Modellierung dynamischer und räumlicher Prozesse practical exercises

Edzer J. Pebesma
Institute for Geoinformatics
Univesity of Münster

January 7, 2008

Contents

1 Introduction to R **2**
1.1 S, S-Plus and R . . . 2
1.2 Downloading and installing R . . . 3
1.3 Starting R; saving or resoring an R session . . . 3

2 Generating random walk data **4**

3 Time series analysis of meteo data **5**
3.1 Exploratory data analysis . . . 5
3.2 Fitting a periodic component . . . 6
3.3 Fitting AR model to the residuals . . . 6
3.4 Model selection with AIC . . . 7
3.5 Prediction with an AR model . . . 7

4 One-dimensional search with golden section **8**

5 Non-linear least squares with `nls` **9**
5.1 Initial conditions . . . 10
5.2 Iteration . . . 10
5.3 Confidence intervals . . . 11
5.4 Prediction . . . 11

6 Fun with Metropolis **12**
6.1 An R function for MCMC . . . 12
6.2 The effect of sigma . . . 13
6.3 Fixing the phase . . . 15
6.4 Computing summary statistics . . . 15
6.5 Initial values and Burn-in . . . 16

7 Spatial modelling: introductory matter **16**
7.1 Data sets . . . 16
7.1.1 meuse . . . 16
7.2 Models: the formula interface; linear regression . . . 17
7.3 Spatial data in R: the sp package . . . 18
7.3.1 Points . . . 18
7.3.2 Grids . . . 19

8 Geostatistics **19**
8.1 Exploratory data analysis . . . 20
8.2 Simple interpolation algorithms . . . 20
8.2.1 Trend surface analysis . . . 20
8.2.2 Inverse distance interpolation . . . 21
8.2.3 Thiessen polygons . . . 21
8.3 Spatial prediction with multiple linear regression . . . 22
8.4 Spatial correlation: the h-scatterplot . . . 24
8.5 Spatial correlation: the variogram cloud . . . 25
8.6 Spatial correlation: the variogram . . . 26
8.7 Sample variogram and variogram model . . . 26
8.8 Simple and ordinary kriging . . . 28
8.9 Universal kriging . . . 31

9 Diffusion and partial differential equations **33**

1 Introduction to R

1.1 S, S-Plus and R

S is a language for data analysis and statistical computing. It has currently two implementations: one commercial, called S-Plus, and one free, open source implementation called R. Both have advantages and disadvantages.

S-Plus has the (initial) advantage that you can work with much of the functionality without having to know the S language: it has a large, extensively developed graphical user interface. R works by typing commands (although the R packages `Rcmdr` and `JGR` do provide graphical user interfaces). Compare typing commands with sending SMS messages: at first it seems clumsy, but when you get to speed it is fun.

Compared to working with a graphical program that requires large amounts of mouse clicks to get something done, working directly with a language has a couple of notable advantages:

- after a few commands, you can exactly (i) see and (ii) replicate what you did, and send the list of commands to someone else to replicate your analysis;
- when you have a large number of analyses that take much time, you can easily perform them as batches, possibly on a remote computer or on a cluster of computers;

- when you tend to repeat a set of commands often, you can very easily turn them into an S function, which then extends the language, is available forever, and can be redistributed. In the world of R, users soon become developers–vice versa, most developers are users in the first place.

Of course the disadvantage (as you may consider it right now) is that you need to learn a few new ideas.

Although S-Plus and R are both full-grown statistical computing engines, throughout this course we use R for the following reasons: (i) it is free (S-Plus may cost in the order of 10,000 USD), (ii) development and availability of novel statistical computation methods is much faster in R.

Recommended text books that deal with both R and S-Plus are e.g.:

1. W. Venables, B. Ripley: Modern applied statistics with S; Springer (emphasizes on statistical analysis)
2. W. Venables, B. Ripley: S Programming; Springer (emphasizes on programming)
3. J. Chambers: Programming with Data; Springer (fundamental; written by one of the designers of S)

Further good books on R are:

1. P. Dalgaard, Introductory Statistics with R; Springer

1.2 Downloading and installing R

To download R on a MS-Windows computer, do the following:

1. point your web browser to `http://www.r-project.org`
2. click on the link to CRAN (abbreviation of the Comprehensive R Archive Network)
3. choose a nearby mirror
4. click on the MS-Windows version of R
5. open the self-extracting executable, and follow the instructions

Instructions for installing R on a Macintosh computer are also found on CRAN. For installing R on a linux machine, you can download the R source code from CRAN, and follow the compilation and installation instructions; usually this involves the commands `./configure; make; make install`. There may be binary distributions for linux distributions such as RedHat FC, or Debian unstable.

1.3 Starting R; saving or resoring an R session

You can start an R session on a MS-Windows machine by double-clicking the R icon, or through the Start menu (Note that this is not true in the ifgi CIP pools: here you have to find the executable in the right direction on the C: drive). When you start R, you'll see the R console with the `>` prompt, on which you can give commands.

Another way of starting R is when you have a data file associated with R (meaning it ends on .RData). If you click on this file, R is started and reads at start-up the data in this file.

Try this using the meteo.RData file. Next, try the following commands:

```
> objects()
> summary(meteo)
> a = 1
> objects()
> print(a)
> a
```

(Note that you can copy and paste the commands *including the* > to the R prompt, only if you use "paste as commands")

You can see that your R workspace now has two objects in it. You can find out which class these objects belong by

```
> class(a)
> class(meteo)
```

If you want to remove an object, use

```
> rm(a)
```

If you want so safe a (modified) work space to a file, you use

```
> save.image(file = "meteo.RData")
```

2 Generating random walk data

Random walk is generated by the process

$$y(t_i) = y(t_{i-1}) + e(t_i)$$

with, e.g., initial conditions $y(t_0) = 0$ and $e(t_i)$ a random value from the standard normal distribution (mean 0, variance 1). Random normal deviates are generated by the function rnorm.

```
> x = rnorm(10)
> x
> cumsum(x)
> plot(cumsum(rnorm(1000)), type = "l")
> plot(cumsum(rnorm(1000)), type = "l")
> plot(cumsum(rnorm(1000)), type = "l")
> plot(cumsum(rnorm(10000)), type = "l")
> plot(cumsum(rnorm(1e+05)), type = "l")
```

Exercise 2.1 Does a RW process in general increase, decrease, or does it not have a dominant direction?

```
> var(cumsum(rnorm(10)))
> var(cumsum(rnorm(100)))
> var(cumsum(rnorm(1000)))
> var(cumsum(rnorm(10000)))
> var(cumsum(rnorm(1e+05)))
> var(cumsum(rnorm(1e+06)))
```

Exercise 2.2 Explain why a RW processs has a variance that increases with its length

3 Time series analysis of meteo data

3.1 Exploratory data analysis

The meteo data set is a time series with 1-minute data, collected during a field exercise in the Haute Provence (France), in a small village called Hauteville, near Serres.

What follows is a short description of the variables.

ID ID of this data logger

year year

julian.day day number starting on Jan 1st

time time, not decimal but as hhmm

T.outside outside temperature in degrees Celcius

pressure 1000 - pressure, mbar

humidity humidity, as percentage

X unknown

windspeed wind speed

std.dev. variability of wind speed

Wind.dir wind direction, in degrees

std.dev..1 variability of wind direction (useless)

TippingBucket tipping bucket: rain fall in mm/min

date ISO time

Look at the summary of the meteo data, and plot temperature

```
> summary(meteo)
> plot(T.outside ~ date, meteo, type = "l")
```

3.2 Fitting a periodic component

We will now fit a periodic component to this data, using a non-linear fit.

```
> f = function(x) sum((meteo$T.outside - (x[1] + x[2] * sin(pi *
     (meteo$hours + x[3])/12)))^2)
> nlm(f, c(0, 0, 0))
```

We used the function `nlm` that will minimize the function in its first argument (`f`), using the initial guess of the parameter vector in its second argument. The function `f` computes the sum of squared residuals:

$$\sum_{i=1}^{n}(\text{observed}_i - \text{predicted}_i)^2$$

Exercise 3.3 How many parameters were fitted?

