Chapter 8
Interpolation and Geostatistics

8.1 Introduction

Geostatistical data are data that could in principle be measured anywhere, but that typically come as measurements at a limited number of observation locations: think of gold grades in an ore body or particulate matter in air samples. The pattern of observation *locations* is usually not of primary interest, as it often results from considerations ranging from economical and physical constraints to being 'representative' or random sampling varieties. The interest is usually in inference of aspects of the variable that have not been measured such as maps of the estimated values, exceedance probabilities or estimates of aggregates over given regions, or inference of the process that generated the data. Other problems include monitoring network optimisation: where should new observations be located or which observation locations should be removed such that the operational value of the monitoring network is maximised.

Typical spatial problems where geostatistics are used are the following:

- The estimation of ore grades over mineable units, based on drill hole data
- Interpolation of environmental variables from sample or monitoring network data (e.g. air quality, soil pollution, ground water head, hydraulic conductivity)
- Interpolation of physical or chemical variables from sample data
- Estimation of spatial averages from continuous, spatially correlated data

In this chapter we use the Meuse data set used by Burrough and McDonnell (1998). The notation we use follows mostly that of Christensen (1991), as this text most closely links geostatistics to linear model theory. Good texts on geostatistics are Chilès and Delfiner (2012), Christensen (1991), Cressie (1993), and Journel and Huijbregts (1978). More applied texts are, for example Isaaks and Strivastava (1989), Goovaerts (1997), and Deutsch and Journel (1992).

R.S. Bivand et al., *Applied Spatial Data Analysis with R*, Use R! 10,
DOI 10.1007/978-1-4614-7618-4_8,

Geostatistics deals with the analysis of random fields $Z(s)$, with Z random and s the non-random spatial index. Typically, at a limited number of sometimes arbitrarily chosen sample locations, measurements on Z are available, and prediction (interpolation) of Z is required at non-observed locations s_0, or the mean of Z is required over a specific region B_0. Geostatistical analysis involves estimation and modelling of spatial correlation (covariance or semivariance), and evaluating whether simplifying assumptions such as stationarity can be justified or need refinement. More advanced topics include the conditional simulation of $Z(s)$, for example over locations on a grid, and model-based inference, which propagates uncertainty of correlation parameters through spatial predictions or simulations.

Much of this chapter will deal with package **gstat**, because it offers the widest functionality in the geostatistics curriculum for R: it covers variogram cloud diagnostics, variogram modelling, everything from global simple kriging to local universal cokriging, multivariate geostatistics, block kriging, indicator and Gaussian conditional simulation, and many combinations. Other R packages that provide additional geostatistical functionality are mentioned where relevant, and discussed at the end of this chapter.

8.2 Exploratory Data Analysis

Spatial exploratory data analysis starts with the plotting of maps with a measured variable. To express the observed value, we can use colour or symbol size:

```
> library(lattice)
> library(sp)
> data(meuse)
> coordinates(meuse) <- c("x", "y")
> spplot(meuse, "zinc", do.log = T, colorkey = TRUE)
> bubble(meuse, "zinc", do.log = T, key.space = "bottom")
```

to produce plots with information similar to that of Fig. 3.8.

The evident structure here is that zinc concentration is larger close to the river Meuse banks. In case of an evident spatial trend, such as the relation between top soil zinc concentration and distance to the river here, we can also plot maps with fitted values and with residuals (Cleveland, 1993), as shown in Fig. 8.1, obtained by

```
> xyplot(log(zinc) ~ sqrt(dist), as.data.frame(meuse))
> zn.lm <- lm(log(zinc) ~ sqrt(dist), meuse)
> meuse$fitted.s <- predict(zn.lm, meuse) - mean(predict(zn.lm,
+     meuse))
> meuse$residuals <- residuals(zn.lm)
> spplot(meuse, c("fitted.s", "residuals"))
```

where the formula `y` $\sim$ `x` indicates dependency of `y` on `x`. This figure reveals that although the trend removes a large part of the variability, the residuals do

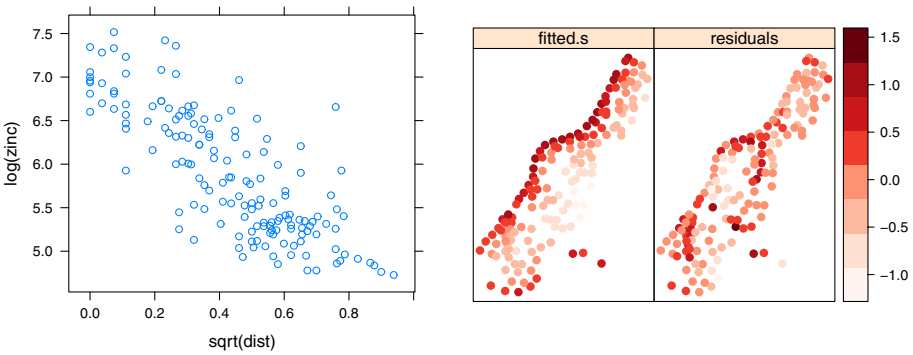

Fig. 8.1 Zinc as a function of distance to river (*left*), and fitted-residual maps (fitted.s: mean subtracted) for the linear regression model of log zinc and square-root transformed distance to the river

not appear to behave as spatially unstructured or white noise: residuals with a similar value occur regularly close to another. More exploratory analysis will take place when we further analyse these data in the context of geostatistical models; first we deal with simple, non-geostatistical interpolation approaches.

8.3 Non-geostatistical Interpolation Methods

Usually, interpolation is done on a regular grid. For the Meuse data set, coordinates of points on a regular grid are already defined in the `meuse.grid` data.frame, and are converted into a `SpatialPixelsDataFrame` by

```
> data(meuse.grid)
> coordinates(meuse.grid) <- c("x", "y")
> meuse.grid <- as(meuse.grid, "SpatialPixelsDataFrame")
```

Alternatively, we could interpolate to individual points, sets of irregularly distributed points, or to averages over square or irregular areas (Sect. 8.5.6).

8.3.1 Inverse Distance Weighted Interpolation

Inverse distance-based weighted interpolation (IDW) computes a weighted average,

$$\hat{Z}(s_0) = \frac{\sum_{i=1}^{n} w(s_i) Z(s_i)}{\sum_{i=1}^{n} w(s_i)},$$

where weights for observations are computed according to their distance to the interpolation location,

$$w(s_i) = ||s_i - s_0||^{-p},$$

with $||\cdot||$ indicating Euclidean distance and p an inverse distance weighting power, defaulting to 2. If s_0 coincides with an observation location, the observed value is returned to avoid infinite weights.

The inverse distance power determines the degree to which the nearer point(s) are preferred over more distant points; for large values IDW converges to the one-nearest-neighbour interpolation. It can be tuned, for example using cross validation (Sect. 8.7.1). IDW can also be used within local search neighbourhoods (Sect. 8.5.5).

Because the **spatstat** package also offers a function called `idw`, we disambiguate the two, should **spatstat** have been loaded into the session workspace, by calling the **gstat** function using the `::` operator to choose the desired `idw`:

```
> library(gstat)
> idw.out <- gstat::idw(zinc ~ 1, meuse, meuse.grid, idp = 2.5)

[inverse distance weighted interpolation]

> as.data.frame(idw.out)[1:5, ]

       x      y var1.pred var1.var
1 181180 333740  701.9621       NA
2 181140 333700  799.9616       NA
3 181180 333700  723.5780       NA
4 181220 333700  655.3131       NA
5 181100 333660  942.0218       NA
```

The output variable is called `var1.pred`, and the `var1.var` values are `NA` because inverse distance does not provide prediction error variances.

Inverse distance interpolation results usually in maps that are very similar to kriged maps when a variogram with no or a small nugget is used. In contrast to kriging, by only considering distances to the prediction location it ignores the spatial configuration of observations; this may lead to undesired effects if the observation locations are strongly clustered. Another difference is that weights are guaranteed to be between 0 and 1, resulting in interpolated values never outside the range of observed values.

8.3.2 Linear Regression

For spatial prediction using simple linear models, we can use the R function `lm`:

```
> zn.lm <- lm(log(zinc) ~ sqrt(dist), meuse)
> meuse.grid$pred <- predict(zn.lm, meuse.grid)
> meuse.grid$se.fit <- predict(zn.lm, meuse.grid, se.fit = TRUE)$se.fit
```

Alternatively, the `predict` method used here can provide the prediction or confidence intervals for a given confidence level. Alternatively, we can use the function `krige` in **gstat** for this,

```
> meuse.lm <- krige(log(zinc) ~ sqrt(dist), meuse, meuse.grid)

[ordinary or weighted least squares prediction]
```

that in this case does not *krige* as no variogram is specified, but uses linear regression.

Used in this form, the result is identical to that of `lm`. However, it can also be used to predict with regression models that are refitted withinlocal neighbourhoods around a prediction location (Sect. 8.5.5) or provide mean predicted values for spatial areas (Sect. 8.5.6). The variance it returns is the prediction error variance when predicting for points or the estimation error variance when used for blocks.

A special form of linear regression is obtained when polynomials of spatial coordinates are used for predictors, for example for a second-order polynomial

```
> meuse.tr2 <- krige(log(zinc) ~ 1, meuse, meuse.grid,
+     degree = 2)

[ordinary or weighted least squares prediction]
```

This form is called trend surface analysis.

It is possible to use `lm` for trend surface analysis, for example for the second-order trend with a formula using `I` to treat powers and products 'as is':

```
> lm(log(zinc) ~ I(x^2) + I(y^2) + I(x * y) + x + y, meuse)
```

or the short form

```
> lm(log(zinc) ~ poly(x, y, degree = 2), meuse)
```

In the first form `lm` does not standardise coordinates, which often yields huge numbers when powered. The second form does standardise coordinates in such a way that it cannot be used in a subsequent `predict` call with different coordinate ranges. Trend surface fitting is highly sensitive to outlying observations. Another place to look for trend surface analysis is function `surf.ls` in package **spatial**.

8.4 Estimating Spatial Correlation: The Variogram

In geostatistics the spatial correlation is modelled by the variogram instead of a correlogram or covariogram, largely for historical reasons. Here, the word variogram will be used synonymously with semivariogram. The variogram plots semivariance as a function of distance.

In standard statistical problems, correlation can be estimated from a scatterplot, when several data pairs $\{x, y\}$ are available. The spatial correlation between two observations of a variable $z(s)$ at locations s_1 and s_2 cannot be estimated, as only a single pair is available. To estimate spatial correlation from observational data, we therefore need to make stationarity assumptions before we can make any progress. One commonly used form of stationarity is *intrinsic stationarity*, which assumes that the process that generated the samples is a random function $Z(s)$ composed of a mean and residual

$$Z(s) = m + e(s), \tag{8.1}$$

with a constant mean

$$\mathrm{E}(Z(s)) = m \tag{8.2}$$

and a variogram defined as

$$\gamma(h) = \frac{1}{2}\mathrm{E}(Z(s) - Z(s+h))^2. \tag{8.3}$$

Under this assumption, we basically state that the variance of Z is constant, and that spatial correlation of Z does not depend on location s, but only on separation distance h. Then, we can form *multiple* pairs $\{z(s_i), z(s_j)\}$ that have (nearly) identical separation vectors $h = s_i - s_j$ and estimate correlation from them. If we further assume *isotropy*, which is direction independence of semivariance, we can replace the vector h with its length, $||h||$.

Under this assumption, the variogram can be estimated from N_h sample data pairs $z(s_i),\ z(s_i + h)$ for a number of distances (or distance intervals) $\tilde{h}_j$ by

$$\hat{\gamma}(\tilde{h}_j) = \frac{1}{2N_h}\sum_{i=1}^{N_h}(Z(s_i) - Z(s_i + h))^2, \quad \forall h \in \tilde{h}_j \tag{8.4}$$

and this estimate is called the *sample* variogram.

A wider class of models is obtained when the mean varies spatially, and can, for example be modelled as a linear function of known predictors $X_j(s)$, as in

$$Z(s) = \sum_{j=0}^{p} X_j(s)\beta_j + e(s) = X\beta + e(s), \tag{8.5}$$

with $X_j(s)$ the known spatial regressors and β_j unknown regression coefficients, usually containing an intercept for which $X_0(s) \equiv 1$. The $X_j(s)$ form the columns of the $n \times (p+1)$ design matrix X, β is the column vector with $p+1$ unknown coefficients.

For varying mean models, stationarity properties refer to the residual $e(s)$, and the sample variogram needs to be computed from estimated residuals.

8.4.1 Exploratory Variogram Analysis

A simple way to acknowledge that spatial correlation is present or not is to make scatter plots of pairs $Z(s_i)$ and $Z(s_j)$, grouped according to their separation distance $h_{ij} = ||s_i - s_j||$. This is done for the `meuse` data set in Fig. 8.2, by

```
> hscat(log(zinc) ~ 1, meuse, (0:9) * 100)
```

where the strip texts indicate the distance classes, and sample correlations are shown in each panel.

