1 Introduction

The meuse data set provided by package sp is a data set comprising of four heavy metals measured in the top soil in a flood plain along the river Meuse, along with a handful of covariates. The process governing heavy metal distribution seems that polluted sediment is carried by the river, and mostly deposited close to the river bank, and areas with low elevation. This document shows a geostatistical analysis of this data set. The data set was introduced by Burrough and McDonnell, 1998.

This tutorial introduced the functionality of the R package gstat, used in conjunction with package sp. Package gstat provides a wide range of univariable and multivariable geostatistical modelling, prediction and simulation functions, where package sp provides general purpose classes and methods for defining, importing/exporting and visualizing spatial data.

2 R geostatistics packages

Package gstat (Pebesma, 2004) is an R package that provides basic functionality for univariable and multivariable geostatistical analysis, including

- variogram modelling, residual variogram modelling, and cross variogram modelling using fitting of parametric models to sample variograms
- geometric anisotropy specified for each partial variogram model
- restricted maximum likelihood fitting of partial sills
- variogram and cross variogram maps
- simple, ordinary, universal and external drift (co)kriging
- (sequential) Gaussian (co)simulation equivalents for each of the kriging varieties
- indicator (co)kriging and sequential indicator (co)simulation
- kriging in a local or global neighbourhood
- block (co)kriging or simulation for each of the varieties, for rectangular or irregular blocks
Other geostatistical packages for R usually lack part of these options (e.g. block kriging, local kriging, or cokriging) but provide others: e.g. package geor and geoRglm (by Paulo Ribeiro and Ole Christensen) provide the model-based geostatistics framework described in Diggle et al. (1998), package fields (Doug Nychka and others) provides thin plate spline interpolation, covariance functions for spherical coordinates (unprojected data), and routines for spatial sampling design optimization.

3 Spatial data frames

As an example, we will look at the meuse data set, which is a regular data frame that comes with package gstat (remove the 88 from the colour strings to make a plot without alpha transparency on windows or X11):

```r
> library(sp)
> data(meuse)
> class(meuse)
[1] "data.frame"
> names(meuse)
[1] "x" "y" "cadmium" "copper" "lead" "zinc" "elev"
[8] "dist" "om" "ffreq" "soil" "lime" "landuse" "dist.m"
> coordinates(meuse) = ~x+y
> class(meuse)
[1] "SpatialPointsDataFrame"
attr("package")
[1] "sp"
> summary(meuse)

Object of class SpatialPointsDataFrame
Coordinates:
  min  max
x 178605 181390
y 329714 333611
Is projected: NA
proj4string : [NA]
Number of points: 155
Data attributes: cadmium copper lead zinc
                Min. : 0.200 Min. : 14.00 Min. : 37.0 Min. : 113.0
                1st Qu.: 0.800 1st Qu.: 23.00 1st Qu.: 72.5 1st Qu.: 198.0
                Median : 2.100 Median : 31.00 Median :123.0 Median : 326.0
                Mean : 3.246 Mean : 40.32 Mean :153.4 Mean : 469.7
                3rd Qu.: 3.850 3rd Qu.: 49.50 3rd Qu.:207.0 3rd Qu.: 674.5
                Max. :18.100 Max. :128.00 Max. :654.0 Max. :1839.0
```
The image contains a table and a code snippet related to a dataset. The table includes columns for `elev`, `dist`, `om`, `ffreq`, `soil`, and `lime`, with summary statistics such as min, 1st quartile, median, mean, 3rd quartile, and max values for each column.

The code snippet includes function calls to manipulate the `meuse` dataset, which appears to be related to environmental data, such as zinc concentrations and land use. The `coordinates` function is used to extract spatial coordinates, and the `bubble` function is used to visualize the data with different colored dots representing varying zinc concentrations (ppm).
and note the following:

1. the function `coordinates`, when assigned (i.e. on the left-hand side of an `=` or `<-` sign), promotes the `data.frame` `meuse` into a `SpatialPointsDataFrame`, which knows about its spatial coordinates; coordinates may be specified by a formula, a character vector, or a numeric matrix or data frame with the actual coordinates.

