Chapter 15

Spatial Statistics

Spatial statistics is a recent and graphical subject that is ideally suited to implementation in S; S-PLUS itself includes one spatial interpolation method, `akima`, and `loess` which can be used for two-dimensional smoothing, but the specialist methods of spatial statistics have been added and are given in our library section `spatial`. The main references for spatial statistics are Ripley (1981, 1988), Diggle (1983), Upton and Fingleton (1985) and Cressie (1991). Not surprisingly, our notation is closest to that of Ripley (1981).

The S-PLUS module[1] S+SPATIALSTATS (Kaluzny and Vega, 1997) provides more comprehensive (and more polished) facilities for spatial statistics than those provided in our library section `spatial`. Details of how to work through our examples in that module may be found in the on-line complements[2] to this book.

More recently other contributed software has become available. There are geostatistical packages called `geoR`/`geoS`[3] and `sgeostat`,[4] and point-process packages `splancs`[5] and `spatstat`.[6]

15.1 Spatial Interpolation and Smoothing

We provide three examples of datasets for spatial interpolation. The dataset `topo` contains 52 measurements of topographic height (in feet) within a square of side 310 feet (labelled in 50 feet units). The dataset `npr1` contains permeability measurements (a measure of the ease of oil flow in the rock) and porosity (the volumetric proportion of the rock which is pore space) measurements from an oil reserve in the USA.

Suppose we are given n observations $Z(x_i)$ and we wish to map the process $Z(x)$ within a region D. (The sample points x_i are usually, but not always, within D.) Although our treatment is quite general, our S code assumes D to be a two-dimensional region, which covers the majority of examples. There are

[1] S-PLUS modules are additional-cost products; contact your S-PLUS distributor for details.
[2] See page 461 for where to obtain these.
[3] `http://www.maths.lancs.ac.uk/~ribeiro/geoR.html` and on CRAN.
[4] `http://www.gis.iastate.edu/SGeoStat/homepage.html`; R port on CRAN
[5] `http://www.maths.lancs.ac.uk/~rowlings/Splancs/`; R port on CRAN.
[6] `http://www.maths.uwa.edu.au/~adrian/spatstat.html` and on CRAN.

however applications to the terrestrial sphere and in three dimensions in mineral and oil applications.

Trend surfaces

One of the earliest methods was fitting *trend surfaces*, polynomial regression surfaces of the form

$$f((x,y)) = \sum_{r+s \leqslant p} a_{rs} x^r y^s \tag{15.1}$$

where the parameter p is the order of the surface. There are $P = (p+1)(p+2)/2$ coefficients. Originally (15.1) was fitted by least squares, and could for example be fitted using `lm` with `poly` which will give polynomials in one or more variables. However, there will be difficulties in prediction, and this is rather inefficient in applications such as ours in which the number of points at which prediction is needed may far exceed n. Our function `surf.ls` implicitly rescales x and y to $[-1, 1]$, which ensures that the first few polynomials are far from collinear. We show some low-order trend surfaces for the `topo` dataset in Figure 15.1, generated by:

```
library(spatial)
par(mfrow = c(2, 2), pty = "s")
topo.ls <- surf.ls(2, topo)
trsurf <- trmat(topo.ls, 0, 6.5, 0, 6.5, 30)
eqscplot(trsurf, , xlab = "", ylab = "", type = "n")
contour(trsurf, levels = seq(600, 1000, 25), add = T)
points(topo)
title("Degree = 2")
topo.ls <- surf.ls(3, topo)
    ....
topo.ls <- surf.ls(4, topo)
    ....
topo.ls <- surf.ls(6, topo)
    ....
```

Notice how `eqscplot` is used to generate geometrically accurate plots.

Figure 15.1 shows trend surfaces for the `topo` dataset. The highest degree, 6, has 28 coefficients fitted from 52 points. The higher-order surfaces begin to show the difficulties of fitting by polynomials in two or more dimensions, when inevitably extrapolation is needed at the edges.

There are several other ways to show trend surfaces. Figure 15.2 uses Trellis to show a greyscale plot from `levelplot` and a perspective plot from `wireframe`. They were generated by

```
topo.ls <- surf.ls(4, topo)
trs <- trsurf <- trmat(topo.ls, 0, 6.5, 0, 6.5, 30)
trs[c("x", "y")] <- expand.grid(x = trs$x, y = tr$y)
plt1 <- levelplot(z ~ x * y, trs, aspect = 1,
    at = seq(650, 1000, 10),  xlab = "", ylab = "")
```