We will now plot observations and fitted model together:

```
> plot(T.outside ~ date, meteo, type = "l")
> meteo$T.per = 18.2 - 4.9 * sin(pi * (meteo$hours + 1.6)/12)
> lines(T.per ~ date, meteo, col = "red")
```

Exercise 3.4 What is the interpretation of the fitted parameters? (if you need to guess, modify them and replot)

We can now also plot the residual from this fitted model:

```
> plot(T.outside - T.per ~ date, meteo, type = "l")
> title("difference from fitted sinus")
```

3.3 Fitting AR model to the residuals

The AR(p) model is defined as

$$y_t = \sum_{j=1}^{p} \phi_j y_{t-p} + e_t$$

with e_t a white noise process. For $p = 1$ this simplifies to

$$y_t = \phi_1 y_{t-1} + e_t.$$

Now try to model the residual process as an AR(5) process, and look at the partial correlations.

```
> an = meteo$T.outside - meteo$T.per
> an.ar5 = arima(an, c(5, 0, 0))
> an.ar5
> acf(an, type = "partial")
> acf(residuals(an.ar5), type = "partial")
```

(Note that this generates 2 plots)

Exercise 3.5 Does the `an` process exhibit temporal correlation for lags larger than 0?

Exercise 3.6 Does the `residuals(an.ar5)` process still exhibit temporal correlation for lags larger than 0?

Exercise 3.7 What is the class of the object returned by `arima`?

Let us see what we can do with such an object.

```
> methods(class = "Arima")
> tsdiag(an.ar5)
```

Exercise 3.8 Try to explain what you see in the first two plots obtained

3.4 Model selection with AIC

```
> temp = meteo$T.outside
> arima(temp, c(1, 0, 0))$aic
> arima(temp, c(2, 0, 0))$aic
> arima(temp, c(3, 0, 0))$aic
> arima(temp, c(4, 0, 0))$aic
> arima(temp, c(5, 0, 0))$aic
> arima(temp, c(6, 0, 0))$aic
> arima(temp, c(7, 0, 0))$aic
> arima(temp, c(8, 0, 0))$aic
> arima(temp, c(9, 0, 0))$aic
> arima(temp, c(10, 0, 0))$aic
```

Exercise 3.9 Which model has the smallest AIC?

3.5 Prediction with an AR model

Let us now work with the AR(6) model for the temperature, ignoring the periodic (diurnal) component. Make sure you have "plot recording" on (activate the plot window to get this option).

```
> x = arima(temp, c(6, 0, 0))
> plot(meteo$T.outside, xlim = c(9860, 9900), type = "l")
> x.pr = as.numeric(predict(x, 10)$pred)
> x.se = as.numeric(predict(x, 10)$se)
> lines(9891:9900, x.pr, col = "red")
> lines(9891:9900, x.pr + 2 * x.se, col = "green")
> lines(9891:9900, x.pr - 2 * x.se, col = "green")
> title("predicting 10 mins")
> plot(meteo$T.outside, xlim = c(9400, 10000), type = "l")
> x.pr = as.numeric(predict(x, 110)$pred)
```

```
> x.se = as.numeric(predict(x, 110)$se)
> lines(9891:10000, x.pr, col = "red")
> lines(9891:10000, x.pr + 2 * x.se, col = "green")
> lines(9891:10000, x.pr - 2 * x.se, col = "green")
> title("predicting 110 mins")
> plot(meteo$T.outside, xlim = c(8000, 11330), type = "l")
> x.pr = as.numeric(predict(x, 1440)$pred)
> x.se = as.numeric(predict(x, 1440)$se)
> lines(9891:11330, x.pr, col = "red")
> lines(9891:11330, x.pr + 2 * x.se, col = "green")
> lines(9891:11330, x.pr - 2 * x.se, col = "green")
> title("predicting 1 day")
> plot(meteo$T.outside, xlim = c(1, 19970), type = "l")
> x.pr = as.numeric(predict(x, 10080)$pred)
> x.se = as.numeric(predict(x, 10080)$se)
> lines(9891:19970, x.pr, col = "red")
> lines(9891:19970, x.pr + 2 * x.se, col = "green")
> lines(9891:19970, x.pr - 2 * x.se, col = "green")
> title("predicting 1 week")
```

Exercise 3.10 Where does, for long-term forecasts, converge the predicted value to?

Now compare this with prediction using an AR(6) model for the residual with respect to the daily cycle:

```
> plot(meteo$T.outside, xlim = c(1, 19970), type = "l")
> x.an = arima(an, c(6, 0, 0))
> x.pr = as.numeric(predict(x.an, 10080)$pred)
> x.se = as.numeric(predict(x.an, 10080)$se)
> hours.all = c(meteo$hours, max(meteo$hours) + (1:10080)/60)
> T.per = 18.2 - 4.9 * sin(pi * (hours.all + 1.6)/12)
> lines(T.per, col = "blue")
> hours.pr = c(max(meteo$hours) + (1:10080)/60)
> T.pr = 18.2 - 4.9 * sin(pi * (hours.pr + 1.6)/12)
> lines(9891:19970, T.pr + x.pr, col = "red")
> lines(9891:19970, T.pr + x.pr + 2 * x.se, col = "green")
> lines(9891:19970, T.pr + x.pr - 2 * x.se, col = "green")
> title("predicting 1 week")
```

Exercise 3.11 Where does now, for long-term forecasts, converge the predicted value to?

4 One-dimensional search with golden section

The function `optimize` (or, for that sake, `optimise`) optimizes a one-dimensional function. It uses a combination of golden section search and another approach.

Exercise 4.12 Which other approach?

Let us fit the phase of a `sin` function.

```
> phase = 1.23456789
> x = runif(10) * 2 * pi
> y = sin(x + phase)
> f = function(x, y, phase) sum(y - sin(x + phase))^2
> optimize(f, c(0, pi), x = x, y = y)
```

The optimization looks quite good, but approximates.

Exercise 4.13 What was the tolerance value used here?

We can make the tolerance smaller, as in

```
> optimize(f, c(0, pi), x = x, y = y, tol = 1e-08)
```

Exercise 4.14 Is this fit acceptable?

We can check whether more significant digits were fitted correctly:

```
> out = optimize(f, c(0, pi), x = x, y = y, tol = 1e-08)
> print(out$minimum, digits = 10)
```

Exercise 4.15 did `optimize` get the full 9 (or more) digits right?

5 Non-linear least squares with `nls`

As a preparation, go through the course slides; if you missed the lecture you may want to go through the wikipedia pages for golden section search and the Gauss-Newton algorithm.

Function `nls` uses a formula as its first argument, and a data frame as its second argument. In the introduction chapter 11, http://cran.r-project.org/doc/manuals/R-intro.html#Statistical-models-in-R, statistical models expressed as formula are explained. The general explanation here is on linear (and generalized linear) models.

Formulas form perhaps the most powerful feature of the S language. They are used everywhere when specifying some sort of statistical dependency. The general form, for dependent variable `y` and independent variable `x`, is

$$y \sim x,$$

in words `y` tilde `x`.