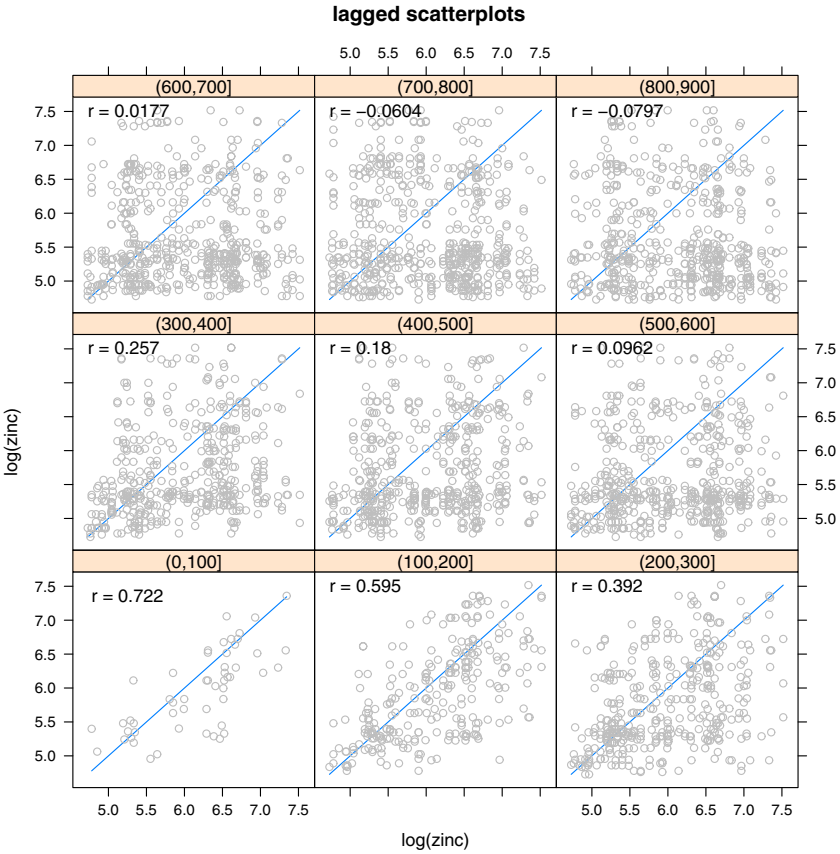

Fig. 8.2 Lagged scatter plot for the log-zinc data in the `meuse` data set

A second way to explore spatial correlation is by plotting the variogram and the variogram cloud. The variogram cloud is obtained by plotting all

possible squared differences of observation pairs $(Z(s_i)-Z(s_j))^2$ against their separation distance h_{ij}. One such variogram cloud, obtained by

```
> hscat(log(zinc) ~ 1, meuse, (0:9) * 100)
```

is plotted in Fig. 8.3 (top). The plot shows a lot of scatter, as could be expected: when $Z(s)$ follows a Gaussian distribution, $(Z(s_i)-Z(s_j))^2$ follows a $\chi^2(1)$ distribution. It does show, however, some increase of maximum values for distances increasing up to 1,000 m.

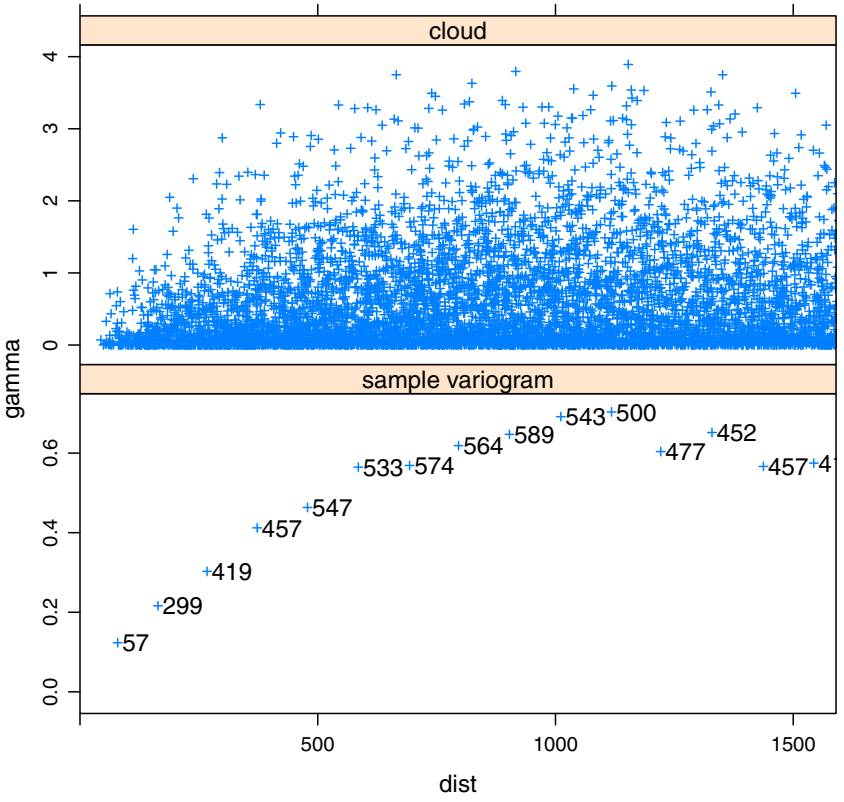

Fig. 8.3 Variogram cloud (*top*) and sample variogram (*bottom*) for log-zinc data; numbers next to symbols refer to the value N_h in (8.4)

Essentially, the sample variogram plot of (8.4) obtained by

```
> library(gstat)
> variogram(log(zinc) ~ 1, meuse, cloud = TRUE)
```

is nothing but a plot of averages of semivariogram cloud values over distance intervals h; it is shown in Fig. 8.3, bottom. It smooths the variation in the

variogram cloud and provides an estimate of semivariance (8.3), although Stein (1999) discourages this approach.

The `~ 1` defines a single constant predictor, leading to a spatially constant mean coefficient, in accordance with (8.2); see p. 26 for a presentation of formula objects.

As any point in the variogram cloud refers to a pair of points in the data set, the variogram cloud can point us to areas with unusual high or low variability. To do that, we need to select a subset of variogram cloud points. In

```
> sel <- plot(variogram(zinc ~ 1, meuse, cloud = TRUE),
+     digitize = TRUE)
> plot(sel, meuse)
```

the user is asked to digitise an area in the variogram cloud plot after the first command. The second command plots the selected point pairs on the observations map. Figure 8.4 shows the output plots of such a session. The point pairs with largest semivariance at short distances were selected, because they indicate the areas with the strongest gradients. The map shows that these areas are not spread randomly: they connect the maximum values closest to the Meuse river with values usually more inland. This can be an indication of non-stationarity or of anisotropy. Log-transformation or detrending may remove this effect, as we see later.

In case of outlying observations, extreme variogram cloud values are easily identified to find the outliers. These may need removal, or else robust measures for the sample variogram can be computed by passing the logical argument `cressie=TRUE` to the `variogram` function call (Cressie, 1993).

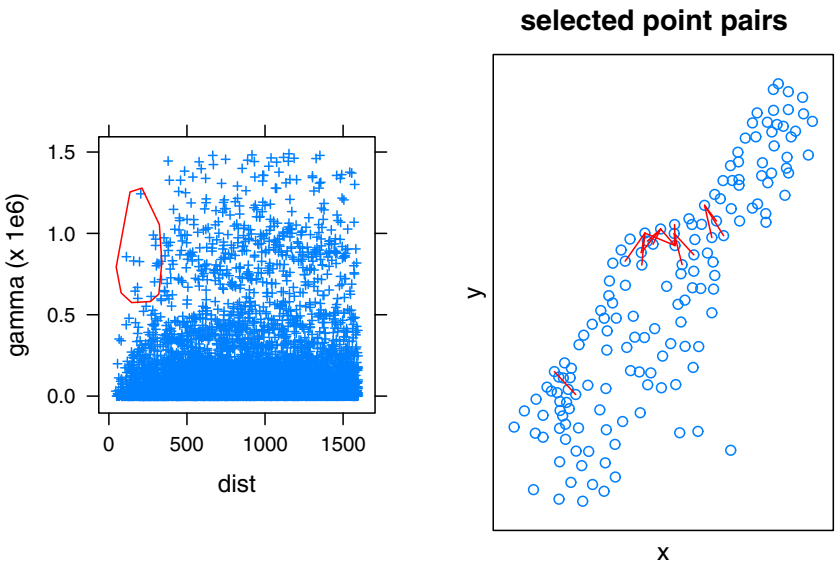

Fig. 8.4 Interactively selected point pairs on the variogram cloud (*left*), and map of selected point pairs (*right*)

A sample variogram $\hat{\gamma}(h)$ always contains a signal that results from the true variogram $\gamma(h)$ and a sampling error, due to the fact that N_h and s are not infinite. To verify whether an increase in semivariance with distance *could* possibly be attributed to chance, we can compute variograms from the same data, after randomly re-assigning measurements to spatial locations. If the sample variogram falls within the (say 95 %) range of these random variograms, complete spatial randomness of the underlying process *may be* a plausible hypothesis. Figure 8.5 shows an example of such a plot for the log zinc data; here the hypothesis of absence of spatial correlation seems unlikely. In general, however, concluding or even assuming that an underlying process is *completely* spatially uncorrelated is quite unrealistic for real, natural processes. A common case is that the spatial correlation is difficult to infer from sample data, because of their distribution, sample size, or spatial configuration. In certain cases spatial correlation is nearly absent.

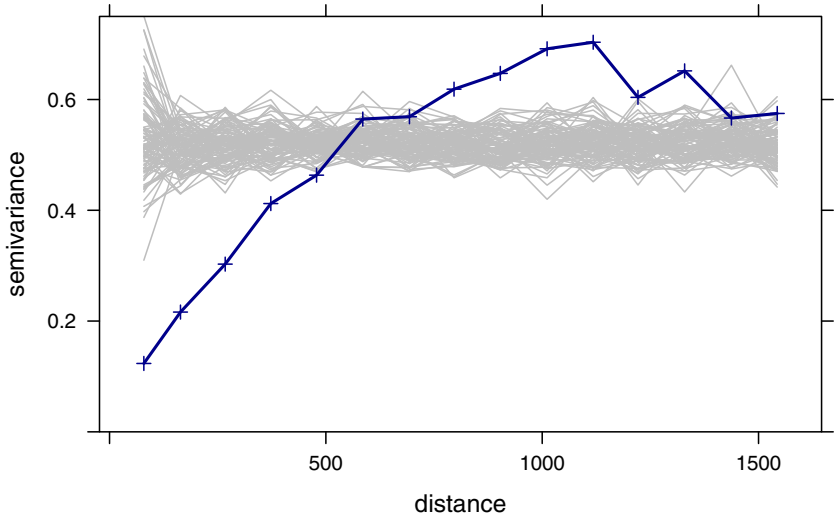

Fig. 8.5 Sample variogram (*bold*) compared to 100 variograms for randomly re-allocated data (*grey lines*)

8.4.2 Cutoff, Lag Width, Direction Dependence

Although the command

```
> plot(variogram(log(zinc) ~ 1, meuse))
```

simply computes and plots the sample variogram, it does make a number of decisions by default. It decides that direction is ignored: point pairs are

merged on the basis of distance, not direction. An alternative is, for example to look in four different angles, as in

```
> plot(variogram(log(zinc) ~ 1, meuse, alpha = c(0, 45,
+     90, 135)))
```

see Fig. 8.7. Directions are now divided over their principal direction, e.g., any point pair between 22.5° and 67.5° is used for the 45° panel. You might want to split into a finer direction subdivision, for example passing `alpha = seq(0,170,10)`, but then the noise component of resulting sample variograms will increase, as the number of point pairs for each separate estimate decreases.

A similar issue is the cutoff distance, which is the maximum distance up to which point pairs are considered and the width of distance interval over which point pairs are averaged in bins.

The default value **gstat** uses for the cutoff value is one third of the largest diagonal of the bounding box (or cube) of the data. Just as for time series data autocorrelations are never computed for lags farther than half the series length, there is little point in computing semivariances for long distances other than mere curiosity: wild oscillations usually show up that reveal little about the process under study. Good reasons to decrease the cutoff may be when a local prediction method is foreseen, and only semivariances up to a rather small distance are required. In this case, the modelling effort, and hence the computing of sample variograms should be limited to this distance (e.g. twice the radius of the planned search neighbourhood).

For the interval width, **gstat** uses a default of the cutoff value divided by 15. Usually, these default values will result in some initial overview of the spatial correlation. Choosing a smaller interval width will result in more detail, as more estimates of $\gamma(h)$ appear, but also in estimates with more noise, as N_h inevitably decreases. It should be noted that apparent local fluctuations of consecutive $\hat{\gamma}(h)$ values may still be attributed to sampling error. The errors $\hat{\gamma}(h_i) - \gamma(h_i)$ and $\hat{\gamma}(h_j) - \gamma(h_j)$ will be correlated, because $\gamma(\hat{h}_i)$ and $\gamma(\hat{h}_j)$ usually share a large number of common points used to form pairs.

The default cutoff and interval width values may not be appropriate at all, and can be overridden, for example by

```
> plot(variogram(log(zinc) ~ 1, meuse, cutoff = 1000, width = 50))
```

The distance vector does not have to be cut in regular intervals; one can specify each interval by

```
> variogram(log(zinc) ~ 1, meuse, boundaries = c(0, 50,
+     100, seq(250, 1500, 250)))
```

which is especially useful for data sets that have much information on short distance variability: it allows one to zoom in on the short distance variogram without revealing irrelevant details for the longer distances.