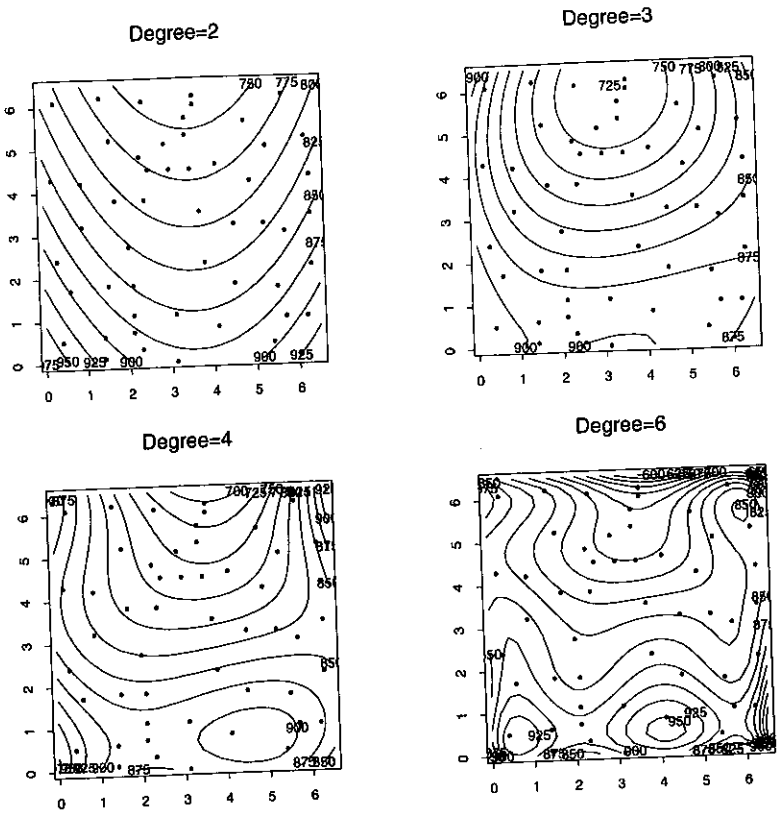

Figure 15.1: Trend surfaces for the `topo` dataset, of degrees 2, 3, 4 and 6.

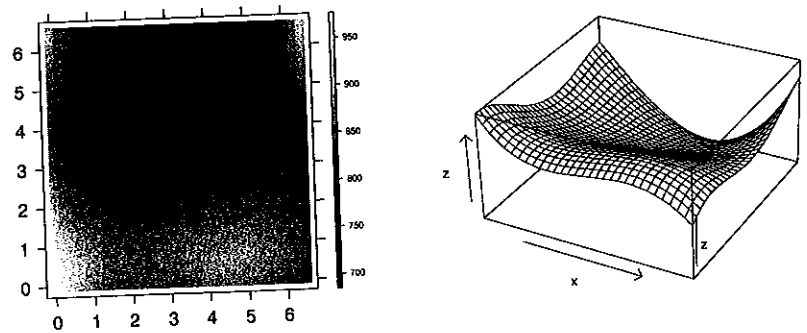

Figure 15.2: The quartic trend surfaces for the `topo` dataset.

```
plt2 <- wireframe(z ~ x * y, trs, aspect = c(1, 0.5),
                  screen = list(z = -30, x = -60))
print(plt1, position = c(0, 0, 0.5, 1), more = T)
print(plt2, position = c(0.45, 0, 1, 1))
```

S+Win Users of S-PLUS under Windows can use the rotatable 3D-plots in the GUI graphics, for example by

```
tr <- data.frame(x = trs$x, y = trs$y, z = as.vector(trs$z))
guiPlot(PlotType = "32 Color Surface", Dataset = "tr")
guiModify("Graph3D", Name = guiGetGraphName(),
          xSizeRatio = 2.2, ySizeRatio = 2.2)
```

where the final commands sets a square box. For R under Windows we can get a rotatable 3D-plot by (see page 69 for package `rgl`)

```
library(rgl)
persp3d(trsurf)
```

One difficulty with fitting trend surfaces is that in most applications the observations are not regularly spaced, and sometimes they are most dense where the surface is high (for example, in mineral prospecting). This makes it important to take the spatial correlation of the errors into consideration. We thus suppose that

$$Z(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{x})^T\boldsymbol{\beta} + \epsilon(\boldsymbol{x})$$

for a parametrized trend term such as (15.1) and a zero-mean spatial stochastic process $\epsilon(\boldsymbol{x})$ of errors. We assume that $\epsilon(\boldsymbol{x})$ possesses second moments, and has covariance matrix

$$C(\boldsymbol{x}, \boldsymbol{y}) = \mathrm{cov}\,(\epsilon(\boldsymbol{x}), \epsilon(\boldsymbol{y}))$$

(this assumption is relaxed slightly later). Then the natural way to estimate $\boldsymbol{\beta}$ is by *generalized least squares*, that is, to minimize

$$[Z(\boldsymbol{x}_i) - \boldsymbol{f}(\boldsymbol{x}_i)^T\boldsymbol{\beta}]^T[C(\boldsymbol{x}_i, \boldsymbol{x}_j)]^{-1}[Z(\boldsymbol{x}_i) - \boldsymbol{f}(\boldsymbol{x}_i)^T\boldsymbol{\beta}]$$

We need some simplified notation. Let $\boldsymbol{Z} = F\boldsymbol{\beta} + \boldsymbol{\epsilon}$ where

$$F = \begin{bmatrix} \boldsymbol{f}(x_1)^T \\ \vdots \\ \boldsymbol{f}(x_n)^T \end{bmatrix}, \qquad \boldsymbol{Z} = \begin{bmatrix} Z(\boldsymbol{x}_1) \\ \vdots \\ Z(\boldsymbol{x}_n) \end{bmatrix}, \qquad \boldsymbol{\epsilon} = \begin{bmatrix} \epsilon(\boldsymbol{x}_1) \\ \vdots \\ \epsilon(\boldsymbol{x}_n) \end{bmatrix}$$

and let $K = [C(\boldsymbol{x}_i, \boldsymbol{x}_j)]$. We assume that K is of full rank. Then the problem is to minimize

$$[\boldsymbol{Z} - F\boldsymbol{\beta}]^T K^{-1}[\boldsymbol{Z} - F\boldsymbol{\beta}] \tag{15.2}$$

The Choleski decomposition (Golub and Van Loan, 1989; Nash, 1990) finds a lower-triangular matrix L such that $K = LL^T$. (The S function `chol` is unusual in working with $U = L^T$.) Then minimizing (15.2) is equivalent to

$$\min_{\boldsymbol{\beta}} \|L^{-1}[\boldsymbol{Z} - F\boldsymbol{\beta}]\|^2$$

which reduces the problem to one of ordinary least squares. To solve this we use the QR decomposition (Golub and Van Loan, 1989) of $L^{-1}F$ as

$$QL^{-1}F = \begin{bmatrix} R \\ 0 \end{bmatrix}$$

for an orthogonal matrix Q and upper-triangular $P \times P$ matrix R. Write

$$QL^{-1}Z = \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix}$$

as the upper P and lower $n - P$ rows. Then $\widehat{\beta}$ solves

$$R\widehat{\beta} = Y_1$$

which is easy to compute as R is triangular.