Today however, we use formulas for non-linear models. The difference is that in linear models the coefficients are implicit, e.g. linear regression uses the function `lm` (linear model), and for instance

```
> lm(T.outside ~ hours, meteo)
> summary(lm(T.outside ~ hours, meteo))
```

fits and summarized the linear regression model

$$\text{T.outside} = \beta_0 + \beta_1 \text{hours} + e,$$

Here, β_0 (intercept) and β_1 (slope) are implicit, and not named in the formula.

For non-linear models, we need to name coefficients explicit. In the example below, the now familiar sinus model is refitted using function `nls`, which uses Gauss-Newton optimization, and the coefficients are named `a`, `b` and `c` in the formula.

5.1 Initial conditions

Try fit the following two models:

```
> nls(T.outside ~ a + b * sin(pi * (hours + c)/12), meteo, c(a = 0,
     b = 1, c = 0))
> nls(T.outside ~ a + b * sin(pi * (hours + c)/12), meteo, c(a = 0,
     b = 1, c = 2))
```

The two fits provide different outcomes.

Exercise 5.16 What does the third argument to `nls` contain?

Exercise 5.17 Which of the fitted models is better?

Exercise 5.18 How can you explain the differences?

The following fit:

```
> nls(T.outside ~ a + b * sin(pi * (hours + c)/12), meteo, c(a = 0,
     b = 0, c = 0))
```

gives an error message.

Exercise 5.19 Explain why the error occured.

Exercise 5.20 Which element(s) of the Jacobian may have been difficult to compute in the first iteration?

5.2 Iteration

The consecutive steps of the optimization are shown when a trace is requested.

```
> nls(T.outside ~ a + b * sin(pi * (hours + c)/12), meteo, c(a = 0,
     b = 1, c = 0), trace = TRUE)
```

Exercise 5.21 What is the meaning of the four columns of additional output?

Exercise 5.22 Why is more than one step needed?

In the following case

```
> nls(T.outside ~ a + b * sin(pi * (hours - 10.4)/12), meteo, c(a = 0,
     b = 1), trace = TRUE)
```

convergence takes place after a single iteration.

Exercise 5.23 Why was this predictable?
Compare the following two fits:

```
> nls(T.outside ~ a + b * sin(pi * (hours + c)/12), meteo, c(a = 0,
     b = 1, c = 0), trace = TRUE)
> nls(T.outside ~ a + b * sin(pi * (hours + c)/12), meteo, c(a = 0,
     b = 1, c = 0), nls.control(tol = 0.01), trace = TRUE)
> nls(T.outside ~ a + b * sin(pi * (hours + c)/12), meteo, c(a = 0,
     b = 1, c = 0), nls.control(maxiter = 3), trace = TRUE)
> nls(T.outside ~ a + b * sin(pi * (hours + c)/12), meteo, c(a = 0,
     b = 1, c = 0), nls.control(tol = 1e-12), trace = TRUE)
> nls(T.outside ~ a + b * sin(pi * (hours + c)/12), meteo, c(a = 0,
     b = 1, c = 0), nls.control(maxiter = 100, tol = 2.2e-09,
     minFactor = 1e-10), trace = TRUE)
```

5.3 Confidence intervals

Confidence intervals can be computed (or approximated) by the appropriate methods available in library MASS:

```
> library(MASS)
> t.nls = nls(T.outside ~ a + b * sin(pi * (hours + c)/12), meteo,
     c(a = 0, b = 1, c = 2))
> confint(t.nls)
```

Exercise 5.24 Which coefficients are, with 95% confidence, different from zero?

5.4 Prediction

Try

```
> plot(T.outside ~ hours, meteo)
> lines(predict(t.nls) ~ hours, meteo, col = "red")
```

to see how predictions are generated for the data points. Interestingly, the two commands

```
> predict(t.nls)[1:10]
> predict(t.nls, se.fit = TRUE)[1:10]
> predict(t.nls, interval = "prediction")[1:10]
```

all do the same. But that is documented behaviour; see `?predict.nls`. Prediction errors or intervals are *much harder* to do for non-linear models than for linear models.

6 Fun with Metropolis

The Metropolis algorithm is a simplified version of the Metropolis-Hastings algorithm (search English wikipedia), and provides a search algorithm for finding global optima, and sampling the (posterior) parameter distribution, given the data.

Let $P(\theta)$ be the probability that parameter value θ has the right value, given the data.

The metropolis algorithm considers proposal values θ' for the parameter vector, and accepts if, given the current value θ_i, the ratio

$$\alpha = \frac{P(\theta')}{P(\theta_i)}$$

either when

- α is larger than one (meaning θ' is an improvement), or
- when a random value drawn uniformly between 0 and 1 is smaller than α, meaning that we accept worse parameters quite often if they're not too much worse.

6.1 An R function for MCMC

Consider the following function, and try to understand what is going on:

```
> Metropolis = function(theta0, sigma, y, fn, n = 1000) {
     m = length(theta0)
     out = matrix(NA, n, m)
     out[1, ] = theta0
     SSy = sum((y - mean(y))^2)
     accept = 0
     for (i in 2:n) {
         proposal = rnorm(m, out[i - 1, ], sigma)
         residualSumSquaresProp = sum((y - fn(proposal))^2)/SSy
         residualSumSquares0 = sum((y - fn(out[i - 1, ]))^2)/SSy
         ratio = exp(-residualSumSquaresProp + residualSumSquares0)
         if (runif(1) < ratio) {
             out[i, ] = proposal
             accept = accept + 1
         }
         else out[i, ] = out[i - 1, ]
     }
     cat("acceptance rate: ", accept/(n - 1), "\n")
     class(out) = c("Metropolis", "matrix")
     out
 }
```

Then, insert the function in your working environment by copying and pasting the whole block.

Let's try and see if it works:

```
> f = function(x) {
     x[1] + x[2] * sin(pi * (hours + x[3])/12)
 }
> temp = meteo$T.outside
> hours = meteo$hours
> out = Metropolis(c(0, 0, 0), rep(0.1, 3), temp, f, n = 5000)
> out[1:10, ]
> out[4990:5000, ]
```

We can try to see what happened to the three parameters if we plot them. For this we will make a little dedicated plot function, called `plot.Metropolis`. It will be automatically called for objects of class `Metropolis`

```
> class(out)
```

Here's the plot function:

```
> plot.Metropolis = function(x) {
     oldpar = par()
     par(mfrow = c(ncol(x), 1), mar = rep(2, 4))
     apply(x, 2, function(x) plot(x, type = "l"))
     par(oldpar)
 }
```

that you should load in the environment. Try it with:

```
> plot(out)
```

Note that now your own `plot.Metropolis` has been called, because `Metropolis` is the class of `out`.

Exercise 6.25 Recall the fitted values for the three parameters, found by Gauss-Newton fitting; write them down and compare them with the values you find now. What is different?

Exercise 6.26 Are the simulated values *on average* right?

6.2 The effect of sigma

Set the plot recording on (activate the plot window, use the menu). Now you can browse previous plots with PgUp and PgDown, when the plot window is active.

Let's first try a very small sigma.

```
> out1 = Metropolis(c(0, 0, 0), rep(0.1, 3), temp, f, n = 1000)
> plot(out1)
> out2 = Metropolis(c(0, 0, 0), rep(0.1, 3), temp, f, n = 2000)
> plot(out2)
> out5 = Metropolis(c(0, 0, 0), rep(0.1, 3), temp, f, n = 5000)
> plot(out5)
> out10 = Metropolis(c(0, 0, 0), rep(0.1, 3), temp, f, n = 10000)
> plot(out10)
```

Exercise 6.27 Does the chain for the three parameters become *stationary*, meaning that it fluctuates for a long time over the same area? **Exercise 6.28**

What does the acceptance rate mean?

