Curso de verão: *Geoestatística e Tópicos de Estatística Espacial*

Aula prática, Semana 1

03/01/2007

Note

- These notes are also available in PDF format.
- A file only with the R commands extracted from this document is also available.

Introduction:
The main objectives of this Lab Session are to:

- provide some information on the usage of add-on packages in R.
- illustrate some resources available in R packages to deal with spatial data.
- point out some relevant issues when dealing with the analysis of spatial data.
- bring some points for discussion through the examples.
- motivate topics which will be discussed during this course.

To do so we will use a rather arbitrary selection of R packages related to spatial statistics. R resources for spatial statistics are an world on its own within the R project and are currently grouped as one of the *CRAN Task Views*, the so called *Spatial Task View* about which detailed information can be found at:
http://cran.r-project.org/src/contrib/Views/Spatial.html

R software and packages:
To follow the examples below you need R installed in your system and the following add-on packages: *spatstat*, *splancs*, *spatial* (which is part of the **VR** bundle), *spdep*, *geoR* and *RandomFields*.

Notice most of these packages have *dependencies*, i.e. in order to load and run they require other packages to be installed. The packages may be already installed in the computer you are using, but if you need to install/update a package (maybe somewhere else) make sure the *package dependencies* are also installed. The easier way to ensure this is to use the argument `dependencies=TRUE` when calling `install.packages()`

For instance:

```
> install.packages("spatstat", dep = T)
```

is the command used to install, with *administrator (root)* permissions, the package **spatstat** and also all the other packages listed in the **spatstat** dependency list.

Installing packages without "root" password:
The commands above assume you have full writing permissions to install packages on the system you are using. However, in some circunstances (mainly when you are not using your own computer) this may not be the case. In such cases you can do the following:

1. install the packages in a directory you have writting permission using the argment `lib`. Lets assume you are installing the package **spatstat** in a directory is called `/home/myself/Rlibs`.

   ```
   > install.packages("spatstat", dep = T, lib = "/home/myself/Rlibs")
   ```

2. add this directory to your search path for R packages with:

   ```
   > .libPaths(c("/home/myself/Rlibs", .libPaths()))
   ```

3. load the package as usual with

   ```
   > require(spatstat)
   ```

After loading a package you have access to all its functions. It is a good practice to unload the package when you finish using it, before start working with a different one, since different packages may have functions with the same name. For instance, reproducing the commands below you will receive a message a function called `Strauss` is available at both **spatial** and **spatstat** and the latter to be load will mask the function in the former. The function `search()` returns the search path for function calls in order of recedence.

```
> require(spatial)
> require(spatstat)
> search()
```

Unloading a package:

```
> detach(package:spatstat)
> detach(package:spatial)
```

Instaling the CRAN Spatial Task View:
If you are interested in installing all the packages available in the *CRAN Spatial Task View* (and their dependencies) you can proceed as follows. Notice this can be time and space consuming!

```
> install.packages("ctv")
> require(ctv)
> install.views("Spatial")
```

Example 1:
The first example uses the data `auckland` from the package **spdep** on child mortality. To load the package, the data and learn more about this data type:

```
> require(spdep)
> data(auckland)
> help(auckland)
> head(auckland)
```

There are several possibilities for visualising this data. Here we show the raw rates (per 1,000) standardised for the time period and grouped in 5 classes of our choice.

```
> auckland <- transform(auckland, rawrates = Deaths.1977.85/(9 *
+     Under.5.1981))
> brks <- c(-Inf, 2, 2.5, 3, 3.5, Inf)
> cols <- gray(seq(1, 0.2, len = length(brks) - 1))
> plot(auckpolys, col = cols[findInterval(auckland$raw * 1000,
+     brks)], forcefill = FALSE)
> legend(c(70, 90), c(70, 95), fill = cols, legend = leglabs(brks),
+     bty = "n")
```

The raw rates and other measures can be obtained with the command below where the resulting object also shows mean expected counts, standardised mortality rates and Poisson probabilities obtained using the observed and expected values (`pmap`).

```
> rates <- probmap(auckland$Deaths.1977.85, 9 * auckland$Under.5.1981)
> head(rates)
```

The commands above will produce a map of the latter and also using another color scheme.

```
> brks <- c(0, 0.05, 0.1, 0.2, 0.8, 0.9, 0.95, 1)
> cols <- terrain.colors(7)
> plot(auckpolys, col = cols[findInterval(rates$pmap, brks)], forcefill = FALSE)
> legend(c(70, 90), c(70, 95), fill = cols, legend = leglabs(brks),
+     bty = "n")
```

Note: check how the columns of the `rates` object were obtained and the interpretation of each one.