Trend surfaces for the `topo` data fitted by generalized least squares are shown later (Figure 15.5), where we discuss the choice of the covariance function C.

Local trend surfaces

We have commented on the difficulties of using polynomials as global surfaces. There are two ways to make their effect local. The first is to fit a polynomial surface for each predicted point, using only the nearby data points. The function `loess` is of this class, and provides a wide range of options. By default it fits a quadratic surface by weighted least squares, the weights ensuring that 'local' data points are most influential. We only give details for the `span` parameter α less than one. Let $q = \lfloor \alpha n \rfloor$, and let δ denote the Euclidean distance to the qth nearest point to x. Then the weights are

$$w_i = \left[1 - \left(\frac{d(x, x_i)}{\delta}\right)^3\right]_+^3$$

for the observation at x_i. ($[\quad]_+$ denotes the positive part.) Full details of `loess` are given by Cleveland, Grosse and Shyu (1992). For our example we have (Figure 15.3):

```
par(mfcol = c(2,2), pty = "s")
topo.loess <- loess(z ~ x * y, topo, degree = 2, span = 0.25,
   normalize = F)
topo.mar <- list(x = seq(0, 6.5, 0.1), y = seq(0, 6.5, 0.1))
topo.lo <- predict(topo.loess, expand.grid(topo.mar), se = T)
eqscplot(topo.mar, xlab = "fit", ylab = "", type = "n")
contour(topo.mar$x, topo.mar$y, topo.lo$fit,
   levels = seq(700, 1000, 25), add = T)
points(topo)
eqscplot(trsurf, , xlab = "standard error", ylab = "", type = "n")
contour(topo.mar$x,topo.mar$y,topo.lo$se.fit,
   levels = seq(5, 25, 5), add = T)
points(topo)
title("Loess degree = 2")
topo.loess <- loess(z ~ x * y, topo, degree = 1, span = 0.25,
   normalize = F, xlab = "", ylab = "")
```

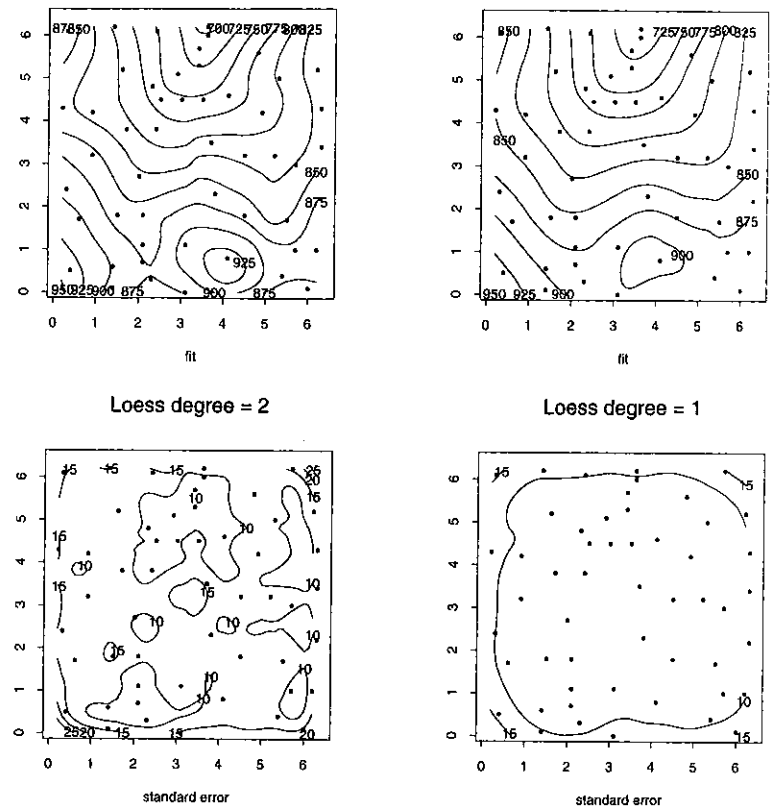

Figure 15.3: `loess` surfaces and prediction standard errors for the `topo` dataset.

We turn normalization off to use Euclidean distance on unscaled variables. Note that the predictions from `loess` are confined to the range of the data in each of the `x` and `y` directions even though we requested them to cover the square; this is a side effect of the algorithms used. The standard-error calculations are slow; `loess` is much faster without them.

Although `loess` allows a wide range of smoothing via its parameter `span`, it is designed for exploratory work and has no way to choose the smoothness except to 'look good'.

The Dirichlet tessellation[7] of a set of points is the set of *tiles*, each of which is associated with a data point, and is the set of points nearer to that data point than any other. There is an associated triangulation, the Delaunay triangulation, in which data points are connected by an edge of the triangulation if and only if their Dirichlet tiles share an edge. (Algorithms and examples are given in Ripley, 1981, §4.3.) There is S code in library section `delaunay` available from `statlib` (see page 464), and in the R packages `deldir` and `tripack` on CRAN.) Akima's (1978) fitting method fits a fifth-order trend surface within each triangle of the

[7] Also known as Voronoi or Thiessen polygons.