Try the same for a larger value for sigma:

```
> out1 = Metropolis(c(0, 0, 0), rep(0.5, 3), temp, f, n = 1000)
> plot(out1)
> out2 = Metropolis(c(0, 0, 0), rep(0.5, 3), temp, f, n = 2000)
> plot(out2)
> out5 = Metropolis(c(0, 0, 0), rep(0.5, 3), temp, f, n = 5000)
> plot(out5)
> out10 = Metropolis(c(0, 0, 0), rep(0.5, 3), temp, f, n = 10000)
> plot(out10)
```

Exercise 6.29 Does the chain for the three parameters become *stationary*, meaning that it fluctuates for a long time over the same area? **Exercise 6.30**

How did the acceptance rate change?

```
> out1 = Metropolis(c(0, 0, 0), rep(1, 3), temp, f, n = 1000)
> plot(out1)
> out2 = Metropolis(c(0, 0, 0), rep(1, 3), temp, f, n = 2000)
> plot(out2)
> out5 = Metropolis(c(0, 0, 0), rep(1, 3), temp, f, n = 5000)
> plot(out5)
> out10 = Metropolis(c(0, 0, 0), rep(1, 3), temp, f, n = 10000)
> plot(out10)
```

Exercise 6.31 Does the chain for the three parameters become *stationary*, meaning that it fluctuates for a long time over the same area? **Exercise 6.32**

How did the acceptance rate change?

```
> out1 = Metropolis(c(0, 0, 0), rep(2, 3), temp, f, n = 1000)
> plot(out1)
> out2 = Metropolis(c(0, 0, 0), rep(2, 3), temp, f, n = 2000)
> plot(out2)
> out5 = Metropolis(c(0, 0, 0), rep(2, 3), temp, f, n = 5000)
> plot(out5)
> out10 = Metropolis(c(0, 0, 0), rep(2, 3), temp, f, n = 10000)
> plot(out10)
```

Exercise 6.33 Does the chain for the three parameters become *stationary*, meaning that it fluctuates for a long time over the same area? **Exercise 6.34**

How did the acceptance rate change?

```
> out1 = Metropolis(c(0, 0, 0), rep(5, 3), temp, f, n = 1000)
> plot(out1)
> out2 = Metropolis(c(0, 0, 0), rep(5, 3), temp, f, n = 2000)
> plot(out2)
> out5 = Metropolis(c(0, 0, 0), rep(5, 3), temp, f, n = 5000)
> plot(out5)
> out10 = Metropolis(c(0, 0, 0), rep(5, 3), temp, f, n = 10000)
> plot(out10)
```

Exercise 6.35 Does the chain for the three parameters become *stationary*, meaning that it fluctuates for a long time over the same area? **Exercise 6.36**

How did the acceptance rate change?

```
> out1 = Metropolis(c(0, 0, 0), rep(10, 3), temp, f, n = 1000)
> plot(out1)
> out2 = Metropolis(c(0, 0, 0), rep(10, 3), temp, f, n = 2000)
> plot(out2)
> out5 = Metropolis(c(0, 0, 0), rep(10, 3), temp, f, n = 5000)
> plot(out5)
> out10 = Metropolis(c(0, 0, 0), rep(10, 3), temp, f, n = 10000)
> plot(out10)
```

Exercise 6.37 Does the chain for the three parameters become *stationary*, meaning that it fluctuates for a long time over the same area? **Exercise 6.38**

How did the acceptance rate change?

6.3 Fixing the phase

It might be the case that the phase fitting disturbs the process – after all, any value of `x[3]`+24 is another equally well fitted value. Suppose we fix it to 1.5:

```
> f2 = function(x) {
     x[1] + x[2] * sin(pi * (hours + 1.5)/12)
 }
> out = Metropolis(c(0, 0), rep(1, 3), temp, f2, n = 5000)
> plot(out)
```

Exercise 6.39 Is the result now more stationary than without the fixed phase?

6.4 Computing summary statistics

A useful function for doing something for each row or column is apply. Read it's help page, and try

```
> apply(out, 2, summary)
```

Exercise 6.40 How do you interpret the result?

6.5 Initial values and Burn-in

Up to now, we have tried zero initial values, as in

```
> out = Metropolis(c(0, 0), rep(0.5, 2), temp, f2, n = 5000)
> plot(out)
```

The burn-in period is the period that the chain needs to go from the initial values to the region where it becomes stationary.

Exercise 6.41 How long (how many steps) is the burn-in period for the above example?

Exercise 6.42 How do you compute the mean and summary statistics for a chain *without* the burn-in values?

Exercise 6.43 How does the burn-in period change when sigma decreases?

7 Spatial modelling: introductory matter

7.1 Data sets

Besides data that are typed in, or data that are imported through import functions such as `read.csv`, some data are already available in packages. In package `gstat` for example, a data set called `meuse` is available. To copy this data set to the current working data base, use, e.g. for the `meuse` data

```
> library(gstat)
> data(meuse)
> objects()
> summary(meuse)
```

7.1.1 meuse

The meuse data set is a data set collected from the Meuse floodplain. The heavy metals analyzed are copper, cadmium, lead and zinc concentrations, extracted from the topsoil of a part of the Meuse floodplain, the area around Meers and Maasband (Limburg, The Netherlands). The data were collected during fieldwork in 1990 (Rikken and Van Rijn, 1993; Burrough and McDonnell, 1998).

Try and read:

```
> library(gstat)
> `?`(meuse)
```

(if this does not work, simply use question mark followed by meuse, as in `?meuse`)

7.2 Models: the formula interface; linear regression

Most statisticians (and many earth scientists as well) like to analyse data through models: models reflect the hypotheses we entertain, and from fitting models, analysing model output, and analysising graphs of fitted values and residuals we can learn from the data whether a given hypothesis was reasonable or not.

Models are often variations of regression models. Here, we will illustrate an example of a simple linear regression model involving the heavy metal pollution data in `meuse`. We will entertain the hypothesis that zinc concentration, present in variable `zinc`, depends linearly on distance to the river, present in `dist`. We can express dependency of `y` on `x` by a formula, coded as `y~x`. Formulas can have multiple right-hand sides for multiple linear regression, as `y~x1+x2`; they also can contain expressions (transformations) of variables, as in `log(y)~sqrt(x)`. See `?formula` for a full description of the full functionality.

To calculate a linear regression model for the meuse data:

```
> library(gstat)
> data(meuse)
> zinc.lm = lm(zinc ~ dist, meuse)
> class(zinc.lm)
> summary(zinc.lm)
```

Do the following yourself:

```
> plot(zinc.lm)
```

Besides the variety of plots you now obtain, there are many further custom options you can set that can help analysing these data. When you ask help by `?plot`, it does not provide very helpful information. To get help on the plot method that is called when plotting an object of class `lm`, remember that the function called is `plot.lm`. Read the help of `plot.lm` by `?plot.lm` or:

```
> `?`(plot.lm)
```

and customize the plot call, e.g. by

```
> plot(zinc.lm, panel = panel.smooth)
```

In this call, the name of the second function argument is added because the argument panel is on position four. An alternative form is:

```
> plot(zinc.lm, , , panel.smooth)
```

which passes `panel.smooth` as argument number 4, without giving its name. The first form is strongly recommended because it is less prone to error, and more readable. Furthermore, argument names are less likely to change over time (living software does change!) than argument order.