We now consider one way to smooth data of this kind set using the *empirical Bayes*. This method basically weights local and global rates with lower weights given for local rates obtained at places with small populations at risk; and higher weights for larger populations.

```
> ebglobal <- EBest(auckland$Deaths.1977.85, 9 * auckland$Under.5.1981)
> brks <- c(-Inf, 2, 2.5, 3, 3.5, Inf)
> cols <- gray(seq(1, 0.2, length = length(brks) - 1))
> plot(auckpolys, col = cols[findInterval(ebglobal$estmm * 1000,
+     brks)], forcefill = FALSE)
> legend(c(70, 90), c(70, 95), fill = cols, legend = leglabs(brks),
+     bty = "n")
```

There is yet another version of the *empirical Bayes* method which replaces the global rates by rates obtained in the neighbourhood of each unit. The example below show the results using a particular definition of neighbourhood: districts which share a common border. The neighbourhood structure is available in the object `auckland.nb`.

```
> summary(auckland.nb)
> eblocal <- EBlocal(spNamedVec("Deaths.1977.85", auckland), 9 *
+     spNamedVec("Under.5.1981", auckland), auckland.nb)
> brks <- c(-Inf, 2, 2.5, 3, 3.5, Inf)
> cols <- gray(seq(1, 0.2, length = length(brks) - 1))
> plot(auckpolys, col = cols[findInterval(eblocal$est * 1000, brks)],
+     forcefill = FALSE)
> legend(c(70, 90), c(70, 95), fill = cols, legend = leglabs(brks),
+     bty = "n")
```

Note: Compare and discuss the plots obtained in this example. You can also try to plot other variables of interest.

```
> search()
> detach(package:spdep)
> detach(package:boot)
```

Example 2:
Our second example still uses the `auckland` data set but now using another "geometry" – the data has X and Y coordinates for the districts. Therefore, we can consider a set of fixed points at which we observe one or more attributes.

Our goal here will be check in an exploratory way whether there is evidence of spatial structure in this data.

Note: which variable(s) can we use?

In what follows we will use some functions from the **geoR** package for which it is convenient to use data in the so called `geodata` format which, in its basic format, consists of coordinates and an attribute. For the commands below we use the points coordinates provided with the `auckland` data and residuals from a Poisson (glm) model.

```
> auckpois <- with(auckland, glm(Deaths.1977.85 ~ 1, offset = log(9 *
+     Under.5.1981), family = poisson))
> require(geoR)
> auckgeo <- as.geodata(cbind(auckland[, 1:2], resid(auckpois,
+     type = "pearson")))
> plot(auckgeo)
> points(auckgeo)
> plot(variog(auckgeo, max.dist = 30))
```

Note: what is this plot suggesting?

A word of caution: default options in functions must be used with care. Try the command below and see how different the output can be.

```
> plot(variog(auckgeo))
```

Example 3:
In the previous examples the "geometry" of the data was fixed – either as polygons or points and the focus of the analysis for possible modelling strategies were on the attribute(s). Now we look at another type of data where the locations themselves are of main interest.

We start with the data-set `bodmin` from the package **splancs**. Remember that more information on this data can be obtained by typing `data(bodmin)` (after loading the package **splancs**).

The initial two plots below highlight the fact that care must be taken to preserve correct scales when plotting spatial data.

```
> require(splancs)
> data(bodmin)
> plot(bodmin)
> plot(bodmin, asp = 1)
```

A nicer plot for this data can be obtained with:

```
> plot(bodmin$poly, asp = 1, ty = "l")
> points(bodmin)
```

Above we have used standard R function to produce the plots but **splancs** also has its own data formats and plotting functions.

```
> pointmap(as.points(bodmin), pch = 19)
> polymap(bodmin$poly, lwd = 2)
> pointmap(as.points(bodmin), pch = 19, add = T)
```

One initial visual summary of point data can be obtained by smoothing the density of points with a moving window over the area. The function **kernel2d** provides one way to do this.

```
> image(kernel2d(as.points(bodmin), bodmin$poly, h0 = 2, nx = 100,
+     ny = 100), asp = 1, col = terrain.colors(20))
> pointmap(as.points(bodmin), add = TRUE, pch = 19)
> polymap(bodmin$poly, add = TRUE, lwd = 2)
```

Note: try with different values for the argument `h0`.