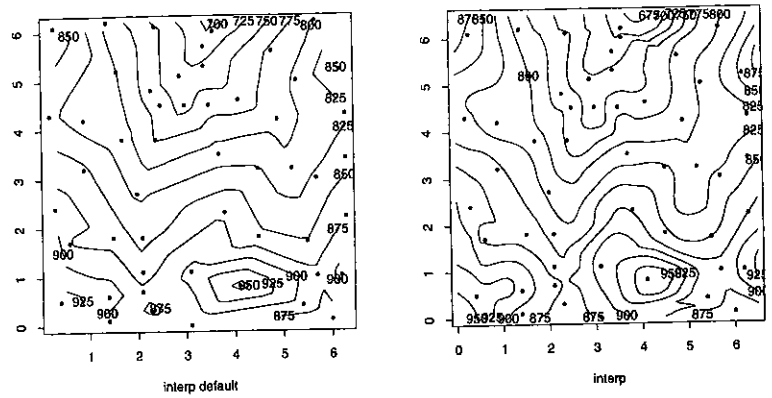

Figure 15.4: `interp` surfaces for the `topo` dataset.

Delaunay triangulation; details are given in Ripley (1981, §4.3). The S implementation is the function `interp`; Akima's example is in datasets `akima.x`, `akima.y` and `akima.z`. The method is forced to interpolate the data, and has no flexibility at all to choose the smoothness of the surface. The arguments `ncp` and `extrap` control details of the method: see the on-line help for details. For Figure 15.4 we used

```
# R: library(akima) # replace interp by interp.old
par(mfrow = c(1, 2), pty=  "s")
topo.int <- interp(topo$x, topo$y, topo$z)
eqscplot(topo.int, xlab = "interp default", ylab = "", type = "n")
contour(topo.int, levels = seq(600, 1000, 25), add = T)
points(topo)
topo.mar <- list(x = seq(0, 6.5, 0.1), y = seq(0, 6.5, 0.1))
topo.int2 <- interp(topo$x, topo$y, topo$z, topo.mar$x, topo.mar$y,
                    ncp = 4, extrap = T)
eqscplot(topo.int2, xlab = "interp", ylab = "", type = "n")
contour(topo.int2, levels = seq(600, 1000, 25), add = T)
points(topo)
```

15.2 Kriging

Kriging is the name of a technique developed by Matheron in the early 1960s for mining applications, which has been independently discovered many times. Journel and Huijbregts (1978) give a comprehensive guide to its application in the mining industry. See also Chilès and Delfiner (1999). In its full form, *universal kriging*, it amounts to fitting a process of the form

$$Z(x) = f(x)^T\beta + \epsilon(x)$$

by generalized least squares, predicting the value at x of both terms and taking their sum. Thus it differs from trend-surface prediction which predicts $\epsilon(x)$ by

zero. In what is most commonly termed *kriging*, the trend surface is of degree zero, that is, a constant.

Our derivation of the predictions is given by Ripley (1981, pp. 48–50). Let $k(\boldsymbol{x}) = [C(\boldsymbol{x}, \boldsymbol{x}_i)]$. The computational steps are as follows.

1. Form $K = [C(\boldsymbol{x}_i, \boldsymbol{y}_i)]$, with Choleski decomposition L.
2. Form F and $\boldsymbol{Z}$.
3. Minimize $\|L^{-1}\boldsymbol{Z} - L^{-1}F\boldsymbol{\beta}\|^2$, reducing $L^{-1}F$ to R.
4. Form $\boldsymbol{W} = \boldsymbol{Z} - F\widehat{\boldsymbol{\beta}}$, and $\boldsymbol{y}$ such that $L(L^T y) = \boldsymbol{W}$.
5. Predict $Z(\boldsymbol{x})$ by $\widehat{Z}(\boldsymbol{x}) = \boldsymbol{y}^T k(\boldsymbol{x}) + \boldsymbol{f}(\boldsymbol{x})^T\widehat{\boldsymbol{\beta}}$, with error variance given by $C(\boldsymbol{x}, \boldsymbol{x}) - \|\boldsymbol{e}\|^2 + \|\boldsymbol{g}\|^2$ where

$$L\boldsymbol{e} = k(\boldsymbol{x}), \qquad R^T\boldsymbol{g} = \boldsymbol{f}(\boldsymbol{x}) - (L^{-1}F)^T\boldsymbol{e}.$$

This recipe involves only linear algebra and so can be implemented in S, but our C version is about 10 times faster. For the `topo` data we have (Figure 15.5):

```
topo.ls <- surf.ls(2, topo)
trsurf <- trmat(topo.ls, 0, 6.5, 0, 6.5, 30)
eqscplot(trsurf, , xlab = "", ylab = "", type = "n")
contour(trsurf, levels = seq(600, 1000, 25), add = T)
points(topo); title("LS trend surface")

topo.gls <- surf.gls(2, expcov, topo, d = 0.7)
trsurf <- trmat(topo.gls, 0, 6.5, 0, 6.5, 30)
eqscplot(trsurf, , xlab = "", ylab = "", type = "n")
contour(trsurf, levels = seq(600, 1000, 25), add = T)
points(topo); title("GLS trend surface")

prsurf <- prmat(topo.gls, 0, 6.5, 0, 6.5, 50)
eqscplot(prsurf, , xlab = "", ylab = "", type = "n")
contour(prsurf, levels = seq(600, 1000, 25), add = T)
points(topo); title("Kriging prediction")
sesurf <- semat(topo.gls, 0, 6.5, 0, 6.5, 30)
eqscplot(sesurf, , xlab = "", ylab = "", type = "n")
contour(sesurf, levels = c(20, 25), add = T)
points(topo); title("Kriging s.e.")
```