Exercise 7.44 Which of the following assumptions underlying linear regression are violated (if any):

- none
- residuals are heteroscedastic (i.e., their variance is not constant over the range of fitted values)

- residuals are not normally distributed (their distribution is for example non-symmetric)
- residuals are heteroscedastic and not normally distributed

7.3 Spatial data in R: the sp package

The R package `sp` provides classes and methods for spatial data; currently it can deal with points, grids, lines and polygons. It further provides interfaces to various GIS functions such as overlay, projection and reading and writing of external GIS formats.

7.3.1 Points

Try:

```
> library(sp)
> data(meuse)
> class(meuse)
> summary(meuse)
> coordinates(meuse) = c("x", "y")
> class(meuse)
> summary(meuse)
> sp.theme(TRUE)
> spplot(meuse, "zinc", key.space = "right")
```

Here, the crucial argument is the `coordinates` function: it specifies for the `meuse` data.frame which variables are spatial coordinates. In effect, it replaces the data.frame meuse which has "just" two columns with coordinates with a structure (of class `SpatialPointsDataFrame`) that *knows* which columns are spatial coordinates. As a consequence, `spplot` knows how to plot the data *in map-form*. If there is any function or data set for which you want help, e.g. `meuse`, read the documentation: `?meuse` , `?spplot` etc.

A shorter form for `coordinates` is by assigning a formula, as in

```
> data(meuse)
> coordinates(meuse) = ~x + y
> coordinates(meuse)[1:10, ]
> spplot(meuse, "zinc", key.space = "right")
```

The function `coordinates` *without assignment* retrieves the matrix with coordinate values.

The plot now obtained shows points, but without reference where on earth they lie. You could e.g. add a reference by showing the coordinate system:

```
> spplot(meuse, "zinc", key.space = "right", scales = list(draw = TRUE))
```

but this tells little about *what* we see. As another option, you can add geographic reference by e.g. lines of rivers, or administrative boundaries. In our example, we prepared the coordinates of (part of) the river Meuse river boundaries, and will add them as reference:

```
> data(meuse.riv)
> dim(meuse.riv)
> meuse.riv[1:10, ]
> meuse.sp = SpatialPolygons(list(Polygons(list(Polygon(meuse.riv)),
     "meuse.riv")))
> meuse.lt = list("sp.polygons", meuse.sp, fill = "grey")
> spplot(meuse, "zinc", key.space = "right", sp.layout = meuse.lt)
```

`SpatialRings`, `Srings` and `Sring` create an `sp` polygon object from a simple matrix of coordinates; the `sp.layout` argument contains map elements to be added to an `spplot` plot.

Note that the points in the plot partly fall in the river; this may be attributed to one or more of the following reasons: (i) the river coordinates are not correct, (ii) the point coordinates are not correct, (iii) points were taken on the river bank when the water was low, whereas the river boundary indicates the high water extent of the river.

7.3.2 Grids

Try:

```
> data(meuse.grid)
> class(meuse.grid)
> coordinates(meuse.grid) = c("x", "y")
> class(meuse.grid)
> gridded(meuse.grid) = TRUE
> class(meuse.grid)
> summary(meuse.grid)
> meuse.lt = list(riv = list("sp.polygons", meuse.sp, fill = "grey"),
     pts = list("sp.points", meuse, pch = 3, cex = 0.5, col = "black"))
> spplot(meuse.grid, sp.layout = meuse.lt, key.space = "right")
```

Here, `gridded(meuse.grid) = TRUE` promotes the (gridded!) points to a structure which knows that the points are on a regular grid. As a consequence they are drawn with filled rectangular symbols, instead of filled circles. (Try the same sequence of commands without the call to `gridded()` if you are curious what happens if `meuse.grid` were left as a set of points).

For explanation on the `sp.layout` argument, read `?spplot`; much of the codes in them (`pch`, `cex`, `col`) are documented in `?par`.

Note that `spplot` plots points on top of the grid, and the grid cells on top of the polygon with the river. (When printing to pdf, transparency options can be used to combine both.)

8 Geostatistics

These practical exercises cover the use of geostatistical applications in environmental research. They give an introduction to exploratory data analysis, sample variogram calculation and variogram modelling and several spatial interpolation techniques. Interpolation methods described and applied are inverse distance interpolation, trend surface fitting, thiessen polygons, ordinary (block) kriging, stratified (block) kriging and indicator (block) kriging.

The gstat R package is used for all geostatistical calculations. See:

E.J. Pebesma, 2004. Multivariable geostatistics in S: the gstat package. Computers & Geosciences 30: 683-691 .

8.1 Exploratory data analysis

Identify the five largest zinc concentrations by clicking them on the plot:

```
> library(sp)
> sp.theme(TRUE)
> data(meuse)
> coordinates(meuse) = c("x", "y")
> sel = spplot(meuse, "zinc", identify = TRUE)
> sel
```

Exercise 8.45 Which points have the largest zinc concentration:

a 44 51 54 55 82

b 49 53 54 55 82

c 46 53 54 55 82

d 53 54 55 59 82

8.2 Simple interpolation algorithms

8.2.1 Trend surface analysis

Trend surface interpolation is multiple linear regression with polynomials of coordinates as independent, predictor variables. Although we could use `lm` to fit these models and predict them, it would require scaling of coordinates prior to prediction in most cases because higher powers of large coordinate values result in large numerical errors. The `krige` function in library `gstat` scales coordinates and allows trend surface up to degree 3.

```
> library(gstat)
> data(meuse.grid)
> coordinates(meuse.grid) = ~x + y
> gridded(meuse.grid) = TRUE
> meuse.grid$tr1 = krige(log(zinc) ~ 1, meuse, meuse.grid, degree = 1)$var1.pred
> meuse.grid$tr2 = krige(log(zinc) ~ 1, meuse, meuse.grid, degree = 2)$var1.pred
> meuse.grid$tr3 = krige(log(zinc) ~ 1, meuse, meuse.grid, degree = 3)$var1.pred
> spplot(meuse.grid, c("tr1", "tr2", "tr3"), sp.layout = meuse.lt,
     main = "log(zinc), trend surface interpolation")
```

Function `surf.ls` from library `spatial` (described in the MASS book) fits trends up to order 6 with scaled coordinates.

As you can see, `?krige` does not provide help on argument `degree`. However, tells that any remaining arguments (...) are passed to function `gstat`, so look for argument `degree` on `?gstat`.

Exercise 8.46 The danger of extreme extrapolation values is likely to occur

a for low trend orders

b for high trend orders

c for none of these models

Local trends, i.e. trends in a local neighbourhood may also be fitted:

```
> m = krige(log(zinc) ~ x + y, meuse, meuse.grid, nmax = 10)
> spplot(m, "var1.pred", sp.layout = meuse.lt, main = "local 1st order trend")
```

8.2.2 Inverse distance interpolation

Inverse distance interpolation calculates a weighted average of points in the (by default global) neighbourhood, using weights inverse proportional to the distance of data locations to the prediction location raised to the power p. This power is by default 2:

```
> library(gstat)
> lzn.tp = idw(log(zinc) ~ 1, meuse, meuse.grid)
> spplot(lzn.tp, "var1.pred", sp.layout = meuse.lt, main = "log(zinc), inverse distance in

> meuse.grid$idp0.5 = idw(log(zinc) ~ 1, meuse, meuse.grid, idp = 0.5)$var1.pred
> meuse.grid$idp02 = idw(log(zinc) ~ 1, meuse, meuse.grid, idp = 2)$var1.pred
> meuse.grid$idp05 = idw(log(zinc) ~ 1, meuse, meuse.grid, idp = 5)$var1.pred
> meuse.grid$idp10 = idw(log(zinc) ~ 1, meuse, meuse.grid, idp = 10)$var1.pred
> spplot(meuse.grid, c("idp0.5", "idp02", "idp05", "idp10"), sp.layout = meuse.lt,
     main = "log(zinc), inverse distance interpolation")
```

The argument `set` is a list because other parameters can be passed to gstat using this list as well.