2. the function `coordinates`, when not assigned, retrieves the spatial coordinates from a `SpatialPointsDataFrame`.

3. the two plotting functions used, `plot` and `bubble` assume that the $x$- and $y$-axis are the spatial coordinates.

4 Spatial data on a regular grid

```r
> data(meuse.grid)
> summary(meuse.grid)

             x         y      part.a      part.b
Min. :178460 Min. :329620 Min. :0.0000 Min. :0.0000
1st Qu.:179420 1st Qu.:330460 1st Qu.:0.0000 1st Qu.:0.0000
Median :179985  Median :331220 Median :0.0000  Median :1.0000
3rd Qu.:180580 3rd Qu.:332140 3rd Qu.:1.0000 3rd Qu.:1.0000
Max. :181540  Max. :333740  Max. :1.0000  Max. :1.0000
dist soil ffreq
Min. :0.0000  1:1665  1: 779
1st Qu.:0.1193 2:1084  2:1335
Median :0.2715  3: 354  3: 989
Mean :0.2971
3rd Qu.:0.4402
Max. :0.9926
```

```r
> class(meuse.grid)
[1] "data.frame"
```

```r
> coordinates(meuse.grid) = ~x+y
> class(meuse.grid)

```

```r
> gridded(meuse.grid) = TRUE
> class(meuse.grid)

```
> image(meuse.grid["dist"])
> title("distance to river (red = 0)")
> library(gstat)
> zinc.idw = idw(zinc~1, meuse, meuse.grid)

[Inverse distance weighted interpolation]

> class(zinc.idw)

[1] "SpatialPixelsDataFrame"
attr(,"package")
[1] "sp"

> spplot(zinc.idw["var1.pred"], main = "zinc inverse distance weighted interpolations")
If you compare the bubble plot of zinc measurements with the map with distances to the river, it becomes evident that the larger concentrations are measured at locations close to the river. This relationship can be linearized by log-transforming the zinc concentrations, and taking the square root of distance to the river:

```r
> plot(log(zinc)~sqrt(dist), meuse)
> abline(lm(log(zinc)~sqrt(dist), meuse))
```
5 Variograms

Variograms are calculated using the function `variogram`, which takes a formula as its first argument: `log(zinc) ~ 1` means that we assume a constant trend for the variable log(zinc).

```r
> lzn.vgm = variogram(log(zinc) ~ 1, meuse)
> lzn.vgm

np   dist  gamma dir.hor dir.ver  id
     57  79.29244 0.1234479 0 0 var1
     299 163.97367 0.2162185 0 0 var1
     419  267.36483 0.3027859 0 0 var1
     457  372.73542 0.4121448 0 0 var1
     547  478.47670 0.4634128 0 0 var1
     533  585.34058 0.5646933 0 0 var1
     574  693.14526 0.5689683 0 0 var1
     564  796.18365 0.6186769 0 0 var1
     589  903.14650 0.6471479 0 0 var1
     543 1011.29177 0.6915705 0 0 var1
     500 1117.86235 0.7033984 0 0 var1
     477 1221.32810 0.6038770 0 0 var1
     452 1329.16407 0.6517158 0 0 var1
     457 1437.25620 0.5665318 0 0 var1
     415 1543.20248 0.5748227 0 0 var1
```

7
> lzn.fit = fit.variogram(lzn.vgm, model = vgm(1, "Sph", 900, 1))
> lzn.fit

model psill range
1  Nug 0.05066243 0.0000
2  Sph 0.59060780 897.0209

> plot(lzn.vgm, lzn.fit)

Instead of the constant mean, denoted by ~1, we can specify a mean function, e.g. using ~sqrt(dist) as a predictor variable:

> lznr.vgm = variogram(log(zinc)~sqrt(dist), meuse)
> lznr.fit = fit.variogram(lznr.vgm, model = vgm(1, "Exp", 300, 1))
> lznr.fit

model psill range
1  Nug 0.05712231 0.0000
2  Exp 0.17641559 340.3201

> plot(lznr.vgm, lznr.fit)
In this case, the variogram of residuals with respect to a fitted mean function are shown. Residuals were calculated using ordinary least squares.