8.4.3 Variogram Modelling

The variogram is often used for spatial prediction (interpolation) or simulation of the observed process based on point observations. To ensure that predictions are associated with non-negative prediction variances, the matrix with semivariance values between all observation points and any possible prediction point needs to be non-negative definite. For this, simply plugging in sample variogram values from (8.4) is not sufficient. One common way is to infer a parametric variogram *model* from the data. A non-parametric way, using smoothing and cutting off negative frequencies in the spectral domain, is given in Yao and Journel (1998); it will not be discussed here.

The traditional way of finding a suitable variogram model is to fit a parametric model to the sample variogram (8.4). An overview of the basic variogram models available in **gstat** is obtained by

```
> show.vgms()
> show.vgms(model = "Mat", kappa.range = c(0.1, 0.2, 0.5,
+     1, 2, 5, 10), max = 10)
```

where the second command gives an overview of various models in the Matérn class.

In **gstat**, valid variogram models are constructed by using one or combinations of two or more basic variogram models. Variogram models are derived from `data.frame` objects, and are built as follows:

```
> vgm(1, "Sph", 300)

  model psill range
1   Sph     1   300

> vgm(1, "Sph", 300, 0.5)

  model psill range
1   Nug   0.5     0
2   Sph   1.0   300

> v1 <- vgm(1, "Sph", 300, 0.5)
> v2 <- vgm(0.8, "Sph", 800, add.to = v1)
> v2

  model psill range
1   Nug   0.5     0
2   Sph   1.0   300
3   Sph   0.8   800

> vgm(0.5, "Nug", 0)

  model psill range
1   Nug   0.5     0
```

and so on. Each component (row) has a model type ('Nug', 'Sph', ...), followed by a partial sill (the vertical extent of the model component) and a

range parameter (the horizontal extent). Nugget variance can be defined in two ways, because it is almost always present. It reflects usually measurement error and/or micro-variability. Note that **gstat** uses range *parameters*, for example for the exponential model with partial sill c and range parameter a

$$\gamma(h) = c(1 - \mathrm{e}^{-h/a}).$$

This implies that for this particular model the *practical range*, the value at which this model reaches 95 % of its asymptotic value, is $3a$; for the Gaussian model the practical range is $\sqrt{3}a$. A list of model types is obtained by

```
> vgm()
```

```
   short                                      long
1    Nug                               Nug (nugget)
2    Exp                          Exp (exponential)
3    Sph                            Sph (spherical)
4    Gau                              Gau (gaussian)
5    Exc                Exclass (Exponential class)
6    Mat                                 Mat (Matern)
7    Ste Mat (Matern, M. Stein's parameterization)
8    Cir                               Cir (circular)
9    Lin                                 Lin (linear)
10   Bes                                 Bes (bessel)
11   Pen                       Pen (pentaspherical)
12   Per                               Per (periodic)
13   Hol                                  Hol (hole)
14   Log                             Log (logarithmic)
15   Pow                                  Pow (power)
16   Spl                                Spl (spline)
17   Leg                               Leg (Legendre)
18   Err                    Err (Measurement error)
19   Int                             Int (Intercept)
```

Not all of these models are equally useful, in practice. Most practical studies have so far used exponential, spherical, Gaussian, Matérn, or power models, with or without a nugget, or a combination of those.

For weighted least squares fitting a variogram model to the sample variogram (Cressie, 1985), we need to take several steps:

1. Choose a suitable model (such as exponential, ...), with or without nugget
2. Choose suitable initial values for partial sill(s), range(s), and possibly nugget (Fig. 8.6)
3. Fit this model, using one of the fitting criteria.

For the variogram obtained by

```
> v <- variogram(log(zinc) ~ 1, meuse)
> plot(v)
```

and shown in Fig. 8.6, the spherical model looks like a reasonable choice. Initial values for the variogram fit are needed for `fit.variogram`, because

for the spherical model (and many other models) fitting the range parameter involves non-linear regression. The following fit works:

```
> fit.variogram(v, vgm(1, "Sph", 800, 1))

  model      psill    range
1   Nug 0.05065923   0.0000
2   Sph 0.59060463 896.9976
```

but if we choose initial values too far off from reasonable values, as in

```
> fit.variogram(v, vgm(1, "Sph", 10, 1))

Warning: singular model in variogram fit
  model psill range
1   Nug     1     0
2   Sph     1    10
```

the fit will not succeed. To stop execution in an automated fitting script, a construct like

```
> v.fit <- fit.variogram(v, vgm(1, "Sph", 10, 1))
> if (attr(v.fit, "singular")) stop("singular fit")
```

will halt the script on this fitting problem.

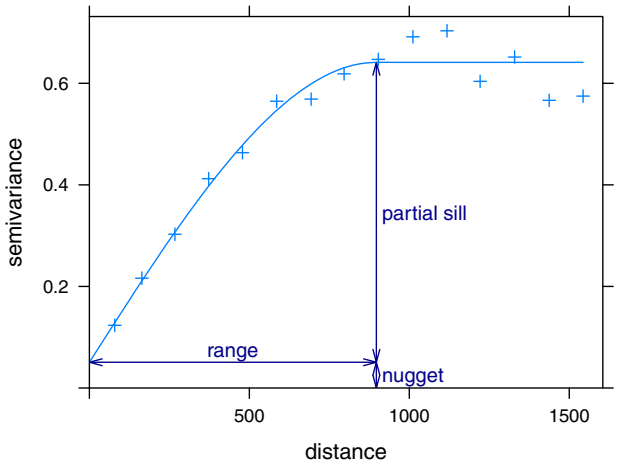

Fig. 8.6 Sample variogram (*plus*) and fitted model (*line*); *blue arrows* indicate the model parameter values

The fitting method uses non-linear regression to fit the coefficients. For this, a weighted sum of square errors $\sum_{j=1}^{p} w_j(\gamma(h) - \hat{\gamma}(h))^2$, with $\gamma(h)$ the value according to the parametric model, is minimised. The optimisation routine alternates the following two steps until convergence: (i) a direct fit over the partial sills, and (ii) non-linear optimising of the range parameter(s) given the last fit of partial sills. The minimised criterion is available as

Table 8.1 Values for argument `fit.method` in function `fit.variogram`

`fit.method`	Weight
1	N_j
2	$N_j/\{\gamma(h_j)\}^2$
6	1
7	N_j/h_j^2

```
> attr(v.fit, "SSErr")
[1] 9.011194e-06
```

Different options for the weights w_j are given in Table 8.1, the default value chosen by **gstat** is 7. Two things should be noted: (i) for option 2, the weights change after each iteration, which may confuse the optimisation routine, and (ii) for the linear variogram with no nugget, option 7 is equivalent to option 2. Option 7 is default as it seems to work in many cases; it will, however, give rise to spurious fits when a sample semivariance estimate for distance (very close to) zero gives rise to an almost infinite weight. This may happen when duplicate observations are available.

An alternative approach to fitting variograms is by visual fitting, the so-called eyeball fit. Package **geoR** provides a graphical user interface for interactively adjusting the parameters:

```
> library(geoR)
> v.eye <- eyefit(variog(as.geodata(meuse["zinc"]), max.dist = 1500))
> ve.fit <- as.vgm.variomodel(v.eye[[1]])
```

The last function converts the model saved in `v.eye` to a form readable by **gstat**.

Typically, visual fitting will minimise $|\gamma(h) - \hat{\gamma}(h)|$ with emphasis on short distance/small $\gamma(h)$ values, as opposed to a weighted squared difference, used by most numerical fitting. An argument to prefer visual fitting over numerical fitting may be that the person who fits has knowledge that goes beyond the information in the data. This may for instance be related to information about the nugget effect, which may be hard to infer from data when sample locations are regularly spread. Information may be borrowed from other studies or derived from measurement error characteristics for a specific device. In that case, one could, however, also consider partial fitting, by keeping, for example the nugget to a fixed value.

Partial fitting of variogram coefficients can be done with **gstat**. Suppose we know for some reason that the partial sill for the nugget model (i.e. the nugget variance) *is* 0.06, and we want to fit the remaining parameters, then this is done by

```
> fit.variogram(v, vgm(1, "Sph", 800, 0.06), fit.sills = c(FALSE,
+     TRUE))
  model     psill    range
1   Nug 0.0600000   0.0000
2   Sph 0.5845836 923.0066
```

Alternatively, the range parameter(s) can be fixed using argument Data set!Meuse bank `fit.ranges`.

Maximum likelihood fitting of variogram models does not need the sample variogram as intermediate form, as it fits a model directly to a quadratic form of the data, that is the variogram cloud. REML (restricted maximum likelihood) fitting of only partial sills, not of ranges, can be done using **gstat** function `fit.variogram.reml`:

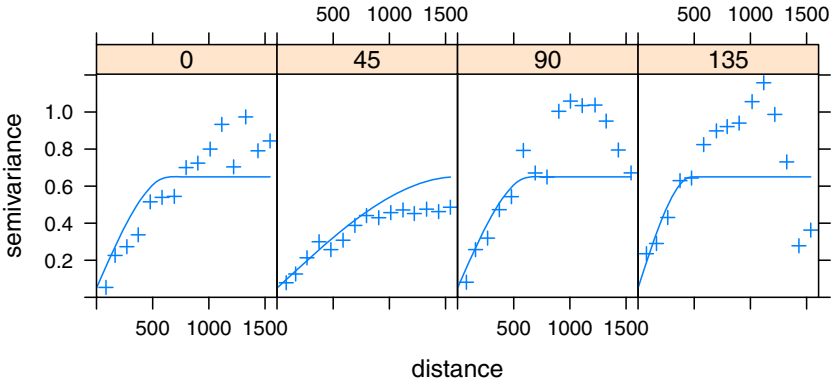

Fig. 8.7 Directional sample variogram (*plus*) and fitted model (*line*), for four directions (0 is North, 90 is East)

```
> fit.variogram.reml(log(zinc) ~ 1, meuse, model = vgm(0.6,
+     "Sph", 800, 0.06))

  model     psill range
1   Nug 0.0201109     0
2   Sph 0.5711620   800
```

Maximum likelihood or restricted maximum likelihood fitting of variogram models, including the range parameters, can be done using function `likfit` in package **geoR**, or with function `fitvario` in package **RandomFields**. Maximum likelihood fitting is optimal under the assumption of a Gaussian random field, and can be very time consuming for larger data sets.

8.4.4 Anisotropy

Anisotropy may be modelled by defining a range *ellipse* instead of a circular or spherical range. In the following example

```
> v.dir <- variogram(log(zinc) ~ 1, meuse, alpha = (0:3) *
+     45)
> v.anis <- vgm(0.6, "Sph", 1600, 0.05, anis = c(45, 0.3))

> plot(v.dir, v.anis)
```

the result of which is shown in Fig. 8.7, for four main directions. The fitted model has a range in the principal direction (45°, NE) of 1,600, and of $0.3 \times 1{,}600 = 480$ in the minor direction (135°).

When more measurement information is available, one may consider plotting a *variogram map*, as in

```
> plot(variogram(log(zinc) ~ 1, meuse, map = TRUE, cutoff = 1000,
+     width = 100))
```

which bins h vectors in square grid cells over x and y, meaning that distance and direction are shown in much more detail. Help is available for the plotting function `plot.variogramMap`.

Package **gstat** does not provide automatic fitting of anisotropy parameters. Function `likfit` in **geoR** does, by using (restricted) maximum likelihood.

8.4.5 Multivariable Variogram Modelling

We use the term multivariable geostatistics here for the case where multiple dependent spatial variables are analysed jointly. The case where the trend of a single dependent variable contains more than a constant only is not called multivariable in this sense, and will be treated in Sect. 8.5.

The main tool for estimating semivariances between different variables is the cross variogram, defined for collocated[1] data as

$$\gamma_{ij}(h) = \frac{1}{2}\mathrm{E}[(Z_i(s) - Z_i(s+h))(Z_j(s) - Z_j(s+h))]$$

and for non-collocated data as

$$\gamma_{ij}(h) = \frac{1}{2}\mathrm{E}[((Z_i(s) - m_i) - (Z_j(s) - m_j))^2],$$

with m_i and m_j the means of the respective variables. Sample cross variograms are the obvious sums over the available pairs or cross pairs, in the line of (8.4).

As multivariable analysis may involve numerous variables, we need to start organising the available information. For that reason, we collect all the observation data specifications in a `gstat` object, created by the function `gstat`. This function does nothing else than ordering (and actually, copying) information needed later in a single object. Consider the following definitions of four heavy metals:

```
> g <- gstat(NULL, "logCd", log(cadmium) ~ 1, meuse)
> g <- gstat(g, "logCu", log(copper) ~ 1, meuse)
> g <- gstat(g, "logPb", log(lead) ~ 1, meuse)
```

[1] Each observation location has all variables measured.

```
> g <- gstat(g, "logZn", log(zinc) ~ 1, meuse)
> g
data:
logCd : formula = log(cadmium)~~1 ; data dim = 155 x 14
logCu : formula = log(copper)~~1 ; data dim = 155 x 14
logPb : formula = log(lead)~~1 ; data dim = 155 x 14
logZn : formula = log(zinc)~~1 ; data dim = 155 x 14
> vm <- variogram(g)
> vm.fit <- fit.lmc(vm, g, vgm(1, "Sph", 800, 1))
> plot(vm, vm.fit)
```

the plot of which is shown in Fig. 8.8. By default, `variogram` when passing a `gstat` object computes all direct and cross variograms, but this can be turned off. The function `fit.lmc` fits a linear model of coregionalization, which is a particular model that needs to have identical model components (here nugget, and spherical with range 800), and needs to have positive definite partial sill matrices, to ensure non-negative prediction variances when used for spatial prediction (cokriging).