Exercise 8.47 With a larger inverse distance power (`idp`), the weight of the nearest data point

a becomes larger

b becomes smaller

c does not change

8.2.3 Thiessen polygons

Thiessen polygons can be constructed (as vector objects) using the functions in library `tripack`. Interpolation on a regular grid is simply obtained by using e.g. inverse distance interpolation and a neighbourhood size of one (`nmax = 1`). Combining the two:

```
> library(gstat)
> library(sp)
> lzn.tp = krige(log(zinc) ~ 1, meuse, meuse.grid, nmax = 1)
> image(lzn.tp["var1.pred"])
> points(meuse, pch = "+", cex = 0.5)
> cc = coordinates(meuse)
> library(tripack)
> plot(voronoi.mosaic(cc[, 1], cc[, 2]), do.points = FALSE, add = TRUE)
> title("Thiessen (or Voronoi) polygon interpolation of log(zinc)")
```

8.3 Spatial prediction with multiple linear regression

In the data set `meuse` we have a large number of variables present that can be used to form predictive regression models for either of the heavy metal variables. If we want to use such a regression model for spatial prediction, we need the variables as a full spatial coverage, e.g. in the form of a regular grid covering the study area, as well. Only few variables are available for this in `meuse.grid`:

```
> library(sp)
> sp.theme(TRUE)
> data(meuse)
> names(meuse)
> data(meuse.grid)
> names(meuse.grid)
```

Now try:

```
> coordinates(meuse) = c("x", "y")
> coordinates(meuse.grid) = c("x", "y")
> gridded(meuse.grid) = TRUE
> lzn.lm = lm(log(zinc) ~ sqrt(dist), meuse)
> summary(lzn.lm)
> plot(lzn.lm)
> plot(log(zinc) ~ sqrt(dist), meuse)
> abline(lzn.lm)
> lzn.pr = predict(lzn.lm, meuse.grid, se.fit = TRUE)
> meuse.grid$lzn.fit = lzn.pr$fit
> spplot(meuse.grid, "lzn.fit", sp.layout = meuse.lt)
> meuse.grid$se.fit = lzn.pr$se.fit
> spplot(meuse.grid, "se.fit", sp.layout = meuse.lt)
```

Exercise 8.48 Why are prediction standard errors largest for small and large distance to the river?

a Because variability is smallest for intermediate distance to river

b because the uncertainty in the regression *slope* is most prevalent when the regressor distance takes extreme values

c because we have too few degrees of freedom for regression (i.e., too few observations) overall

d because we have too little data close to the river and far away from the river

We can use the prediction errors provided by `lm` to calculate regression prediction intervals:

```
> plot(log(zinc) ~ sqrt(dist), meuse)
> x = 0:100/100
> lzn.lm = lm(log(zinc) ~ sqrt(dist), meuse)
> pr = predict(lzn.lm, data.frame(dist = x), interval = "prediction")
> abline(lzn.lm)
> lines(sqrt(x), pr[, 2], lty = "dashed", col = "red")
> lines(sqrt(x), pr[, 3], lty = "dashed", col = "red")
> pr = predict(lzn.lm, data.frame(dist = x), interval = "confidence")
> lines(sqrt(x), pr[, 2], lty = "dashed", col = "blue")
> lines(sqrt(x), pr[, 3], lty = "dashed", col = "blue")
> title("95% confidence intervals (blue) and prediction intervals (red)")
```

Next, we can make maps of the 95% prediction intervals:

```
> pr = predict(lzn.lm, as.data.frame(meuse.grid), interval = "prediction")
> meuse.grid$fit = pr[, 1]
> meuse.grid$upper = pr[, 3]
> meuse.grid$lower = pr[, 2]
> spplot(meuse.grid, c("fit", "lower", "upper"), sp.layout = meuse.lt,
    layout = c(3, 1))
```

Note that all these prediction intervals refer to prediction of single observations; confidence intervals for mean values (i.e., the regression line alone) are obtained by specifying `interval = "confidence"`.

Exercise 8.49 Compared to prediction intervals, the intervals obtained by `interval = "confidence"` are

a wider

b narrower

c of approximately equal width

Another regression model is obtained by relating zinc to flood frequency, try

```
> plot(log(zinc) ~ ffreq, meuse)
```

Exercise 8.50 Why is a box-and whisker plot drawn for this plot, and not a scatter plot?

- `ffreq` is a categorical variable with numeric levels
- `ffreq` has only three values, 1, 2 and 3
- flood frequency is allways expressed by boxes

Exercise 8.51 What do the boxes in a box-and whisker plot refer to?

- the mean plus and minus one standard deviation of the group
- the 95% confidence interval for the mean
- the inter-quartile range (from 25-th to 75-th percentiles)

Exercise 8.52 What is needed for flood frequency to use it for mapping zinc?

- it needs to be the number of times a grid cell floods
- it needs to be known at each prediction location (i.e., grid cells in `meuse.grid`
- it needs to be independent of distance to the river

Try the model in prediction mode:

```
> lzn.lm2 = lm(log(zinc) ~ ffreq, meuse)
> summary(lzn.lm2)
> pr = predict(lzn.lm2, meuse.grid, interval = "prediction")
> meuse.grid$fit = pr[, 1]
> meuse.grid$upper = pr[, 3]
> meuse.grid$lower = pr[, 2]
> spplot(meuse.grid, c("fit", "lower", "upper"), sp.layout = meuse.lt,
    layout = c(3, 1))
```

Exercise 8.53 Why is the pattern obtained not smooth?

8.4 Spatial correlation: the h-scatterplot

Enter the following R function in your session (don't feel frustrated if you do not understand what is going on):

```
> hscat = function(formula, data, breaks) {
    require(lattice)
    require(gstat)
    stopifnot(!missing(breaks))
    x = variogram(formula, data, cloud = T)
    .BigInt = attr(x, ".BigInt")
    x$left = x$np%%.BigInt + 1
    x$right = x$np%/%.BigInt + 1
    x$class = cut(x$dist, breaks = breaks)
    y = model.frame(formula, data)[[1]]
    x$xx = y[x$left]
    x$yy = y[x$right]
    lab = as.character(formula)[2]
```

```
    panel = function(x, y, subscripts, ...) {
        xr = c(min(x), max(x))
        llines(xr, xr)
        lpoints(x, y, ...)
        ltext(min(x), max(y), paste("r =", signif(cor(x, y),
            3)), adj = c(0, 0.5))
    }
    xyplot(xx ~ yy | class, x, panel = panel, main = "lagged scatterplots",
        xlab = lab, ylab = lab)
}
```

We will use this function to compute h-scatterplots. An h-scatterplot plots data pairs $z(x_i), z(x_j)$ for selected pairs that have $|x_i - x_j|$ fall within a given interval. The intervals are given by the third argument, (`breaks`).

```
> hscat(log(zinc) ~ 1, meuse, c(0, 30, 80, 100, 200, 300, 500,
    1000))
```

Exercise 8.54 Why is there not a distance interval $0 - 30$ in the plot?

Exercise 8.55 One would expect the correlation to decrease for larger distances. This does not happen. Why?

```
> hscat(log(zinc) ~ 1, meuse, c(0, 80, 120, 250, 500, 1200))
```

Exercise 8.56 The correlation now steadily drops with larger distance. Why?

```
> hscat(log(zinc) ~ 1, meuse, c(0, 120, 300, 600, 1200))
```

Exercise 8.57 The correlation for the largest distance interval is negative. How is this possible?