6 Kriging

> lzn.kriged = krige(log(zinc)~1, meuse, meuse.grid, model = lzn.fit)
[using ordinary kriging]

> spplot(lzn.kriged["vari.pred"])
7 Conditional simulation

```r
> lzn.condsim = krige(log(zinc)~1, meuse, meuse.grid, model = lzn.fit,
+ nmax = 30, nsim = 4)

drawing 4 GLS realisations of beta...
[using conditional Gaussian simulation]

> spplot(lzn.condsim, main = "four conditional simulations")
```
For UK/residuals:

```r
> lzn.condsim2 = krigge(log(zinc)~sqrt(dist), meuse, meuse.grid, model = lznr.fit,
+     nmax = 30, nsim = 4)

drawing 4 GLS realisations of beta...
[using conditional Gaussian simulation]

> spplot(lzn.condsim2, main = "four UK conditional simulations")
```
8 Directional variograms

The following command calculates a directional sample variogram, where directions are binned by direction angle alone. For two point pairs, $Z(s)$ and $Z(s+h)$, the separation vector is $h$, and it has a direction. Here, we will classify directions into four direction intervals:

```r
> lzn.dir = variogram(log(zinc)~1, meuse, alpha = c(0, 45, 90, 135))
> lzndir.fit = vgm(.59, "Sph", 1200, .05, anis = c(45, .4))
> plot(lzn.dir, lzndir.fit, as.table = TRUE)
```
Looking at directions between 180 and 360 degrees will repeat the image shown above, because the variogram is a symmetric measure: $(Z(s) - Z(s + h))^2 = (Z(s + h) - Z(s))^2$.

The first plot gives the variogram in the zero direction, which is North; 90 degrees is East. By default, point pairs are assigned to the directional variogram panel with their nearest direction, so North contains everything between -22.5 and 22.5 degrees (North-West to North-East). After classifying by direction, point pairs are binned by separation distance class, as is done in the usual omnidirectional case.

In the figure, the partial sill, nugget and model type of the model are equal to those of the omnidirectional model fitted above; the range is that in the direction with the largest range (45°), and the anisotropy ratio, the range in the 135 direction and the range in the 45 direction, estimated “by eye” by comparing the 45 and 135 degrees sample variograms. Gstat does not fit anisotropy parameters automatically.

We do not claim that the model fitted here is “best” in some way; in order to get to a better model we may want to look at more directions, other directions (e.g. try alpha = c(22, 67, 112, 157) ), and to variogram maps (see below). More elaborate approaches may use directions in three dimensions, and want to further control the direction tolerance (which may be set such that direction intervals overlap).

For the residual variogram from the linear regression model using sqrt(dist) as covariate, the directional dependence is much less obvious; the fitted model here is the fitted isotropic model (equal in all directions).

```R
> lznr.dir = variogram(log(zinc)~sqrt(dist), meuse, alpha = c(0, 45, 90, 135))
> plot(lznr.dir, lznr.fit, as.table = TRUE)
```
Another means of looking at directional dependence in semivariograms is obtained by looking at variogram maps. Instead of classifying point pairs $Z(s)$ and $Z(s + h)$ by direction and distance class separately, we can classify them jointly. If $h = \{x, y\}$ be the two-dimensional coordinates of the separation vector, in the variogram map the semivariance contribution of each point pair $(Z(s) - Z(s + h))^2$ is attributed to the grid cell in which $h$ lies. The map is centered around $(0, 0)$, as $h$ is geographical distance rather than geographical location. Cutoff and width correspond to some extent to map extent and cell size; the semivariance map is point symmetric around $(0, 0)$, as $\gamma(h) = \gamma(-h)$.

```r
> vgm.map = variogram(log(zinc) ~ sqrt(dist), meuse, cutoff = 1500, width = 100, 
+ map = TRUE)
> plot(vgm.map, threshold = 5)
```
The threshold assures that only semivariogram map values based on at least 5 point pairs are shown, removing too noisy estimation.