A common starting point to analyse data of this kind is check whether the spatial pattern looks as *random*, *clustered* or *regular*. Look at the data below – what would you say about it?

```
> data(cardiff)
> plot(cardiff, asp = 1)
```

Visual inspection of the points can be difficult or even misleading. There are tools to explore such patterns and one of them is the K-function shown below. The *random* pattern is the benchmark and the function detects deviations from it using Monte Carlo simulations under this assumption.

```
> UL.khat <- Kenv.csr(length(cardiff$x), cardiff$poly, nsim = 29,
+     seq(2, 30, 2))
> plot(seq(2, 30, 2), sqrt(khat(as.points(cardiff), cardiff$poly,
+     seq(2, 30, 2))/pi) - seq(2, 30, 2), type = "l", xlab = "Splancs - polygon boundary",
+     ylab = "Estimated L", ylim = c(-1, 1.5))
> lines(seq(2, 30, 2), sqrt(UL.khat$upper/pi) - seq(2, 30, 2),
+     lty = 2)
> lines(seq(2, 30, 2), sqrt(UL.khat$lower/pi) - seq(2, 30, 2),
+     lty = 2)

> detach(package:splancs)
```

Example 4:
The package **spatstats** has an nice example on how the conclusion can be conditional to the observation window. Consider first a dataset on locations of trees mentioned by some authors.

```
> require(spatstat)
> data(redwoodfull)
> plot(redwoodfull)
```

Now, look at an expanded version of this data-set collected in a wider region.

```
> redwoodfull.extra$plot()
```

Note: you can try to explore both, the initial and expanded data with the smoothing function `kernel2d` and the K-function mentioned above.

```
> detach(package:spatstat)
```

Example 5:
In this example we illustrate usage of yet another package, the package **spatial** from the "bundle" **VR**. This was the first "spatial" package to appear in R.

The first example shows an example of a point pattern and K-function. Notice the "standardised" version of the K-function shown above is visually easier to be inspected.

```
> require(spatial)
> towns <- ppinit("towns.dat")
> par(pty = "s")
> plot(Kfn(towns, 40), type = "b")
> plot(Kfn(towns, 10), type = "b", xlab = "distance", ylab = "L(t)")
> for (i in 1:10) lines(Kfn(Psim(69), 10))
> lims <- Kenvl(10, 100, Psim(69))
> lines(lims$x, lims$l, lty = 2, col = "green")
> lines(lims$x, lims$u, lty = 2, col = "green")
> lines(Kaver(10, 25, Strauss(69, 0.5, 3.5)), col = "red")
```

From this you can see there are some functionalities implemented in more than one package. For instance, there are implementations of the K-function in **splancs**, **spatial** and **spatstat**. However the algorithms, syntax, arguments and outputs can be different. Choose you favorite!

A second example with this package illustrate how data on a attribute can be interpolated over an area. The commands below show the original data, predictions over the area and the associated std errors.

```
> require(spatial)
> require(MASS)
> require(geoR)
> data(topo, package = "MASS")
> points(as.geodata(topo))
> topo.kr <- surf.gls(2, expcov, topo, d = 0.7)
> trsurf <- trmat(topo.kr, 0, 6.5, 0, 6.5, 50)
> contour(trsurf, add = TRUE)
> prsurf <- prmat(topo.kr, 0, 6.5, 0, 6.5, 50)
> contour(prsurf, levels = seq(700, 925, 25))
> sesurf <- semat(topo.kr, 0, 6.5, 0, 6.5, 30)
> eqscplot(sesurf, type = "n")
> contour(sesurf, levels = c(22, 25), add = TRUE)

> detach(package:spatial)
```

Example 6:
Our final examples shows that several packages contain functions to simulate spatial data assuming particular models.

First consider 2 distinctive examples of simulated points patterns using functions from **spatstat**.

```
> require(spatstat)
> X <- rThomas(15, 0.2, 5)
> plot(X)
> plot(Gest(X))
> pp <- rSSI(0.05, 200)
> plot(pp)
> plot(Gest(pp))
```

And now, simulation of the values of an attribute on a fine grid covering a particular area following a Gaussian random field model.

```
> require(RandomFields)
> x <- y <- seq(0, 20, 0.1)
```

```
> f <- GaussRF(x = x, y = x, model = "stable", grid = TRUE, param = c(0,
+     4, 1, 10, 1))
> image(x, x, f, col = gray(seq(1, 0, l = 15)))
```

Exercises

1. Some packages have a build in `demo()` illustrating the package resources. For instance, load the package **spatstat** and type `demo(spatstats)` and inspect the demonstration of this package trying to guess what is going on at each stage.

2. Consider the Scottish Lip Cancer data set available at: this link.

   This data was described and analysed in Breslow and Clayton (1993) and re-visited by several authors. The columns have registers on the number of observed and expected cases lip cancer cases on 56 districts. Also available is the percentage of population employed in agriculture, fishing or forestry (AFF column) which was considered to be a relevant information, together with the lat-long coordinates for the centre of the districts.

   The aim here is to perform an analysis of this data and summarise your conclusions. Some points for discussion includes: visualisation of the data, exploratory analysis, effects of potential covariates, presence of spatial patterns and choice of modelling strategy.