8.5 Spatial correlation: the variogram cloud

As you could see, above the variogram cloud was used to get h-scatterplots. The variogram cloud is basically the plot (or collection of plotted points) of

$$0.5(Z(x_i) - Z(x_j))^2$$

against

$$h = |x_i - x_j|$$

in words: for each point pair $z(x_i), z(x_j)$ half the squared difference in measured values against the spatial separation distance.

Plot a variogram cloud for the log-zinc data:

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

Exercise 8.58 Why are there so many points?

Exercise 8.59 Why are both distance and semivariance non-negative?

Exercise 8.60 Why is there so much scatter in this plot?

Exercise 8.61 How many points does the variogram cloud contain if the separation distance is not limited?

Identify a couple of points in the cloud, and plot the pairs in a map by repeating the following commands a couple of times.

```
> out = plot(variogram(log(zinc) ~ 1, meuse, cloud = TRUE), digitize = TRUE)
> plot(out, meuse)
```

Exercise 8.62 Where are the short-distance-high-variability point pairs, as opposed to the short-distance-small-variability point pairs?

8.6 Spatial correlation: the variogram

A variogram for the log-zinc data is computed, printed and plotted by

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

Exercise 8.63 How is the variogram computed from the variogram cloud?

- semivariance is averaged for each distance
- distance is averaged for each semivariance
- distance is averaged for each semivariance class
- semivariance and distance are averaged for each distance class
- semivariance and distance are averaged for each semivariance class

8.7 Sample variogram and variogram model

To detect, or model spatial correlation, we average the variogram cloud values over distance intervals (bins):

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

For the maximum distance (cutoff) and bin width, sensible defaults are chosen as one-third the largest area diagonal and 15 bins, but we can modify this when needed by setting `cutoff` and `width`:

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

Exercise 8.64 What is the problem if we set width to e.g. a value of 5?

a we get too many semivariance estimates

b the semivariance values are too large

c the semivariance values are too small

d the variability of the semivariance values is too large

Exercise 8.65 What is the problem if we set the cutoff value to e.g. a value of 500 and width to 50?

a we cannot see up to which distance the data are spatially correlated

b the semivariance values are too large

c the semivariance values are too small

d we cannot see if data are spatially correlated

For kriging, we need a suitable, statistically valid model fitted to the sample variogram. The reason for this is that kriging requires the specification of semivariances for *any* distance value in the spatial domain we work with. Simply connecting sample variogram points, as in

```
> lzn.vgm = variogram(log(zinc) ~ 1, meuse, cutoff = 1000, width = 50)
> plot(gamma ~ dist, lzn.vgm)
> lines(c(0, lzn.vgm$dist), c(0, lzn.vgm$gamma))
```

will *not* result in a valid model. Instead, we will fit a valid parametric model. Some of the valid parametric models are shown by e.g.

```
> show.vgms()
```

These models may be combined, and the most commonly used are Exponential or Spherical models combined by a Nugget model.

We will try a spherical model:

```
> lzn.vgm = variogram(log(zinc) ~ 1, meuse)
> plot(lzn.vgm)
> lzn.mod = vgm(0.6, "Sph", 1000, 0.05)
> plot(lzn.vgm, lzn.mod)
```

Exercise 8.66 The range, 1000, relates to

a the semivariance value at distance (nearly) zero

b the semivariance value reached when the variogram stops increasing

c the distance value at which the variogram stops increasing

Exercise 8.67 The sill, 0.6 + 0.05, relates to

a the semivariance value at distance (nearly) zero

b the semivariance value reached when the variogram stops increasing

c the distance value at which the variogram stops increasing

Exercise 8.68 The nugget, 0.05, relates to

a the semivariance value at distance (nearly) zero

b the semivariance value reached when the variogram stops increasing

c the distance value at which the variogram stops increasing

The variogram model (`"Sph"` for spherical) and the three parameters were chosen by eye. We can fit the three parameters also automatically:

```
> lzn.vgm = variogram(log(zinc) ~ 1, meuse)
> lzn.mod = fit.variogram(lzn.vgm, vgm(0.6, "Sph", 1000, 0.05))
> lzn.mod
> plot(lzn.vgm, lzn.mod)
```

but we need to have good "starting" values, e.g. the following will not yield a satisfactory fit:

```
> lzn.misfit = fit.variogram(lzn.vgm, vgm(0.6, "Sph", 10, 0.05))
> plot(lzn.vgm, lzn.misfit)
```

8.8 Simple and ordinary kriging

Ordinary kriging is an interpolation method that uses weighted averages of all, or a defined set of neighbouring, observations. It calculates a weighted average, where weights are chosen such that (i) prediction error variance is minimal, and (ii) predictions are unbiased. The second requirement results in weights that sum to one. Simple kriging assumens the mean (or mean function) to be known; consequently it drops the second requirement and the weights do not have to sum to unity. Kriging assumes the variogram to be known; based on the variogram (or the covariogram, derived from the variogram) it calculates prediction variances.

Given that `lzn.mod` a suitable variogram model contains, we can apply ordinary kriging by

```
> lzn.ok = krige(log(zinc) ~ 1, meuse, meuse.grid, lzn.mod)
> spplot(lzn.ok, "var1.pred", main = "log(zinc), ordinary kriging")
```

For simple kriging, we have to specify in addition the known mean value, passed as argument `beta`:

```
> lzn = krige(log(zinc) ~ 1, meuse, meuse.grid, lzn.mod, beta = 5.9)
> lzn$sk = lzn$var1.pred
> lzn$ok = lzn.ok$var1.pred
> spplot(lzn, c("ok", "sk"), main = "log(zinc), ordinary and simple kriging")
```

Comparing the kriging standard errors:

```
> lzn$sk.se = sqrt(lzn$var1.var)
> lzn$ok.se = sqrt(lzn.ok$var1.var)
> meuse.lt$pts$col = "green"
> spplot(lzn, c("ok.se", "sk.se"), sp.layout = meuse.lt, main = "log(zinc), ordinary and s
```

Exercise 8.69 Why is the kriging standard error not zero on grid cells that contain data points?

a kriging standard errors are not zero on data points

b the data points do not coincide exactly with grid cell centres

c kriging standard errors do not depend on data locations, they only depend on the variability of measured values

Local kriging is obtained when, instead of all data points, only a subset of points nearest to the prediction location are selected. If we use only the nearest 5 points:

```
> lzn.lok = krige(log(zinc) ~ 1, meuse, meuse.grid, lzn.mod, nmax = 5)
> lzn$lok = lzn.lok$var1.pred
> spplot(lzn, c("lok", "ok"), main = "lok: local (n=5); ok: global")
```

Exercise 8.70 When we increase the value of `nmax`, the differences between the local and global kriging will

a become smaller

b become larger

c remain the same

Another, commonly used strategy is to select a kriging neighbourhood based on spatial distance to prediction location. To exaggerate the effect of distance-based neighbourhoods, a very small neighbourhood is chosen of 200 m:

```
> lzn.lok2 = krige(log(zinc) ~ 1, meuse, meuse.grid, lzn.mod, maxdist = 200)
> lzn.ok$lok2 = lzn.lok2$var1.pred
> lzn.ok$ok = lzn.ok$var1.pred
> spplot(lzn.ok, c("lok2", "ok"), main = "lok: local (maxdist=200); ok: global",
     sp.layout = meuse.lt, scales = list(draw = TRUE))
```

Exercise 8.71 How can the gaps that now occur in the local neighbourhood kriging map be explained? At these prediction locations

a the kriging neighbourhood is empty

b the kriging neighbourhood only contains a single observation

c the kriging neighbourhood contains less than 2 observations

```
> lzn.sok2 = krige(log(zinc) ~ 1, meuse, meuse.grid, lzn.mod, maxdist = 200,
     beta = 5.9)
> lzn.ok$sok2 = lzn.sok2$var1.pred
> spplot(lzn.ok, c("lok2", "sok2"), main = "local kriging (maxdist=200); left: ordinary, r
     sp.layout = meuse.lt, scales = list(draw = TRUE))
```

Ordinary and simple kriging behave different in this latter aspect.