10 Cross variography

Fitting a linear model of coregionalization.

```r
> g = gstat(NULL, "log(zn)", log(zinc)~sqrt(dist), meuse)
> g = gstat(g, "log(cd)", log(cadmium)~sqrt(dist), meuse)
> g = gstat(g, "log(pb)", log(lead)~sqrt(dist), meuse)
> g = gstat(g, "log(cu)", log(copper)~sqrt(dist), meuse)
> v = variogram(g)
> g = gstat(g, model = vgm(1, "Exp", 300, 1), fill.all = TRUE)
> g.fit = fit.lmc(v, g)
> g.fit
```

data:
log(zn) : formula = log(zinc)^~sqrt(dist) ; data dim = 155 x 12
log(cd) : formula = log(cadmium)^~sqrt(dist) ; data dim = 155 x 12
log(pb) : formula = log(lead)^~sqrt(dist) ; data dim = 155 x 12
log(cu) : formula = log(copper)^~sqrt(dist) ; data dim = 155 x 12

variograms:

<table>
<thead>
<tr>
<th>model</th>
<th>psill</th>
<th>range</th>
</tr>
</thead>
<tbody>
<tr>
<td>log(zn)[1]</td>
<td>Nug 0.05141798</td>
<td>0</td>
</tr>
<tr>
<td>log(zn)[2]</td>
<td>Exp 0.17556219</td>
<td>300</td>
</tr>
<tr>
<td>log(cd)[1]</td>
<td>Nug 0.39996573</td>
<td>0</td>
</tr>
<tr>
<td>log(cd)[2]</td>
<td>Exp 0.47893816</td>
<td>300</td>
</tr>
</tbody>
</table>
\log(pb)[1] \quad \text{Nug} \quad 0.04770893 \quad 0 \\
\log(pb)[2] \quad \text{Exp} \quad 0.21323027 \quad 300 \\
\log(cu)[1] \quad \text{Nug} \quad 0.04577523 \quad 0 \\
\log(cu)[2] \quad \text{Exp} \quad 0.07827374 \quad 300 \\
\log(zn).\log(cd)[1] \quad \text{Nug} \quad 0.09190848 \quad 0 \\
\log(zn).\log(cd)[2] \quad \text{Exp} \quad 0.24542024 \quad 300 \\
\log(zn).\log(pb)[1] \quad \text{Nug} \quad 0.04528367 \quad 0 \\
\log(zn).\log(pb)[2] \quad \text{Exp} \quad 0.18407011 \quad 300 \\
\log(cd).\log(pb)[1] \quad \text{Nug} \quad 0.06425412 \quad 0 \\
\log(cd).\log(pb)[2] \quad \text{Exp} \quad 0.25525359 \quad 300 \\
\log(zn).\log(cu)[1] \quad \text{Nug} \quad 0.02912806 \quad 0 \\
\log(zn).\log(cu)[2] \quad \text{Exp} \quad 0.10438748 \quad 300 \\
\log(cd).\log(cu)[1] \quad \text{Nug} \quad 0.09441635 \quad 0 \\
\log(cd).\log(cu)[2] \quad \text{Exp} \quad 0.13073936 \quad 300 \\
\log(pb).\log(cu)[1] \quad \text{Nug} \quad 0.02369778 \quad 0 \\
\log(pb).\log(cu)[2] \quad \text{Exp} \quad 0.10267516 \quad 300

> plot(v, g.fit)
> vgm.map = variogram(g, cutoff = 1500, width = 100, map = TRUE)
> plot(vgm.map, threshold = 5, col.regions = bpy.colors(), xlab = "", ylab = "")
References