As the variograms in Fig. 8.8 indicate, the variables have a strong cross correlation. Because these variables are collocated, we could compute direct correlations:

```
> cor(as.data.frame(meuse)[c("cadmium", "copper", "lead",
+     "zinc")])
          cadmium    copper      lead      zinc
cadmium 1.0000000 0.9254499 0.7989466 0.9162139
copper  0.9254499 1.0000000 0.8183069 0.9082695
lead    0.7989466 0.8183069 1.0000000 0.9546913
zinc    0.9162139 0.9082695 0.9546913 1.0000000
```

which confirm this, but ignore spatial components. For non-collocated data, the direct correlations may be hard to compute.

The `fit.lmc` function fits positive definite coefficient matrices by first fitting models individually (while fixing the ranges) and then replacing non-positive definite coefficient matrices by their nearest positive definite approximation, taking out components that have a negative eigenvalue. When eigenvalues of exactly zero occur, a small value may have to be added to the direct variogram sill parameters; use the `correct.diagonal` argument for this.

Variables do not need to have a constant mean but can have a trend function specified, as explained in Sect. 8.4.6.

8.4.6 Residual Variogram Modelling

Residual variograms are calculated by default when a more complex model for the trend is used, for example as in

```
> variogram(log(zinc) ~ sqrt(dist), meuse)
```

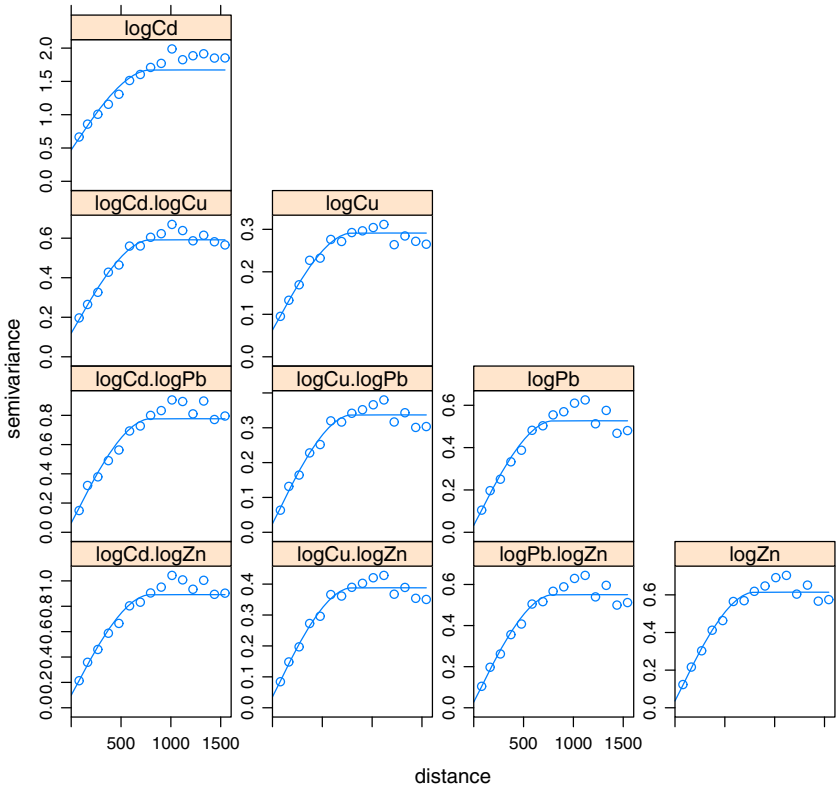

Fig. 8.8 Direct variograms (diagonal) and cross variograms (off-diagonal) along with fitted linear model of coregionalization (—)

where the trend is simple linear (Fig. 8.1), for example reworking (8.5) to

$$\log(Z(s)) = \beta_0 + \sqrt{D(s)}\beta_1 + e(s),$$

with $D(s)$ the distance to the river. For defining trends, the full range of R formulas can be used: the right-hand side may contain factors, in which case trends are calculated with respect to the factor level means, and may contain interactions of all sorts; see p. 26 for explanation on SSformula syntax.

By default, the residuals **gstat** uses are ordinary least squares residuals (i.e. regular regression residuals), meaning that for the sake of estimating the trend, observations are considered independent. To honour a dependence structure present, generalised least squares residuals can be calculated instead. For this, a variogram model to define the covariance structure is needed. In the following example

```
> f <- log(zinc) ~ sqrt(dist)
> vt <- variogram(f, meuse)
```

```
> vt.fit <- fit.variogram(vt, vgm(1, "Exp", 300, 1))
> vt.fit

  model      psill    range
1   Nug 0.05712231   0.0000
2   Exp 0.17641559 340.3201

> g.wls <- gstat(NULL, "log-zinc", f, meuse, model = vt.fit,
+     set = list(gls = 1))
> (variogram(g.wls)$gamma - vt$gamma)/mean(vt$gamma)

 [1]  1.133887e-05 -6.800894e-05 -1.588582e-04 -2.520913e-04
 [5] -5.461007e-05 -1.257573e-04  2.560629e-04  1.509185e-04
 [9]  4.812184e-07 -5.292472e-05 -2.998868e-04  2.169712e-04
[13] -1.771773e-04  1.872195e-04  3.095021e-05
```

it is clear that the difference between the two approaches is marginal, but this does not need to be the case in other examples.

For multivariable analysis, **gstat** objects can be formed where the trend structure can be specified uniquely for each variable. If multivariable residuals are calculated using weighted least squares, this is done on a per-variable basis, ignoring cross correlations for trend estimation.

8.5 Spatial Prediction

Spatial prediction refers to the prediction of unknown quantities $Z(s_0)$, based on sample data $Z(s_i)$ and assumptions regarding the form of the trend of Z and its variance and spatial correlation.

Suppose we can write the trend as a linear regression function, as in (8.5). If the predictor values for s_0 are available in the $1 \times p$ row-vector $x(s_0)$, V is the covariance matrix of $Z(s)$ and v the covariance vector of $Z(s)$ and $Z(s_0)$, then the best linear unbiased predictor of $Z(s_0)$ is

$$\hat{Z}(s_0) = x(s_0)\hat{\beta} + v'V^{-1}(Z(s) - X\hat{\beta}), \tag{8.6}$$

with $\hat{\beta} = (X'V^{-1}X)^{-1}\,X'V^{-1}Z(s)$ the generalized least squares estimate of the trend coefficients and where X' is the transpose of the design matrix X. The predictor consists of an estimated mean value for location s_0, plus a weighted mean of the residuals from the mean function, with weights $v'V^{-1}$, known as the *simple kriging weights*.

The predictor (8.6) has prediction error variance

$$\sigma^2(s_0) = \sigma_0^2 - v'V^{-1}v + \delta(X'V^{-1}X)^{-1}\delta', \tag{8.7}$$

where σ_0^2 is $\mathrm{var}(Z(s_0))$, or the variance of the Z process, and where $\delta = x(s_0) - v'V^{-1}X$. The term $v'V^{-1}v$ is zero if v is zero, that is if all observations are uncorrelated with $Z(s_0)$, and equals σ_0^2 when s_0 is identical to

an observation location. The third term of (8.7) is the contribution of the estimation error $\text{var}(\hat{\beta} - \beta) = (X'V^{-1}X)^{-1}$ to the prediction (8.6): it is zero if s_0 is an observation location, and increases, for example when $x(s_0)$ is more distant from X, as when we extrapolate in the space of X.

8.5.1 Universal, Ordinary, and Simple Kriging

The instances of this best linear unbiased prediction method with the number of predictors $p > 0$ are usually called *universal kriging*. Sometimes the term *kriging with external drift* is used for the case where $p = 1$ and X does not include coordinates.

A special case is that of (8.2), for which $p = 0$ and $X_0 \equiv 1$. The corresponding prediction is called *ordinary kriging*.

Simple kriging is obtained when, for whatever reason, β is a priori assumed to be known. The known β can then be substituted for $\hat{\beta}$ in (8.6). The simple kriging variance is obtained by omitting the third term, which is associated with the estimation error of $\hat{\beta}$ in (8.7).

Applying these techniques is much more straightforward than this complicated jargon suggests, as an example will show:

```
> lz.sk <- krige(log(zinc) ~ 1, meuse, meuse.grid, v.fit,
+     beta = 5.9)

[using simple kriging]

> lz.ok <- krige(log(zinc) ~ 1, meuse, meuse.grid, v.fit)

[using ordinary kriging]

> lz.uk <- krige(log(zinc) ~ sqrt(dist), meuse, meuse.grid,
+     vt.fit)

[using universal kriging]
```

Clearly, the `krige` command chooses the kriging method itself, depending on the information it is provided with: are trend coefficients given? is the trend constant or more complex? How this is done is shown in the decision tree of Fig. 8.9.

8.5.2 Multivariable Prediction: Cokriging

The kriging predictions equations can be simply extended to obtain multivariable prediction equations, see, for example Hoef and Cressie (1993) and Pebesma (2004). The general idea is that multiple variables may be cross correlated, meaning that they exhibit not only autocorrelation but that the

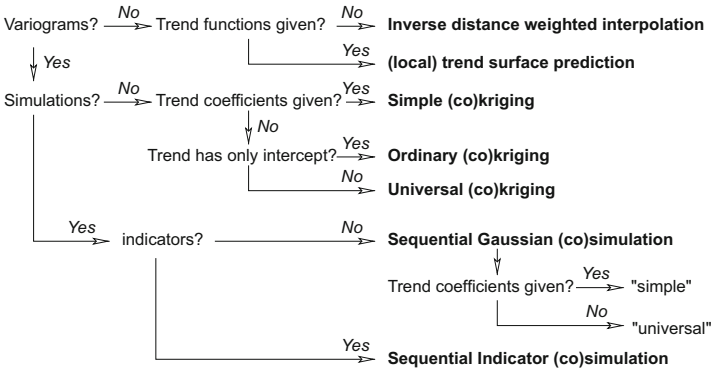

Fig. 8.9 Decision tree for the gstat `predict` method

spatial variability of variable A is correlated with variable B, and can therefore be used for its prediction, and vice versa. Typically, both variables are assumed to be measured on a limited set of locations, and the interpolation addresses unmeasured locations.

The technique is not limited to two variables. For each prediction location, multivariable prediction for q variables yields a $q \times 1$ *vector* with a prediction for each variable, and a $q \times q$ matrix with prediction error variances and covariances from which we can obtain the error correlations:

```
> cok.maps <- predict(vm.fit, meuse.grid)

Linear Model of Coregionalization found. Good.
[using ordinary cokriging]

> names(cok.maps)

 [1] "logCd.pred"       "logCd.var"        "logCu.pred"
 [4] "logCu.var"        "logPb.pred"       "logPb.var"
 [7] "logZn.pred"       "logZn.var"        "cov.logCd.logCu"
[10] "cov.logCd.logPb" "cov.logCu.logPb" "cov.logCd.logZn"
[13] "cov.logCu.logZn" "cov.logPb.logZn"
```

Here, only the unique matrix elements are stored; to get an overview of the prediction error variance and covariances, a utility function wrapping `spplot` is available; the output of

```
> spplot.vcov(cok.maps)
```

is given in Fig. 8.10.