Covariance estimation

To use either generalized least squares or kriging we have to know the covariance function C. We assume that

$$C(\boldsymbol{x}, \boldsymbol{y}) = c(d(\boldsymbol{x}, \boldsymbol{y})) \tag{15.3}$$

where $d()$ is Euclidean distance. (An extension known as *geometric anisotropy* can be incorporated by rescaling the variables, as we did for the Mahalanobis

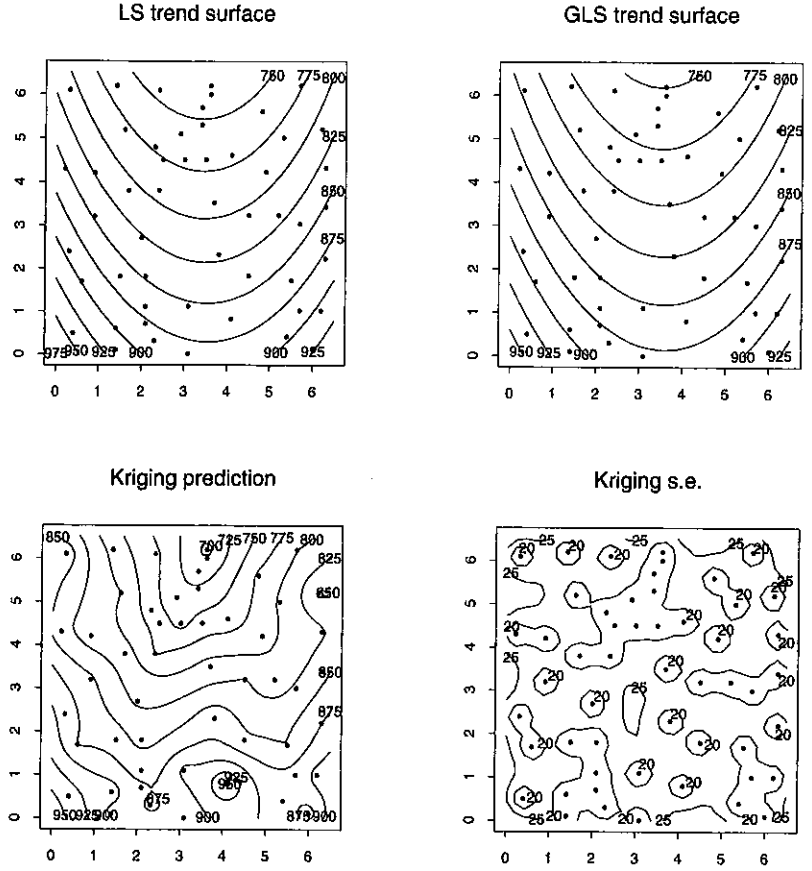

Figure 15.5: Trend surfaces by least squares and generalized least squares, and a kriged surface and standard error of prediction, for the `topo` dataset.

distance in Chapter 11.) We can compute a *correlogram* by dividing the distance into a number of bins and finding the covariance between pairs whose distance falls into that bin, then dividing by the overall variance.

Choosing the covariance is very much an iterative process, as we need the covariance of the residuals, and the fitting of the trend surface by generalized least squares depends on the assumed form of the covariance function. Furthermore, as we have residuals their covariance function is a biased estimator of c. In practice it is important to get the form right for small distances, for which the bias is least.

Although $c(0)$ must be one, there is no reason why $c(0+)$ should not be less than one. This is known in the kriging literature as a *nugget effect* since it could arise from a very short-range component of the process $Z(x)$. Another explanation is measurement error. In any case, if there is a nugget effect, the predicted surface will have spikes at the data points, and so effectively will not interpolate but smooth.

The kriging literature tends to work with the *variogram* rather than the covari-

ance function. More properly termed the semi-variogram, this is defined by

$$V(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{2} E[Z(\boldsymbol{x}) - Z(\boldsymbol{y})]^2$$

and is related to C by

$$V(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{2}[C(\boldsymbol{x}, \boldsymbol{x}) + C(\boldsymbol{y}, \boldsymbol{y})] - C(\boldsymbol{x}, \boldsymbol{y}) = c(0) - c(d(\boldsymbol{x}, \boldsymbol{y}))$$

under our assumption (15.3). However, since different variance estimates will be used in different bins, the empirical versions will not be so exactly related. Much heat and little light emerges from discussions of their comparison.

There are a number of standard forms of covariance functions that are commonly used. A nugget effect can be added to each. The exponential covariance has

$$c(r) = \sigma^2 \exp -r/d$$

the so-called Gaussian covariance is

$$c(r) = \sigma^2 \exp -(r/d)^2$$

and the spherical covariance is in two dimensions

$$c(r) = \sigma^2 \left[1 - \frac{2}{\pi}\left(\frac{r}{d}\sqrt{1 - \frac{r^2}{d^2}} + \sin^{-1}\frac{r}{d}\right)\right]$$

and in three dimensions (but also valid as a covariance function in two)

$$c(r) = \sigma^2 \left[1 - \frac{3r}{2d} + \frac{r^3}{2d^3}\right]$$

for $r \leqslant d$ and zero for $r > d$. Note that this is genuinely local, since points at a greater distance than d from $\boldsymbol{x}$ are given zero weight at step 5 (although they do affect the trend surface).

