons.

```
> both <- factor(paste(nc.sids$L.id, nc.sids$M.id, sep = ":"))
> cols <- sample(rainbow(length(table(unclass(both)))))

> plotpolys(sidspolys, col = cols[both[order(nc.sids$CNTY.ID)]])
> legend(c(-84, -81), c(33.5, 34.6), legend = levels(both), fill = cols,
+     bty = "n", cex = 0.9, y.intersp = 0.9, ncol = 2)
```

(document to be extended in next release — terminates here to at least show how **maptools** and **spdep** can be used together).

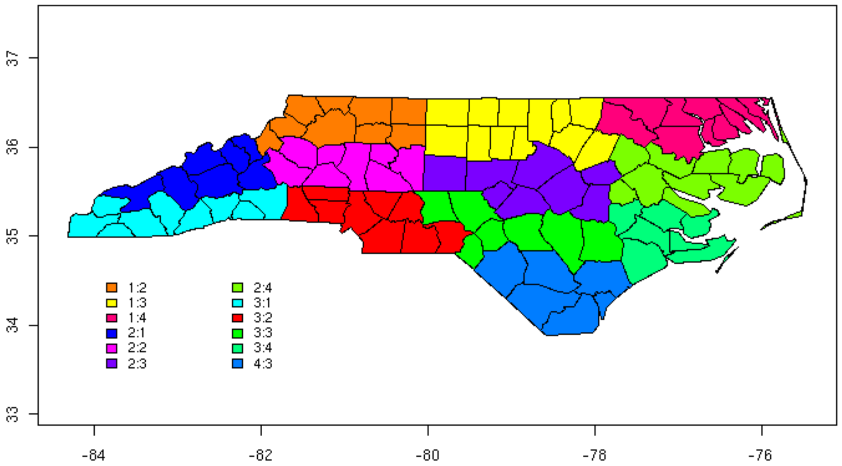

Figure 5: Rough rectangles used by Cressie and Read (1985) to bring out spatial trends.

References

Cressie, N (1991), Statistics for spatial data. New York: Wiley.

Cressie, N, Chan NH (1989), Spatial modelling of regional variables. *Journal of the American Statistical Association* 84, 393–401.

Cressie, N, Read, TRC (1985), Do sudden infant deaths come in clusters? *Statistics and Decisions* Supplement Issue 2, 333–349.

Kaluzny, SP, Vega, SC, Cardoso, TP, Shelly, AA (1996), *S-PLUSSPATIALSTATS user's manual version 1.0*. Seattle: MathSoft Inc.

Symons, MJ, Grimson, RC, Yuan, YC (1983), Clustering of rare events. *Biometrics* 39, 193–205.