Before the cokriging starts, **gstat** reports success in finding a Linear Model of Coregionalization (LMC). This is good, as it will assure non-negative cokriging variances. If this is not the case, for example because the ranges differ,

```
> vm2.fit <- vm.fit
> vm2.fit$model[[3]]$range = c(0, 900)
> predict(vm2.fit, meuse.grid)
```

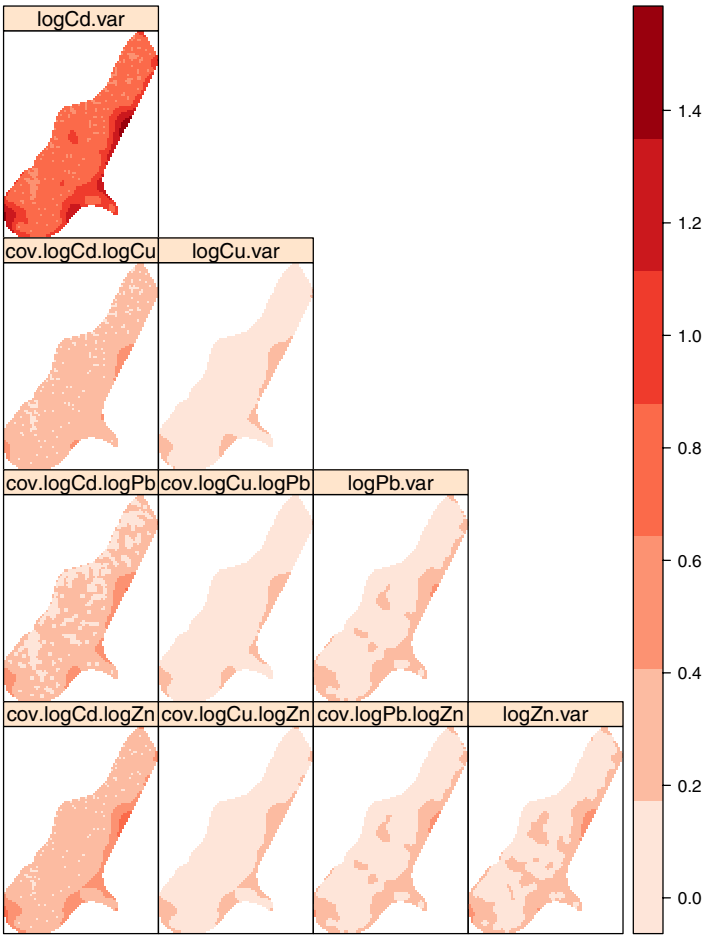

Fig. 8.10 Cokriging variances (diagonal) and covariances (off-diagonal)

will stop with an error message. Stopping on this check can be avoided by

```
> vm2.fit$set <- list(nocheck = 1)
> x <- predict(vm2.fit, meuse.grid)

Warning: No Intrinsic Correlation or Linear Model of Coregionalization
 found
Reason: ranges differ
Now checking for Cauchy-Schwartz inequalities:
variogram(var0,var1) passed Cauchy-Schwartz
variogram(var0,var2) passed Cauchy-Schwartz
variogram(var1,var2) passed Cauchy-Schwartz
variogram(var0,var3) passed Cauchy-Schwartz
variogram(var1,var3) passed Cauchy-Schwartz
variogram(var2,var3) passed Cauchy-Schwartz
[using ordinary cokriging]
```

```
> names(as.data.frame(x))

 [1] "x"                "y"                "logCd.pred"
 [4] "logCd.var"        "logCu.pred"       "logCu.var"
 [7] "logPb.pred"       "logPb.var"        "logZn.pred"
[10] "logZn.var"        "cov.logCd.logCu"  "cov.logCd.logPb"
[13] "cov.logCu.logPb"  "cov.logCd.logZn"  "cov.logCu.logZn"
[16] "cov.logPb.logZn"

> any(as.data.frame(x)[c(2, 4, 6, 8)] < 0)

[1] FALSE
```

which does check for pairwise Cauchy-Schwartz inequalities, that is $|\gamma_{ij}(h)| \leq \sqrt{\gamma_i(h)\gamma_j(h)}$, but will not stop on violations. Note that this latter check is not sufficient to guarantee positive variances. The final check confirms that we actually did not obtain any negative variances, for this particular case.

8.5.3 Collocated Cokriging

Collocated cokriging is a special case of cokriging, where a secondary variable is available at all prediction locations, and instead of choosing all observations of the secondary variable or those in a local neighbourhood, we restrict the secondary variable search neighbourhood to this single value on the prediction location. For instance, consider `log(zinc)` as primary and `dist` as secondary variable:

```
> g.cc <- gstat(NULL, "log.zinc", log(zinc) ~ 1, meuse,
+     model = v.fit)
> meuse.grid$distn <- meuse.grid$dist - mean(meuse.grid$dist) +
+     mean(log(meuse$zinc))
> vd.fit <- v.fit
> vov <- var(meuse.grid$distn)/var(log(meuse$zinc))
> vd.fit$psill <- v.fit$psill * vov
> g.cc <- gstat(g.cc, "distn", distn ~ 1, meuse.grid, nmax = 1,
+     model = vd.fit, merge = c("log.zinc", "distn"))
> vx.fit <- v.fit
> vx.fit$psill <- sqrt(v.fit$psill * vd.fit$psill) * cor(meuse$dist,
+     log(meuse$zinc))
> g.cc <- gstat(g.cc, c("log.zinc", "distn"), model = vx.fit)
> x <- predict(g.cc, meuse.grid)

Intrinsic Correlation found. Good.
[using ordinary cokriging]
```

Figure 8.11 shows the predicted maps using ordinary kriging, collocated cokriging and universal cokriging, using `log(zinc)` $\sim$ `sqrt(dist)` as trend.

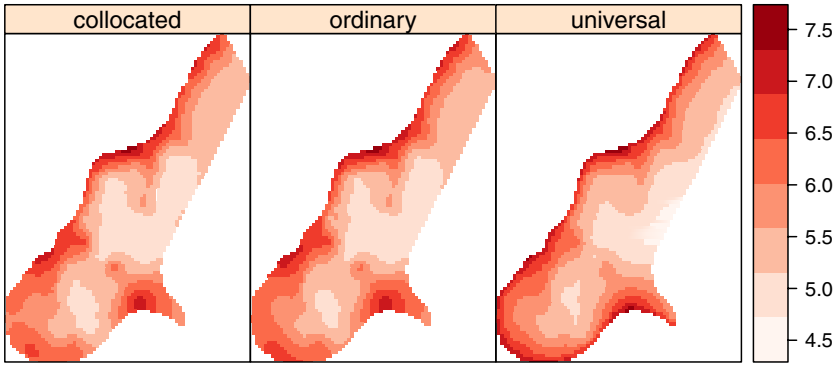

Fig. 8.11 Predictions for collocated cokriging, ordinary kriging and universal kriging

8.5.4 Cokriging Contrasts

Cokriging error covariances can be of value when we want to compute functions of multiple predictions. Suppose Z_1 is measured on time 1, and Z_2 on time 2, and both are non-collocated. When we want to estimate the change $Z_2 - Z_1$, we can use the estimates for both moments. For the prediction error of this difference, in addition to the prediction error variances for $\hat{Z}_1$ and $\hat{Z}_2$ we need the prediction error covariance. The function `get.contr` helps computing the predicted value and prediction error variance for *any* linear combination (contrast) in a set of predictors, obtained by cokriging. A demo in **gstat**,

```
> demo(pcb)
```

gives a full space-time cokriging example that shows how time trends can be estimated for PCB-138 concentration in sea floor sediment, from four consecutive five-yearly rounds of monitoring, using universal cokriging.

8.5.5 Kriging in a Local Neighbourhood

By default, all spatial predictions method provided by **gstat** use all available observations for each prediction. In many cases, it is more convenient to use only the data in the neighbourhood of the prediction location. The reasons for this may be statistical or computational. Statistical reasons include that the hypothesis of constant mean or mean function should apply locally, or that the assumed knowledge of the variogram is only valid up to a small distance. Computational issues may involve both memory and speed: kriging for n data requires solving an $n \times n$ system. For large n (say more than 1,000) this may be too slow, and discarding anything but the closest say 100 observations may not result in notable different predictions.

It should be noted that for global kriging the matrix V needs to be decomposed only once, after which the result is re-used for each prediction location to obtain $V^{-1}v$. Decomposing a linear system of equations is an $O(n^2)$ operation, solving another system $O(n)$. Therefore, if a neighbourhood size is chosen slightly smaller than the global neighbourhood, the computation time may even increase, compared to using a global neighbourhood.

Neighbourhoods in **gstat** are defined by passing the arguments `nmax`, `nmin`, and/or `maxdist` to functions like `predict`, `krige`, or `gstat`. Arguments `nmax` and `nmin` define a neighbourhood size in terms of number of nearest points, `maxdist` specifies a circular search range to select point. They may be combined: when less than `nmin` points are found in a search radius, a missing value is generated.

For finding neighbourhood selections fast, **gstat** first builds a PR bucket quadtree, or for three-dimensional data octree search index (Hjaltason and Samet, 1995). With this index it finds any neighbourhood selection with only a small number of distance evaluations.

8.5.6 Change of Support: Block Kriging

Despite the fact that no measurement can ever be taken on something that has size zero, in geostatistics, by default observations $Z(s_i)$ are treated as being observed on point location. Kriging a value with a physical size equal to that of the observations is called *point kriging*. In contrast, *block kriging* predicts averages of larger areas or volumes. The term block kriging originates from mining, where early geostatistics was developed (Journel and Huijbregts, 1978). In mines, predictions based on bore hole data had to be made for *mineable units*, which tended to have a block shape. *Change of support* occurs when predictions are made for a larger physical support based on small physical support observations. There is no particular reason why the larger support needs to have a rectangular shape, but it is common.

Besides the practical relevance to the mining industry, a consideration in many environmental applications has been that point kriging usually exhibits large prediction errors. This is due to the larger variability in the observations. When predicting averages over larger areas, much of the variability (i.e. that *within* the blocks) averages out and block mean values have lower prediction errors, while still revealing spatial patterns if the blocks are not too large. In environmental problems, legislation may be related to means or medians over larger areas, rather than to point values.

Block kriging (or other forms of prediction for blocks) can be obtained by **gstat** in three ways:

1. For regular blocks, by specifying a block size
2. For irregular but constant ‘blocks’, by specifying points that discretise the irregular form

3. For blocks or areas of varying size, by passing an object of class `SpatialPolygons` to the `newdata` argument (i.e. replacing `meuse.grid`)

Ordinary block kriging for blocks of size 40×40 is simply obtained by

```
> lz.ok <- krige(log(zinc) ~ 1, meuse, meuse.grid, v.fit,
+     block = c(40, 40))

[using ordinary kriging]
```

For a circular shape with radius 20, centred on the points of `meuse.grid`, one could select points on a regular grid within a circle:

```
> xy <- expand.grid(x = seq(-20, 20, 4), y = seq(-20, 20,
+     4))
> xy <- xy[(xy$x^2 + xy$y^2) <= 20^2, ]
> lz.ok <- krige(log(zinc) ~ 1, meuse, meuse.grid, v.fit,
+     block = xy)

[using ordinary kriging]
```

For block averages over varying regions, the `newdata` argument, usually a grid, can be replaced by a polygons object. Suppose `meuse.polygons` contains polygons for which we want to predict block averages, then this is done by

```
> lz.pols <- krige(log(zinc) ~ 1, meuse, meuse.polygons,
+     v.fit)
```

To discretise each (top level) Polygons object, coordinates that discretize the polygon are obtained by

```
> spsample(polygon, n = 500, type = "regular", offset = c(0.5,
+     0.5))
```

meaning that a regular discretisation is sought with approximately 500 points. These default arguments to `spsample` can be modified by altering the `sps.args` argument to `predict.gstat`; `spsample` is described on p. 146.

A much less efficient way to obtain block kriging predictions and prediction errors is to use Gaussian conditional simulation (Sect. 8.8) over a fine grid, calculate block means from each realisation, and obtain the mean and variance from a large number of realisations. In the limit, this should equal the analytical block kriging prediction.

When instead of a block *average* a non-linear spatial aggregation is required, such as a quantile value of points within a block, or the area fraction of a block where points exceed a threshold (Sect. 8.8.2), the simulation path is the more natural approach.

8.5.7 Stratifying the Domain

When a categorical variable is available that splits the area of interest in a number of disjoint areas, for example based on geology, soil type, land use or some other factor, we might want to apply separate krigings to the different units. This is called stratified kriging. The reason for doing kriging per-class may be that the covariance structure (semivariogram) is different for the different classes. In contrast to universal kriging with a categorical predictor, no correlation is assumed between residuals from different classes.

The example assumes there is a variable `part.a`, which partitions the area in two sub-areas, where `part.a` is 0 and where it is 1. First we can try to find out in which grid cells the observations lie:

```
> meuse$part.a <- gstat::idw(part.a ~ 1, meuse.grid,
+     meuse, nmax = 1)$var1.pred

[inverse distance weighted interpolation]
```

here, any interpolation may do, as we basically use the first nearest neighbour as predictor. A more robust approach may be to use the `over` method,

```
> meuse$part.a <- over(meuse, meuse.grid["part.a"])[[1]]
```

or equivalently,

```
> meuse$part.a <- meuse.grid$part.a[over(meuse, geometry(meuse.grid))]
```

because assign `NA` to point values not in a grid cell.

Next, we can perform kriging for each of the sub-domains, store them in `x1` and `x2`, and merge the result using `rbind` in their non-spatial representation:

```
> x1 <- krige(log(zinc) ~ 1, meuse[meuse$part.a == 0, ],
+     meuse.grid[meuse.grid$part.a == 0, ], model = vgm(0.548,
+         "Sph", 900, 0.0654), nmin = 20, nmax = 40, maxdist = 1000)

[using ordinary kriging]

> x2 <- krige(log(zinc) ~ 1, meuse[meuse$part.a == 1, ],
+     meuse.grid[meuse.grid$part.a == 1, ], model = vgm(0.716,
+         "Sph", 900), nmin = 20, nmax = 40, maxdist = 1000)

[using ordinary kriging]

> lz.stk <- rbind(as.data.frame(x1), as.data.frame(x2))
> coordinates(lz.stk) <- c("x", "y")
> lz.stk <- as(x, "SpatialPixelsDataFrame")

> spplot(lz.stk["var1.pred"], main = "stratified kriging predictions")
```

8.5.8 Trend Functions and Their Coefficients

For cases of exploratory data analysis or analysis of specific regression diagnostics, it may be of interest to limit prediction to the trend component $x(s_0)\hat{\beta}$, ignoring the prediction of the residual, that is ignoring the second term in the right-hand side of (8.6). This can be accomplished by setting argument `BLUE = TRUE` to `predict.gstat`:

```
> g.tr <- gstat(formula = log(zinc) ~ sqrt(dist), data = meuse,
+     model = v.fit)
> predict(g.tr, meuse[1, ])

[using universal kriging]
       coordinates var1.pred     var1.var
1 (181072, 333611)  6.929517 2.409685e-34

> predict(g.tr, meuse[1, ], BLUE = TRUE)

[generalized least squares trend estimation]
       coordinates var1.pred   var1.var
1 (181072, 333611)  6.862085 0.06123864
```

The first output yields the observed value (with zero variance), the second yields the generalised least squares trend component.