We promised to relax the assumption of second-order stationarity slightly. As we only need to predict residuals, we only need a covariance to exist in the space of linear combinations $\sum a_i Z(\boldsymbol{x}_i)$ that are orthogonal to the trend surface. For degree 0, this corresponds to combinations with sum zero. It is possible that the variogram is finite, without the covariance existing, and there are extensions to more general trend surfaces given by Matheron (1973) and reproduced by Cressie (1991, §5.4). In particular, we can always add a constant to c without affecting the predictions (except perhaps numerically). Thus if the variogram v is specified, we work with covariance function $c = \text{const} - v$ for a suitably large constant. The main advantage is in allowing us to use certain functional forms that do not correspond to covariances, such as

$$v(d) = d^\alpha, 0 \leqslant \alpha < 2 \quad \text{or} \quad d^3 - \alpha d$$

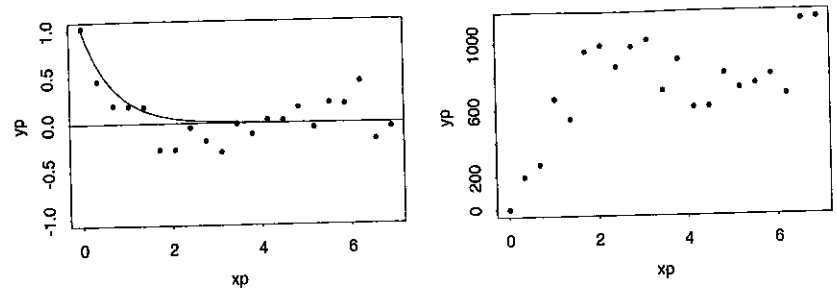

Figure 15.6: Correlogram (left) and variogram (right) for the residuals of `topo` dataset from a least-squares quadratic trend surface.

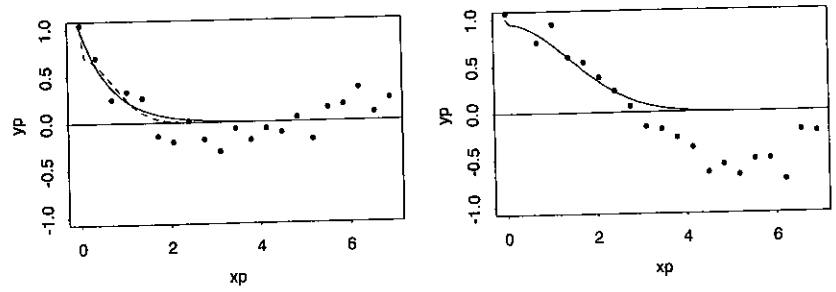

Figure 15.7: Correlograms for the `topo` dataset: (left) residuals from quadratic trend surface showing exponential covariance (solid) and Gaussian covariance (dashed); (right) raw data with fitted Gaussian covariance function.

The variogram $d^2 \log d$ corresponds to a thin-plate spline in $\mathbb{R}^2$ (see Wahba, 1990, and the review in Cressie, 1991, §3.4.5).

Our functions `correlogram` and `variogram` allow the empirical correlogram and variogram to be plotted and functions `expcov`, `gaucov` and `sphercov` compute the exponential, Gaussian and spherical covariance functions (the latter in two and three dimensions) and can be used as arguments to `surf.gls`. For our running example we have

```
topo.kr <- surf.ls(2, topo)
correlogram(topo.kr, 25)
d <- seq(0, 7, 0.1)
lines(d, expcov(d, 0.7))
variogram(topo.kr, 25)
```

See Figure 15.6. We then consider fits by generalized least squares.

```
## left panel of Figure 15.7
topo.kr <- surf.gls(2, expcov, topo, d=0.7)
correlogram(topo.kr, 25)
lines(d, expcov(d, 0.7))
lines(d, gaucov(d, 1.0, 0.3), lty = 3) # try nugget effect
```

```
## right panel
topo.kr <- surf.ls(0, topo)
correlogram(topo.kr, 25)
lines(d, gaucov(d, 2, 0.05))

## top row of Figure 15.8
topo.kr <- surf.gls(2, gaucov, topo, d = 1, alph = 0.3)
prsurf <- prmat(topo.kr, 0, 6.5, 0, 6.5, 50)
eqscplot(prsurf, , xlab = "fit", ylab = "", type = "n")
contour(prsurf, levels = seq(600, 1000, 25), add = T)
points(topo)
sesurf <- semat(topo.kr, 0, 6.5, 0, 6.5, 25)
eqscplot(sesurf, , xlab = "standard error", ylab = "", type = "n")
contour(sesurf, levels = c(15, 20, 25), add = T)
points(topo)

## bottom row of Figure 15.8
topo.kr <- surf.gls(0, gaucov, topo, d = 2, alph = 0.05,
                    nx = 10000)
prsurf <- prmat(topo.kr, 0, 6.5, 0, 6.5, 50)
eqscplot(prsurf, , xlab = "fit", ylab = "", type = "n")
contour(prsurf, levels = seq(600, 1000, 25), add = T)
points(topo)
sesurf <- semat(topo.kr, 0, 6.5, 0, 6.5, 25)
eqscplot(sesurf, , xlab = "standard error", ylab = "", type = "n")
contour(sesurf, levels = c(15, 20, 25), add = T)
points(topo)
```

We first fit a quadratic surface by least squares, then try one plausible covariance function (Figure 15.7). Re-fitting by generalized least squares suggests this function and another with a nugget effect, and we predict the surface from both. The first was shown in Figure 15.5, the second in Figure 15.8. We also consider not using a trend surface but a longer-range covariance function, also shown in Figure 15.8. (The small nugget effect is to ensure numerical stability as without it the matrix K is very ill-conditioned; the correlations at short distances are very near one. We increased `nx` for a more accurate lookup table of covariances.)