If we want to do significance testing of regression coefficients under a full model with spatially correlated residuals, we need to find out what the estimated regression coefficients and their standard errors are. For this, we can use **gstat** in a debug mode, in which case it will print a lot of information about intermediate calculations to the screen; just try

```
> predict(g, meuse[1, ], BLUE = TRUE, debug = 32)
```

but this does not allow saving the actual coefficients as data in R. Another way is to 'fool' the prediction mode with a specific contrast on the regression coefficients, for example the vector $x(s_0) = (0, 1)$, such that $x(s_0)\hat{\beta} = 0\hat{\beta}_0 + 1\hat{\beta}_1 = \hat{\beta}_1$. Both regression coefficient estimates are obtained by

```
> meuse$Int <- rep(1, 155)
> g.tr <- gstat(formula = log(zinc) ~ -1 + Int + sqrt(dist),
+     data = meuse, model = v.fit)
> rn <- c("Intercept", "beta1")
> df <- data.frame(Int = c(0, 1), dist = c(1, 0), row.names = rn)
> spdf <- SpatialPointsDataFrame(SpatialPoints(matrix(0,
+     2, 2)), df)
> spdf

          coordinates Int dist
Intercept      (0, 0)   0    1
beta1          (0, 0)   1    0

> predict(g.tr, spdf, BLUE = TRUE)
```

```
[generalized least squares trend estimation]
  coordinates var1.pred   var1.var
1      (0, 0) -2.471753 0.20018883
2      (0, 0)  6.953173 0.06633691
```

The `Int` variable is a 'manual' intercept to replace the automatic intercept, and the `-1` in the formula removes the automatic intercept. This way, we can control it and give it the zero value. The predictions now contain the generalised least squares estimates of the regression model.

8.5.9 Non-linear Transforms of the Response Variable

For predictor variables, a non-linear transform simply yields a new variable and one can proceed as if nothing has happened. Searching for a good transform, such as using `sqrt(dist)` instead of direct `dist` values, may help in approaching the relationship with a straight line. For dependent variables this is not the case: because statistical expectation ('averaging') is a linear operation, $E(g(X)) = g(E(X))$ only holds if $g(\cdot)$ is a linear operator. This means that if we compute kriging predictors for zinc on the log scale, we do not obtain the expected zinc concentration by taking the exponent of the kriging predictor.

A large class of monotonous transformations is provided by the Box–Cox family of transformations, which take a value λ:

$$f(y, \lambda) = \begin{cases} (y^\lambda - 1)/\lambda & \text{if } \lambda \neq 0, \\ \ln(y) & \text{if } \lambda = 0. \end{cases}$$

A likelihood profile plot for lambda is obtained by the `boxcox` method in the package bundle **MASS**. For example, the plot resulting from

```
> library(MASS)
> boxcox(zinc ~ sqrt(dist), data = as.data.frame(meuse))
```

suggests that a Box–Cox transform with a slightly negative value for λ, for example $\lambda = -0.2$, might be slightly better in approaching a log-normal distribution.

Yet another transformation is the normal score transform (Goovaerts, 1997) computed by the function `qqnorm`, defined as

```
> meuse$zinc.ns <- qqnorm(meuse$zinc, plot.it = FALSE)$x
```

Indeed, the resulting variable has mean zero, variance close to 1 (exactly one if n is large), and plots a straight line on a normal probability plot. So simple as this transform is, so complex can the back-transform be: it requires linear interpolation between the sample points, and for the extrapolation of values outside the data range the cited reference proposes several different models for tail distributions, all with different coefficients. There seems to be little

guidance as how to choose between them based on sample data. It should be noted that back-transforming normal-score transformed values outside the data range is less of a problem with interpolation, but more so for simulation (Sect. 8.8).

Indicator kriging is obtained by indicator transforming a continuous variable, or reducing a categorical variable to a binary variable. An example for the indicator whether zinc is below 500 ppm is

```
> ind.f <- I(zinc < 500) ~ 1
> ind.fit <- fit.variogram(variogram(ind.f, meuse), vgm(1,
+     "Sph", 800, 1))
> ind.kr <- krige(ind.f, meuse, meuse.grid, ind.fit)

[using ordinary kriging]

> summary(ind.kr$var1.pred)

    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-0.03472  0.47490  0.80730  0.70390  0.94540  1.08800
```

Clearly, this demonstrates the difficulty of interpreting the resulting estimates of ones and zeros as probabilities, as one has to deal with negative values and values larger than one.

When it comes to non-linear transformations such as the log transform, the question whether to transform or not to transform is often a hard one. On the one hand, it introduces the difficulties mentioned; on the other hand, transformation solves problems like negative predictions for non-negative variables, and, for example heteroscedasticity: for non-negative variables the variability is larger for areas with larger values, which opposes the stationarity assumption where variability is independent from location.

When a continuous transform is taken, such as the log-transform or the Box–Cox transform, it is possible to back-transform quantiles using the inverse transform. So, under log-normal assumptions the exponent of the kriging mean on the log scale is an estimate of the median on the working scale. From back-transforming a large number of quantiles, the mean value and variance may be worked out.

8.5.10 Singular Matrix Errors

Kriging cannot deal with duplicate observations, or observations that share the same location, because they are perfectly correlated, and lead to singular covariance matrices V, meaning that $V^{-1}v$ has no unique solution. Obtaining errors due to a singular matrix is a common case.

```
> meuse.dup <- rbind(as.data.frame(meuse)[1, ], as.data.frame(meuse))
> coordinates(meuse.dup) = ~x + y
> krige(log(zinc) ~ 1, meuse.dup, meuse[1, ], v.fit)
```

will result in the output:

```
[using ordinary kriging]

"chfactor.c", line 130: singular matrix in function LDLfactor()
Error in predict.gstat(g, newdata=newdata, block=block, nsim=nsim:
        LDLfactor
```

which points to the `C` function where the actual error occurred (`LDLfactor`). The most common case where this happens is when duplicate observations are present in the data set. Duplicate observations can be found by

```
> zd <- zerodist(meuse.dup)
> zd

     [,1] [,2]
[1,]    1    2

> meuse0 <- meuse.dup[-zd[, 1], ]
> krige(log(zinc) ~ 1, meuse0, meuse[1, ], v.fit)

[using ordinary kriging]
       coordinates var1.pred var1.var
1 (181072, 333611)  6.929517        0
```

which tells that observations 1 and 2 have identical location; the third command removes the first of the pair. Near-duplicate observations are found by increasing the `zero` argument of function `zerodist` to a very small threshold.

Other common causes for singular matrices are the following:

- The use of variogram models that cause observations to have nearly perfect correlation, despite the fact that they do not share the same location, for example from `vgm(0, "Nug", 0)` or `vgm(1, "Gau", 1e20)`. The Gaussian model is *always* a suspect if errors occur when it is used; adding a small nugget often helps.
- Using a regression model with perfectly correlated variables; note that, for example a global regression model may lead to singularity in a local neighbourhood where a predictor may be constant and correlate perfectly with the intercept, or otherwise perfect correlation occurs.

Stopping execution on singular matrices is usually best: the cause needs to be found and somehow resolved. An alternative is to skip those locations and continue. For instance,

```
> setL <- list(cn_max = 1e+10)
> krige(log(zinc) ~ 1, meuse.dup, meuse[1, ], v.fit, set = setL)

[using ordinary kriging]
Warning:
Covariance matrix (nearly) singular at location [181072,333611,0]:
 skipping...
       coordinates var1.pred var1.var
1 (181072, 333611)        NA       NA
```

checks whether the estimated condition number for V and $X'V^{-1}X$ exceeds 10^{10}, in which case `NA` values are generated for prediction. Larger condition numbers indicate that a matrix is closer to singular. This is by no means a solution. It will also report whether V or $X'V^{-1}X$ *are* singular; in the latter case the cause is more likely related to collinear regressors, which reside in X.

Near-singularity may not be picked up, and can potentially lead to dramatically bad results: predictions that are orders of magnitude different from the observation data. The causes should be sought in the same direction as real singularity. Setting the `cn_max` value may help finding where this occurs.

8.6 Kriging, Filtering, Smoothing

Kriging results in spatially smooth interpolators that are, in case of a nonzero nugget effect, discontinuous in the measurement points. At measurement points, kriging prediction yields the measured value with a zero prediction error variance. In the following code, we fit a variogram model, compute the kriging prediction at the first observation location

```
> v <- variogram(log(zinc) ~ 1, meuse)
> v.fit <- fit.variogram(v, vgm(1, "Sph", 800, 1))
> v.fit

  model      psill    range
1   Nug 0.05065923   0.0000
2   Sph 0.59060463 896.9976

> log(meuse$zinc[1])

[1] 6.929517

> krige(log(zinc) ~ 1, meuse, meuse[1, ], v.fit)

[using ordinary kriging]
       coordinates var1.pred var1.var
1 (181072, 333611)  6.929517        0
```

If we would shift this first point spatially with any small amount, say 1 m, we would see a strongly different prediction:

```
> krige(log(zinc) ~ 1, meuse, meuse_shift[1, ], v.fit)

[using ordinary kriging]
       coordinates var1.pred var1.var
1 (181073, 333612)  6.880461 0.089548
```

As a matter of fact, kriging in presence of a nugget effect, when considered as prediction in a small region around a prediction location, is discontinuous and hence not smooth. This effect goes unnoticed when data locations do not coincide with the (gridded) prediction locations.

An alternative approach could conceive the measured process $Z(s)$ as $Z(s) = U(s) + \epsilon(s)$, the sum of an underlying, smooth process $U(s)$ and

a (rough) measurement error $\epsilon(s)$, and one could focus on predicting $U(s)$ rather than $Z(s)$. The motivation for this is that the reproduction (prediction) of measurement errors, even when known, is not of interest. Spatial prediction, or *filtering*, now follows (8.6) and (8.7) but with v being the covariance vector of $Z(s)$ and $U(s_0)$, which does *not* involve the discontinuity of the nugget effect.

Predicting the $U(s)$ process is obtained by rephrasing the nugget effect as an error:

```
> err.fit <- fit.variogram(v, vgm(1, "Sph", 800, Err = 1))
> err.fit

  model      psill    range
1   Err 0.05065923   0.0000
2   Sph 0.59060463 896.9976

> krige(log(zinc) ~ 1, meuse, meuse[1, ], err.fit)

[using ordinary kriging]
       coordinates var1.pred   var1.var
1 (181072, 333611)  6.884405 0.03648707
```

where one can see that the prediction is no longer equal to the data value but very similar to the prediction at the minimally shifted prediction location. The prediction variance is very similar to the kriging variance at the minimally shifted prediction location minus the nugget variance (the variance of $\epsilon(s)$).

One could also find some compromise between kriging and filtering, or nugget and measurement error. When it is known that the measurement error variance is 0.01, a nugget could still be fitted:

```
> v = fit.variogram(v, vgm(1, "Sph", 800, Err = 0.01, nugget = 1),
+     fit.sill = c(FALSE, TRUE, TRUE))
> v

  model      psill    range
1   Err 0.01000000   0.0000
2   Nug 0.04065923   0.0000
3   Sph 0.59060463 896.9976

> krige(log(zinc) ~ 1, meuse[1, ], meuse[1, ], v)

[using ordinary kriging]
       coordinates var1.pred var1.var
1 (181072, 333611)  6.929517     0.01
```

Usually, nugget and measurement error cannot be fitted separately from typical datasets.

8.7 Model Diagnostics

The model diagnostics we have seen so far are fitted and residual plots (for linear regression models), spatial identification of groups of points in the variogram cloud, visual and numerical inspection of variogram models, and visual and numerical inspection of kriging results. Along the way, we have seen many model decisions that needed to be made; the major ones being the following:

- Possible transformation of the dependent variable
- The form of the trend function
- The cutoff, lag width, and possibly directional dependence for the sample variogram
- The variogram model type
- The variogram model coefficient values, or fitting method
- The size and criterion to define a local neighbourhood

and we have seen fairly little *statistical* guidance as to which choices are better. To some extent we can 'ask' the data what a good decision is, and for this we may use cross validation. We see that there are some model choices that do not seem very important, and others that cross validation simply cannot inform us about.

8.7.1 Cross Validation Residuals

Cross validation splits the data set into two sets: a modelling set and a validation set. The modelling set is used for variogram modelling and kriging on the locations of the validation set, and then the validation measurements can be compared to their predictions. If all went well, cross validation residuals are small, have zero mean, and no apparent structure.

How should we choose or isolate a set for validation? A possibility is to randomly partition the data in a model and test set. Let us try this for the `meuse` data set, splitting it in 100 observations for modelling and 55 for testing:

```
> sel100 <- sample(1:155, 100)
> m.model <- meuse[sel100, ]
> m.valid <- meuse[-sel100, ]
> v100.fit <- fit.variogram(variogram(log(zinc) ~ 1, m.model),
+     vgm(1, "Sph", 800, 1))
> m.valid.pr <- krige(log(zinc) ~ 1, m.model, m.valid,
+     v100.fit)

[using ordinary kriging]

> resid.kr <- log(m.valid$zinc) - m.valid.pr$var1.pred
> summary(resid.kr)
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-0.79990 -0.18240 -0.03922 -0.01881  0.19210  1.06900
```

```
> resid.mean <- log(m.valid$zinc) - mean(log(m.valid$zinc))
> R2 <- 1 - sum(resid.kr^2)/sum(resid.mean^2)
> R2
```

```
[1] 0.717017
```

which indicates that kriging prediction is a better predictor than the mean, with an indicative R^2 of 0.72. Running this analysis again will result in different values, as another random sample is chosen. Also note that no visual verification that the variogram model fit is sensible has been applied. A map with cross validation residuals can be obtained by

```
> m.valid.pr$res <- resid.kr
```

```
> bubble(m.valid.pr, "res")
```

A similar map is shown for 155 residuals in Fig. 8.12. Here, symbol size denotes residual size, with symbol area proportional to absolute value.

To use the data to a fuller extent, we would like to use all observations to create a residual once; this may be used to find influential observations. It can be done by replacing the first few lines in the example above with

```
> nfold <- 3
> part <- sample(1:nfold, 155, replace = TRUE)
> sel <- (part != 1)
> m.model <- meuse[sel, ]
> m.valid <- meuse[-sel, ]
```

and next define `sel = (part != 2)`, etc. Again, the random splitting brings in a random component to the outcomes. This procedure is threefold cross validation, and it can be easily extended to n-fold cross validation. When n equals the number of observations, the procedure is called leave-one-out cross validation.

A more automated way to do this is provided by the **gstat** functions `krige.cv` for univariate cross validation and `gstat.cv` for multivariable cross validation:

```
> v.fit <- vgm(0.59, "Sph", 874, 0.04)
> cv155 <- krige.cv(log(zinc) ~ 1, meuse, v.fit, nfold = 5,
+     verbose = FALSE)
```

```
> bubble(cv155, "residual", main = "log(zinc): 5-fold CV residuals")
```

the result of which is shown in Fig. 8.12. It should be noted that these functions do not re-fit variograms for each fold; usually a variogram is fit on the complete data set, and in that case validation residuals are not completely independent from modelling data, as they already did contribute to the variogram model fitting.

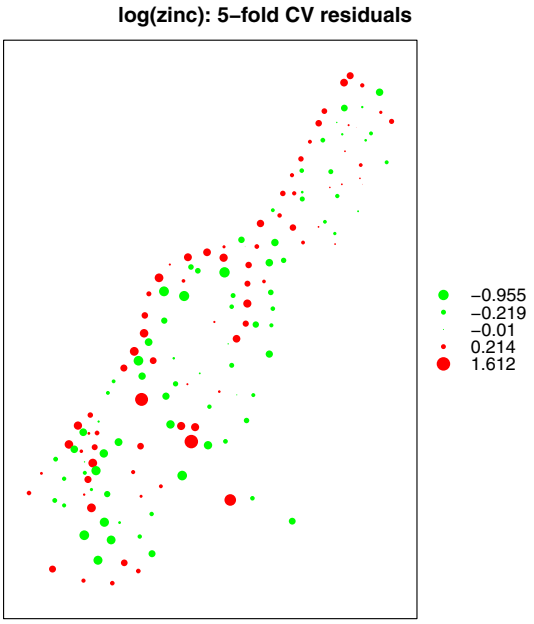

Fig. 8.12 Cross validation residuals for fivefold cross validation; symbol size denotes residual magnitude, positive residuals indicate under-prediction

8.7.2 Cross Validation z-Scores

The `krige.cv` object returns more than residuals alone:

```
> summary(cv155)

Object of class SpatialPointsDataFrame
Coordinates:
     min    max
x 178605 181390
y 329714 333611
Is projected: NA
proj4string : [NA]
Number of points: 155
Data attributes:
   var1.pred        var1.var         observed        residual
 Min.   :4.808   Min.   :0.1102   Min.   :4.727   Min.   :-0.955422
 1st Qu.:5.380   1st Qu.:0.1519   1st Qu.:5.288   1st Qu.:-0.218794
 Median :5.881   Median :0.1779   Median :5.787   Median :-0.010007
 Mean   :5.887   Mean   :0.1914   Mean   :5.886   Mean   :-0.001047
 3rd Qu.:6.333   3rd Qu.:0.2145   3rd Qu.:6.514   3rd Qu.: 0.213743
 Max.   :7.257   Max.   :0.5370   Max.   :7.517   Max.   : 1.611695
     zscore               fold
 Min.   :-2.270139   Min.   :1.000
 1st Qu.:-0.508470   1st Qu.:2.000
 Median :-0.023781   Median :3.000
```

```
 Mean   : 0.001421   Mean   :2.987
 3rd Qu.: 0.498811   3rd Qu.:4.000
 Max.   : 3.537729   Max.   :5.000
```

the variable `fold` shows to which fold each record belonged, and the variable `zscore` is the z-score, computed as

$$z_i = \frac{Z(s_i) - \hat{Z}_{[i]}(s_i)}{\sigma_{[i]}(s_i)},$$

with $\hat{Z}_{[i]}(s_i)$ the cross validation prediction for s_i, and $\sigma_{[i]}(s_i)$ the corresponding kriging standard error. In contrast to standard residuals the z-score takes the kriging variance into account: it is a standardised residual, and if the variogram model is correct, the z-score should have mean and variance values close to 0 and 1. If, in addition, $Z(s)$ follows a normal distribution, so should the z-score do.

8.7.3 Multivariable Cross Validation

Multivariable cross validation is obtained using the `gstat.cv` function:

```
> g.cv <- gstat.cv(g, nmax = 40)
```

Here, the neighbourhood size is set to the nearest 40 observations for computational reasons. With multivariable cross validation, two additional parameters need be considered:

- `remove.all = FALSE` By default only the first variable is cross-validated, and all other variables are used to their full extent for prediction on the validation locations; if set to `TRUE`, also secondary data at the validation locations are removed.
- `all.residuals = FALSE` By default only residuals are computed and returned for the primary variable; if set to `TRUE`, residuals are computed and returned for all variables.

In a truly multivariable setting, where there is no hierarchy between the different variables to be considered, both should be set to `TRUE`.

8.7.4 Limitations to Cross Validation

Cross validation can be useful to find artefacts in data, but it should be used with caution for confirmatory purposes: one needs to be careful not to conclude that our (variogram and regression) model is correct if cross

validation does not lead to unexpected findings. It is for instance not good at finding what is not in the data.

As an example, the Meuse data set does not contain point pairs with a separation distance closer than 40. Therefore, the two variogram models

```
> v1.fit <- vgm(0.591, "Sph", 897, 0.0507)
> v2.fit <- vgm(0.591, "Sph", 897, add.to = vgm(0.0507,
+     "Sph", 40))
```

that only differ with respect to the spatial correlation at distances smaller than 40 m yield identical cross validation results:

```
> set.seed(13331)
> cv155.1 <- krige.cv(log(zinc) ~ 1, meuse, v1.fit, nfold = 5,
+     verbose = FALSE)
> set.seed(13331)
> cv155.2 <- krige.cv(log(zinc) ~ 1, meuse, v2.fit, nfold = 5,
+     verbose = FALSE)
> summary(cv155.1$residual - cv155.2$residual)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      0       0       0       0       0       0
```

Note that the `set.seed(13331)` was used here to force identical assignments of the otherwise random folding. When used for block kriging, the behaviour of the variogram at the origin is of utmost importance, and the two models yield strongly differing results. As an example, consider block kriging predictions at the `meuse.grid` cells:

```
> b1 <- krige(log(zinc) ~ 1, meuse, meuse.grid, v1.fit,
+     block = c(40, 40))$var1.var

[using ordinary kriging]

> b2 <- krige(log(zinc) ~ 1, meuse, meuse.grid, v2.fit,
+     block = c(40, 40))$var1.var

[using ordinary kriging]

> summary((b1 - b2)/b1)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-0.4313 -0.2195 -0.1684 -0.1584 -0.1071  0.4374
```

where some kriging variances drop, but most increase, up to 30 % when using the variogram without nugget instead of the one with a nugget. The decision which variogram to choose for distances shorter than those available in the data is up to the analyst, and matters.

8.8 Geostatistical Simulation

Geostatistical simulation refers to the simulation of possible realisations of a random field, given the specifications for that random field (e.g. mean structure, residual variogram, intrinsic stationarity) and possibly observation data. Conditional simulation produces realisations that exactly honour observed data at data locations, unconditional simulations ignore observations and only reproduce means and prescribed variability.

Geostatistical simulation is fun to do, to see how much (or little) realisations can vary given the model description and data, but it is a poor means of quantitatively communicating uncertainty: many realisations are needed and there is no obvious ordering in which they can be viewed. They are, however, often needed when the uncertainty of kriging predictions is, for example input to a next level of analysis, and spatial correlation plays a role. An example could be the use of rainfall fields as input to spatially distributed rainfall-runoff models: interpolated values and their variances are of little value, but running the rainfall-runoff model with a large number of simulated rainfall fields may give a realistic assertion of the uncertainty in runoff, resulting from uncertainty in the rainfall field.

Calculating runoff given rainfall and catchment characteristic can be seen as a non-linear spatial aggregation process. Simpler non-linear aggregations are, for example for a given area or block the fraction of the variable that exceeds a critical limit, the 90th percentile within that area, or the actual area where (or its size distribution for which) a measured concentration exceeds a threshold. Simulation can give answers in terms of predictions as well as predictive distributions for all these cases. Of course, the outcomes can never be better than the degree to which the simulation model reflects reality.

The next subsection introduces sequential simulation, a very generic and flexible method. Package **RandomFields** provides a large number of other simulation algorithms; packages **RandomFields** and **fields** both contain implementations of the circulant embedding method (Dietrich and Newsam, 1993), which is often preferred for its computational speed.

8.8.1 Sequential Simulation

For simulating random fields, package **gstat** only provides the sequential simulation algorithm (see, e.g. Goovaerts, 1997 for an explanation), and provides this for Gaussian simulation and indicator simulation, possibly multivariable, optionally with simulation of trend components, and optionally for block mean values.

Sequential simulation proceeds as follows; following a random path through the simulation locations, it repeats the following steps:

1. Compute the conditional distribution given data and previously simulated values, using simple kriging
2. Draw a value from this conditional distribution
3. Add this value to the data set
4. Go to the next unvisited location, and go back to 1

until all locations have been visited. In step 2, either the Gaussian distribution is used or the indicator kriging estimates are used to approximate a conditional distribution, interpreting kriging estimates as probabilities (after some fudging!).

Step 1 of this algorithm will become the most computationally expensive when all data (observed and previously simulated) are used. Also, as the number of simulation nodes is usually much larger than the number of observations, the simulation process will slow down more and more when global neighbourhoods are used. To obtain simulations with a reasonable speed, we need to set a maximum to the neighbourhood. This is best done with the `nmax` argument, as spatial data density increases when more and more simulated values are added. For simulation we again use the functions `krige` or `predict.gstat`; the argument `nsim` indicates how many realisations are requested:

```
> lzn.sim <- krige(log(zinc) ~ 1, meuse, meuse.grid, v.fit,
+     nsim = 6, nmax = 40)

drawing 6 GLS realisations of beta...
[using conditional Gaussian simulation]

> spplot(lzn.sim)
```

the result of which is shown in Fig. 8.13. It should be noted that these realisations are created following a single random path, in which case the expensive results ($V^{-1}v$ and the neighbourhood selection) can be re-used. Alternatively, one could use six function calls, each with `nsim = 1`.

The simulation procedure above also gave the output line `drawing 6 GLS realisations of beta...`, which confirms that prior to simulation of the field *for each realisation* a trend vector (in this case a mean value only) is drawn from the normal distribution with mean $(X'V^{-1}X)^{-1}X'V^{-1}Z(s)$ and variance $(X'V^{-1}X)^{-1}$, that is the generalised least squares estimate and estimation variance. This procedure leads to simulations that have mean and variance equal to the ordinary or universal kriging mean and variance, and that have residual spatial correlation according to the variogram prescribed (Abrahamsen and Benth, 2001). For simulations that omit the simulation of the trend coefficients, the vector β should be passed, for example as `beta = 5.9` to the `krige` function, as with the simple kriging example. In that case, the simulated fields will follow the simple kriging mean and variance.