15.3 Point Process Analysis

A spatial point pattern is a collection of n points within a region $D \subset \mathbb{R}^2$. The number of points is thought of as random, and the points are considered to be generated by a stationary isotropic point process in $\mathbb{R}^2$. (This means that there is no preferred origin or orientation of the pattern.) For such patterns probably the most useful summaries of the process are the first and second moments of the counts $N(A)$ of the numbers of points within a set $A \subset D$. The first moment can be specified by a single number, the *intensity* λ giving the expected number of points per unit area, obviously estimated by n/a where a denotes the area of D.

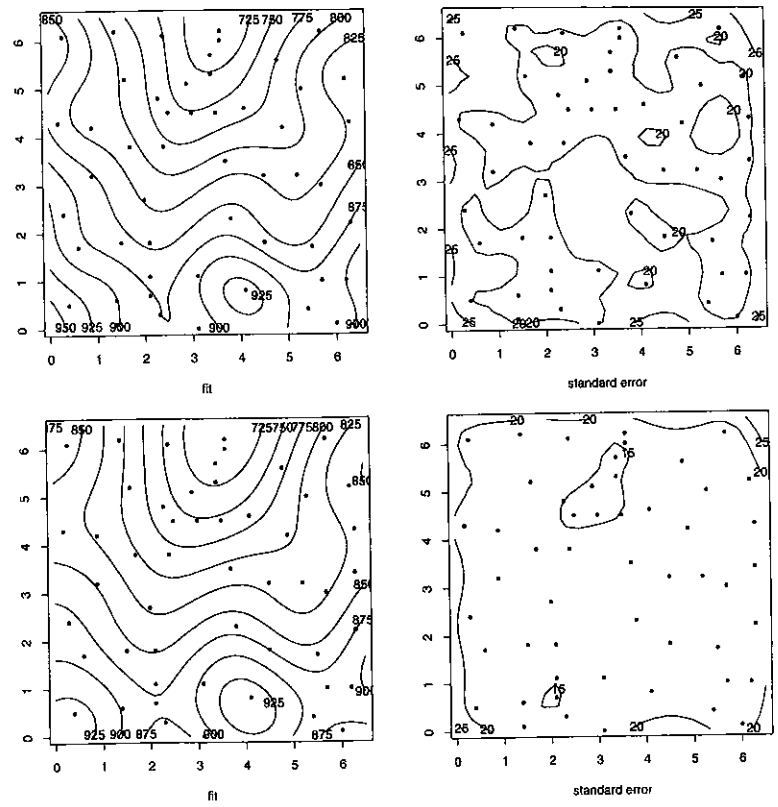

Figure 15.8: Two more kriged surfaces and standard errors of prediction for the `topo` dataset. The top row uses a quadratic trend surface and a nugget effect. The bottom row is without a trend surface.

The second moment can be specified by Ripley's K function. For example, $\lambda K(t)$ is the expected number of points within distance t of a point of the pattern. The benchmark of complete randomness is the Poisson process, for which $K(t) = \pi t^2$, the area of the search region for the points. Values larger than this indicate clustering on that distance scale, and smaller values indicate regularity. This suggests working with $L(t) = \sqrt{K(t)/\pi}$, which will be linear for a Poisson process.

We only have a single pattern from which to estimate K or L. The definition in the previous paragraph suggests an estimator of $\lambda K(t)$; average over all points of the pattern the number seen within distance t of that point. This would be valid but for the fact that some of the points will be outside D and so invisible. There are a number of edge-corrections available, but that of Ripley (1976) is both simple to compute and rather efficient. This considers a circle centred on the point x and passing through another point y. If the circle lies entirely within D, the point is counted once. If a proportion $p(x, y)$ of the circle lies within D, the point is counted as $1/p$ points. (We may want to put a limit on small p, to reduce