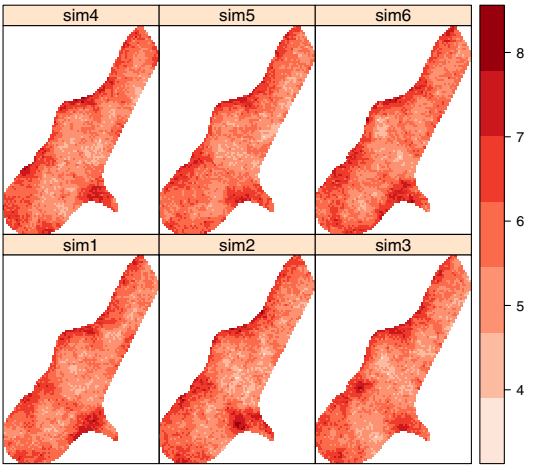

Fig. 8.13 Six realisations of conditional Gaussian simulation for log-zinc

8.8.2 Non-linear Spatial Aggregation and Block Averages

Suppose the area shown in Fig. 8.14 is the target area for which we want to know the fraction above a threshold; the area being available as a `SpatialPolygons` object `area`. We can now compute the distribution of the fraction of the area above a cutoff of 500 ppm by simulation:

```
> nsim <- 1000
> cutoff <- 500
> sel.grid <- meuse.grid[area.sp, ]
> lzn.sim <- krige(log(zinc) ~ 1, meuse, sel.grid, v.fit,
+     nsim = nsim, nmax = 40)

drawing 1000 GLS realisations of beta...
[using conditional Gaussian simulation]

> res <- apply(as.data.frame(lzn.sim)[1:nsim], 2, function(x) mean(x >
+     log(cutoff)))

> hist(res, main = paste("fraction above", cutoff), xlab = NULL,
+     ylab = NULL)
```

shown in the right-hand side of Fig. 8.14. Note that if we had been interested in the probability of `mean(x) > log(cutoff)`, which is a rather different issue, then block kriging would have been sufficient:

```
> bkr <- krige(log(zinc) ~ 1, meuse, area.sp, v.fit)

[using ordinary kriging]

> 1 - pnorm(log(cutoff), bkr$var1.pred, sqrt(bkr$var1.var))

[1] 0.9998791
```

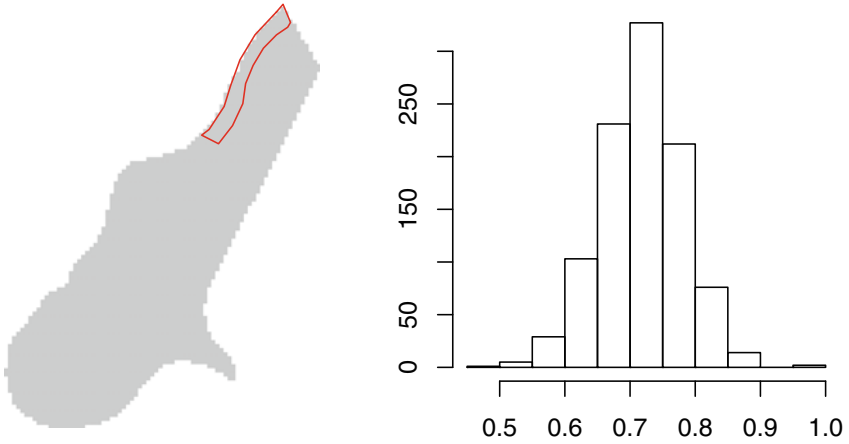

Fig. 8.14 A non-rectangular area for which a non-linear aggregation is required (*left*: *red*), and distribution of the fraction with zinc concentration above 500 ppm for this area

Block averages can be simulated directly by supplying the block argument to `krige`; simulating points and aggregating these to block means *may* be more efficient because simulating blocks calls for the calculation of many block–block covariances, which involves the computation of quadruple integrals.

8.8.3 Multivariable and Indicator Simulation

Multivariable simulation is as easy as cokriging, try

```
> cok.sims <- predict(vm.fit, meuse.grid, nsim = 1000)
```

after passing the `nmax = 40`, or something similar to the `gstat` calls used to build up `vm.fit` (Sect. 8.4.5).

Simulation of indicators is done along the same lines. Suppose we want to simulate soil class 1, available in the Meuse data set:

```
> table(meuse$soil)

 1  2  3
97 46 12

> s1.fit <- fit.variogram(variogram(I(soil == 1) ~ 1, meuse),
+     vgm(1, "Sph", 800, 1))
> s1.sim <- krige(I(soil == 1) ~ 1, meuse, meuse.grid,
+     s1.fit, nsim = 6, indicators = TRUE, nmax = 40)

drawing 6 GLS realisations of beta...
[using conditional indicator simulation]

> spplot(s1.sim)
```

which is shown in Fig. 8.15.

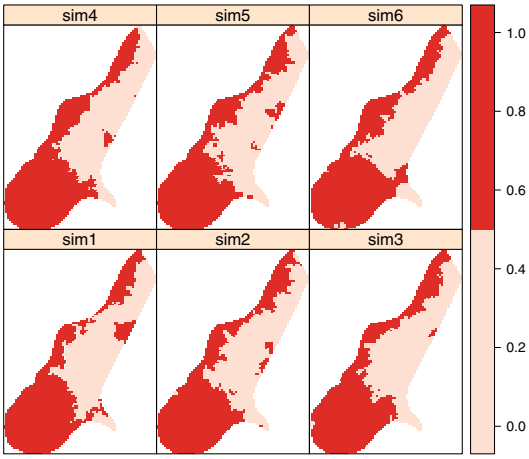

Fig. 8.15 Six realisations of conditional indicator simulation for soil type 1

8.9 Model-Based Geostatistics and Bayesian Approaches

Up to now, we have only seen kriging approaches where it was assumed that the variogram model, fitted from sample data, is assumed to be *known* when we do the kriging or simulation: any uncertainty about it is ignored. Diggle et al. (1998) give an approach, based on linear mixed and generalized linear mixed models, to provide what they call *model-based* geostatistical predictions. It incorporates the estimation error of the variogram coefficients.

When is uncertainty of the variogram an important issue? Obviously, when the sample is small, or, for example when variogram modelling is problematic due to the presence of extreme observations or data that come from a strongly skewed distribution.

8.10 Monitoring Network Optimisation

NA Monitoring costs money. Monitoring programs have to be designed, started, stopped, evaluated, and sometimes need to be enlarged or shrunken. The difficulty of finding optimal network designs (Mateu and Müller, 2013) is that a quantitative criterion is often a priori not present. For example, should one focus on mean kriging variances, or variance of some global mean estimator, or rather on the ability to delineate a particular contour?

A very simple approach towards monitoring network optimisation is to find the point whose removal leads to the smallest increase in mean kriging variance:

```
> m1 <- sapply(1:155, function(x) mean(krige(log(zinc) ~
+     1, meuse[-x, ], meuse.grid, v.fit)$var1.var))
> which(m1 == min(m1))
```

which will point to observation 72 as the first candidate for removal. Looking at the sorted magnitudes of change in mean kriging variance, by

```
> plot(sort(m1))
```

will reveal that for several other candidate points their removal will have an almost identical effect on the mean variance.

Another approach could be, for example to delineate say the 500 ppm contour. We could, for example express the doubt about whether a location is below or above 500 as the closeness of $G((\hat{Z}(s_0) - 500)/\sigma(s_0))$ to 0.5, with $G(\cdot)$ the Gaussian distribution function.

```
> cutoff <- 1000
> f <- function(x) {
+     kr = krige(log(zinc) ~ 1, meuse[-x, ], meuse.grid,
+         v.fit)
+     mean(abs(pnorm((kr$var1.pred - log(cutoff))/sqrt(kr$var1.var)) -
+         0.5))
+ }
> m2 <- sapply(1:155, f)
> which(m2 == max(m2))
```

Figure 8.16 shows that different objectives lead to different candidate points. Also, deciding based on the kriging variance alone results in an outcome that is highly predictable from the points configuration alone: points in the densest areas are candidate for removal.

For *adding* observation points, one could loop over a fixed grid and find the point that increases the objective most; this is more work as the number of options for addition are much larger than that for removal. Evaluating on the kriging variance is not a problem, as the observation value does not affect the kriging variance. For the second criterion, it does.

The problem when two or more points have to be added or removed *jointly* becomes computationally very intensive, as the sequential solution (first the best point, then the second best) is not necessarily equal to the joint solution, for example which configuration of n points is best. Instead of an exhaustive search a more clever optimisation strategy such as simulated annealing or a genetic algorithm should be used.

Package **fields** has a function `cover.design` that finds a set of points on a finite grid that minimises a geometric space-filling criterion

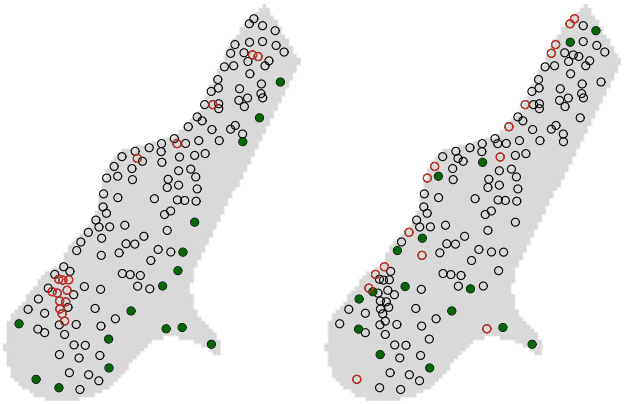

Fig. 8.16 Candidate points for removal. *Left*: for mean kriging variance, *right*: for delineating the 1,000 ppm contour. *Open*, *red circles*: 10 % most favourite points; *closed*, *green circles*: 10 % least favourite points

8.11 Other R Packages for Interpolation and Geostatistics

8.11.1 Non-geostatistical Interpolation

Besides inverse distance weighted interpolation and kriging, other interpolation methods may be used, for example based on generalised additive models or smoothers, such as given in package **mgcv**. Additive models in coordinates without interaction will not yield rotation-invariant solutions, but two-dimensional smoothing splines will. The interested reader is referred to Wood (2006).

Package **fields** also provides a function `Tps`, for thin plate (smoothing) splines. Package **akima** provides interpolation methods based on bilinear or bicubic splines (Akima, 1978); the package has a restricive license, and work on an unencumbered replacement is planned.

Package **stinepack** provides a 'consistently well behaved method of interpolation based on piecewise rational functions using Stineman's algorithm' (Stineman, 1980).

An interpolation method that also has this property but that does take observation configuration into account is *natural neighbour* interpolation (Sibson, 1981), but this is not available in R.

None of the packages mentioned in this sub-section accept or return data in one of the `Spatial` classes of package **sp**.

8.11.2 Spatial

Package **spatial** is part of the **VR** bundle that comes with Venables and Ripley (2002) and is probably one of the oldest spatial packages available in R. It provides calculation of spatial correlation using functions `correlogram` or `variogram`. It allows ordinary and universal point kriging over a grid for spherical, exponential, and Gaussian covariance models. For universal kriging predictors it only allows polynomials in the spatial coordinates. Function `surf.gls` fits a trend surface (i.e. a polynomial in the coordinates) by generalised least squares, and it has a `predict` method.

8.11.3 RandomFields

RandomFields offers sample variogram computation, variogram fitting by least squares or maximum likelihood or restricted maximum likelihood. Simple and ordinary point kriging are provided, and unconditional and conditional simulation using a large variety of modern simulation methods different from sequential simulation, many of them based on spectral methods and Fourier transforms; their abbreviated code is shown as column labels in the table obtained by

```
> PrintModelList()
```

an explanation of its output is available in the help for `PrintMethodList`. The package provides a large set of covariance functions, shown as row labels in the model list.

8.11.4 geoR and geoRglm

In addition to variogram estimation, variogram model function fitting using least squares or (restricted) maximum likelihood (`likfit`), and ordinary and universal point kriging, package **geoR** allows for Bayesian kriging (function `krige.bayes` of (transformed) Gaussian variables). This requires the user to specify priors for each of the variogram model parameters (but not for the trend coefficients); `krige.bayes` will then compute the posterior kriging distribution. The function documentation points to a long list of documents that describe the implementation details. The book by Diggle and Ribeiro Jr. (2007) describes more details and gives worked examples.

Package **geoR** uses its own class for spatial data, called `geodata`. It contains coercion method for point data in **sp** format, try, for example

```
> library(geoR)
> plot(variog(as.geodata(meuse["zinc"]), max.dist = 1500))
```

Package **geoR** also has an `xvalid` function for leave-one-out cross validation that (optionally) allows for re-estimating model parameters (trend and variogram coefficients) when leaving out each observation. It also provides the `eyefit`, for interactive visual fitting of functions to sample variograms (see Sect. 8.4.3).

Package **geoRglm** extends package **geoR** for binomial and Poisson processes, and includes Bayesian and conventional kriging methods for trans-Gaussian processes. It mostly uses MCMC approaches, and may be slow for larger data sets.

8.11.5 Fields

Package **fields** is an R package for curve and function fitting with an emphasis on spatial data. The main spatial prediction methods are thin plate splines (function `Tps`) and kriging (function `Krig` and `krig.image`). The kriging functions allow you to supply a covariance function that is written in native code. Functions that are positive definite on a sphere (i.e. for unprojected data) are available. Function `cover.design` is written for monitoring network optimisation.

8.11.6 spBayes

Package **spBayes** fits univariate and multivariate models with Markov chain Monte Carlo (MCMC). Core functions include univariate and multivariate Gaussian regression with spatial random effects (functions `spLM`, `spMvLM`) and univariate and multivariate logistic and Poisson regression with spatial random effects (functions `spGLM` and `spMvGLM`).

8.12 Spatio-Temporal Prediction

Spatio-temporal prediction methods are provided by several R packages. Package **gstat** contains a vignette that explains how to compute and fit spatio-temporal variogram models, and use them in spatio-temporal kriging. It uses the classes of **spacetime** for this. Variogram models include the metric, product-sum, sum-metric, and separable model. The vignette is accessed by

```
> vignette("st", package = "gstat")
```

Other packages providing particular space-time variogram model fitting and prediction options include **RandomFields**, **SpatioTemporal**, and **spTimer**. The SpatioTemporal task view on CRAN gives an overview of packages for analyzing spatio-temporal data, which includes spatio-temporal geostatistics. It is found at http://cran.r-project.org/view=SpatioTemporal.