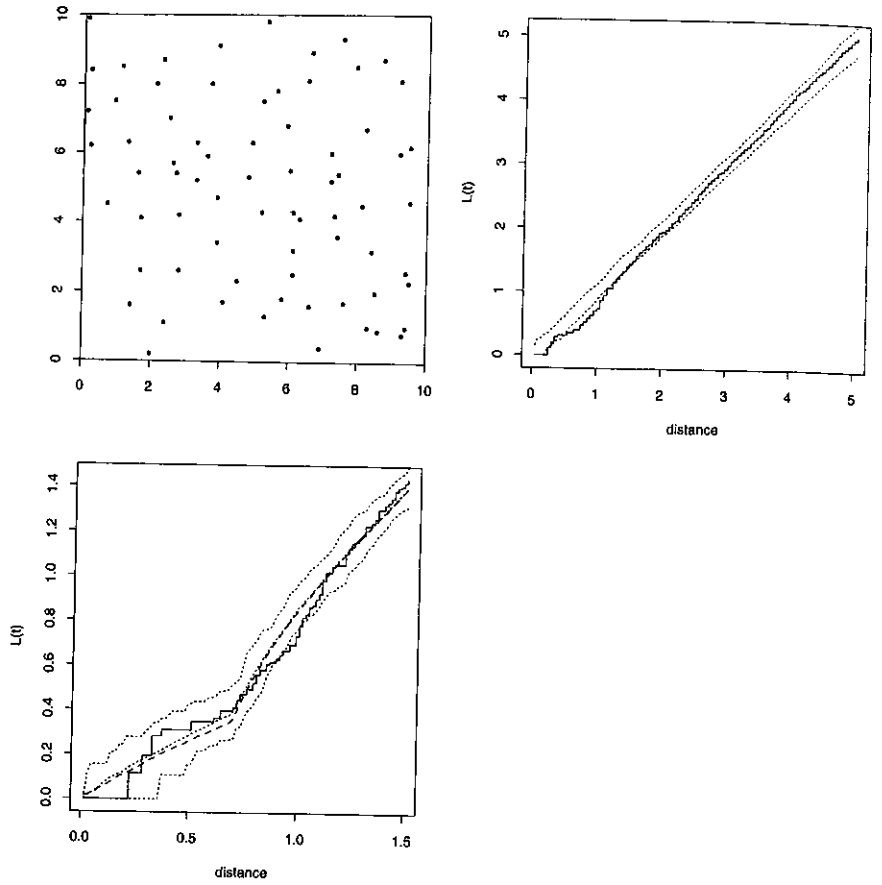

Figure 15.9: The Swedish pines dataset from Ripley (1981), with two plots of $L(t)$. That at the upper right shows the envelope of 100 binomial simulations, that at the lower left the average and the envelope (dotted) of 100 simulations of a Strauss process with $c = 0.2$ and $R = 0.7$. Also shown (dashed) is the average for $c = 0.15$. All units are in metres.

the variance at the expense of some bias.) This gives an estimator $\lambda\widehat{K}(t)$ which is unbiased for t up to the circumradius of D (so that it is possible to observe two points $2t$ apart). Since we do not know λ, we estimate it by $\hat{\lambda} = n/a$. Finally

$$\widehat{K}(t) = \frac{a}{n^2} \sum_{x \in D, d(y,x) \leqslant t} \frac{1}{p(x, y)}$$

and obviously we estimate $L(t)$ by $\sqrt{\widehat{K}(t)/\pi}$. We find that on square-root scale the variance of the estimator varies little with t.

Our example is the Swedish pines data from Ripley (1981, §8.6). This records 72 trees within a 10-metre square. Figure 15.9 shows that $\widehat{L}$ is not straight, and comparison with simulations from a binomial process (a Poisson process condi-

tioned on $N(D) = n$, so n independently uniformly distributed points within D) shows that the lack of straightness is significant. The upper two panels of Figure 15.9 were produced by the following code:

```
library(spatial)
pines <- ppinit("pines.dat")
par(mfrow = c(2, 2), pty = "s")
plot(pines, xlim = c(0, 10), ylim = c(0, 10),
     xlab = "", ylab = "", xaxs = "i", yaxs = "i")
plot(Kfn(pines,5), type = "s", xlab = "distance", ylab = "L(t)")
lims <- Kenvl(5, 100, Psim(72))
lines(lims$x, lims$l, lty = 2)
lines(lims$x, lims$u, lty = 2)
```

The function `ppinit` reads the data from the file and also the coordinates of a rectangular domain D. The latter can be reset, or set up for simulations, by the function `ppregion`. (It *must* be set for each session.) The function `Kfn` returns an estimate of $L(t)$ and other useful information for plotting, for distances up to its second argument `fs` (for full-scale).

The functions `Kaver` and `Kenvl` return the average and, for `Kenvl`, also the extremes of K-functions (on L scale) for a series of simulations. The function `Psim(n)` simulates the binomial process on `n` points within the domain D, which has already been set.

Alternative processes

We need to consider alternative point processes to the Poisson. One of the most useful for regular point patterns is the so-called Strauss process, which is simulated by `Strauss(n, c, r)`. This has a density of n points proportional to

$$c^{\text{number of } R\text{-close pairs}}$$

and so has $K(t) < \pi t^2$ for $t \leqslant R$ (and up to about $2R$). For $c = 0$ we have a 'hard-core' process that never generates pairs closer than R and so can be envisaged as laying down the centres of non-overlapping discs of diameter $\mathtt{r} = R$.

Figure 15.9 also shows the average and envelope of the L-plots for a Strauss process fitted to the pines data by Ripley (1981). There the parameters were chosen by trial-and-error based on a knowledge of how the L-plot changed with (c, R). Ripley (1988) considers the estimation of c for known R by the pseudo-likelihood. This is done by our function `pplik` and returns an estimate of about $c = 0.15$ ('about' since it uses numerical integration). As Figure 15.9 shows, the difference between $c = 0.2$ and $c = 0.15$ is small. We used the following code:

```
ppregion(pines)
plot(Kfn(pines, 1.5), type = "s",
     xlab = "distance", ylab = "L(t)")
lims <- Kenvl(1.5, 100, Strauss(72, 0.2, 0.7))
lines(lims$x, lims$a, lty = 2)
```

```
lines(lims$x, lims$l, lty = 2)
lines(lims$x, lims$u, lty = 2)
pplik(pines, 0.7)
lines(Kaver(1.5, 100, Strauss(72, 0.15, 0.7)), lty = 3)
```

The theory is given by Ripley (1988, p. 67). For a point $\xi \in D$ let $t(\xi)$ denote the number of points of the pattern within distance t of ξ. Then the pseudo-likelihood estimator solves

$$\frac{\int_D t(\xi) c^{t(\xi)} \, \mathrm{d}\xi}{\int_D c^t(\xi) \, \mathrm{d}\xi} = \frac{\#(\, R\text{–close pairs}\,)}{n} = \frac{n\widehat{K}(R)}{a}$$

and the left-hand side is an increasing function of c. The function `pplik` uses the S-PLUS function `uniroot` to find a solution in the range $(0, 1]$.

Other processes for which simulation functions are provided are the binomial process (`Psim(n)`) and Matérn's sequential spatial inhibition process (`SSI(n, r)`), which sequentially lays down centres of discs of radius r that do not overlap existing discs.