Exercise 8.72 In the areas with gaps in the ordinary kriging map, simple kriging yields

a the simple kriging mean value (5.9)

b a compromise between the mean value (5.9) and the nearest observation

c a compromise between the mean value (5.9) and the nearest two or more observations

Some texts advocate to choose search neighbourhood distances equal to the range of the variogram, however there is no theoretical nor a practical justification for this: weights of points beyond the correlation range are usually not zero. Choosing a local neighbourhood may save time and memory when kriging large data sets. In addition, the assumption of a global constant mean is weakened to that of a constant mean within the search neighbourhood. Examine the run time for the following problem: (interpolation of approximately 1000 points with random values).

Randomly select approximately 1000 points:

```
> n.request = 1000
> pts = spsample(meuse.grid, n = n.request, type = "random")
```

See how many we obtained:

```
> n.obtained = dim(coordinates(pts))[1]
> n.obtained
```

Create nonsense data from a standard normal distribution; interpolate and plot them:

```
> dummy = SpatialPointsDataFrame(pts, data.frame(z = rnorm(n.obtained)))
> system.time(dummy.ok <- krige(z ~ 1, dummy, meuse.grid, vgm(1,
     "Exp", 300)))
> system.time(dummy.lok <- krige(z ~ 1, dummy, meuse.grid, vgm(1,
```

```
    "Exp", 300), nmax = 20))
> dummy.ok$var1.local = dummy.lok$var1.pred
> spplot(dummy.ok, c("var1.pred", "var1.local"), sp.layout = list("sp.points",
    dummy, col = "black", cex = 0.2))
```

and compare the run time for global kriging to local kriging with the nearest 20 observations.

Exercise 8.73 The gain in speed is approximately a factor

a 5

b 10

c 20

d 60

Although not needed here, for exact timings, you can use `system.time`, but then replace `dummy.int = ...` with `dummy.int <- ...` in the expression argument.

8.9 Universal kriging

Just like stratified kriging, universal kriging provides a less restrictive model than ordinary kriging. Suppose the model for ordinary kriging is written as

$$Z(s) = m + e(s)$$

with $Z(s)$ the observed variable Z at spatial location s, m a constant but unknown mean value and $e(s)$ an intrinsically stationary residual, we can write stratified kriging as j models

$$Z_j(s) = m_j + e_j(s),$$

j referring to the stratum: each stratum has a different mean and different parameters describing the spatial correlation of e_j, and $e_j(s)$ and $e_k(s)$ are independent if $j \neq k$.

Universal kriging extends the ordinary kriging model by replacing the constant mean value by a non-constant mean *function*:

$$Z(s) = \beta_0 + \beta_1 X_1(s) + ... + \beta_p X_p(s) + e(s)$$

with β_j the trend (i.e., regression) coefficients, and $X_j(s)$ p *known* predictor functions (independent variables, regressors).

The hope is that the predictor functions carry information in them that explains the variability of $Z(s)$ to a considerable extent, in which case the residual $e(s)$ will have lower variance and less spatial correlation compared to the ordinary kriging case. To see that this model *extends* ordinary kriging, take $p = 0$ and note that for that case $\beta_0 = m$. Another simplification is that if $e(s)$ is spatially independent, the universal kriging model reduces to a multiple linear regression model. One problem with universal kriging is that we need the variogram of the residuals, without having measured the residuals.

As we've seen before, we can predict the zinc concentrations fairly well from the sqrt-transformed values of distance to river:

```
> plot(log(zinc) ~ sqrt(dist), as.data.frame(meuse))
> lzn.lm = lm(log(zinc) ~ sqrt(dist), as.data.frame(meuse))
> summary(lzn.lm)
> abline(lzn.lm)
```

To compare the variograms from raw (but log-transformed) data and regression residuals, look at the conditioning plot obtained by

```
> g = gstat(id = "raw data", formula = log(zinc) ~ 1, data = meuse)
> g = gstat(g, id = "residuals", formula = log(zinc) ~ sqrt(dist),
    data = meuse)
> plot(variogram(g, cross = F), scales = list(relation = "same"),
    layout = c(2, 1))
```

Exercise 8.74 How is the residual standard error from the regression model above (`lzn.lm`) related to the variogram of the residuals:

a it is half the mean semivariance of the raw data and that of the residuals

b the square of the residual standard error (the residual variance) equals the sill of the residual variogram

c it equals the range of the residual variogram

d it equals the sill of the residual variogram

d it equals the nugget of the residual variogram

e the residual variance (approximately) equals the nugget of the variogram

Exercise 8.75 Why is the residual variogram much lower than that of raw data?

a residuals have less spatial correlation than original data

b residuals vary less because here the regression model removed part of the total variability

c the residuals have little in common with the (log) zinc data

d residuals allways have much smaller variability than raw data

When a regression model formula is passed to `variogram`, the variogram of the residuals is calculated. When a model formula is passed to `krige`, universal kriging is used.

Now model the residual variogram with an exponential model, say `zn.res.m`, and apply universal kriging. Compare the maps with predictions from universal kriging and ordinary kriging in a single plot (with a single scale), using `spplot`.

Exercise 8.76 Where are the differences most prevalent:

a In areas with values for `dist` close to 0 and 1

b In areas with few data points

c In areas where both (a) and (b) are valid

Exercise 8.77 Why are the differences happening exactly there?

a on locations far from measurements, interpolation relies heavily on the estimated trend value; where dist is close to 0 or 1, the trend value is most different from the value m used in ordinary kriging

b on locations far from measurements we are most uncertain about the true values

c on locations where dist is close to 0 or 1, it was hard to collect data, because these locations were hard to reach.

Now print the maps with prediction errors for ordinary kriging and universal kriging with a single scale. See if you can answer the previous two questions in the light of the differences between prediction errors.

9 Diffusion and partial differential equations

Consider the following function, that solves a 1-D diffusion equation, using backward differencing, and plots the result. Do the following exercises after you studied the material (slides/podcast) on differential equations. The following

```
> diffuse = function(x.end = 1000, release = 1000, n.end = 1000,
     dx = 1, dt = 0.1, plot.start = 10, cols = rainbow(n.end),
     dirichlet = TRUE, ylim = c(0, 100)) {
     u = rep(0, x.end)
     u[x.end/2] = release
     new = u
     plot.new()
     start = TRUE
     for (n in 1:(n.end/dt)) {
         if (dirichlet)
             new[c(1, x.end)] = 0
         else {
             new[1] = u[2]
             new[x.end] = u[x.end - 1]
         }
         new[2:(x.end - 1)] = dt * (u[1:(x.end - 2)] - 2 * u[2:(x.end -
             1)] + u[3:x.end])/(dx * dx) + u[2:(x.end - 1)]
         if (n * dt > plot.start) {
             if (start)
                 plot(new, type = "l", col = cols[n * dt], xlab = "x",
                   ylab = "u", ylim = ylim)
```

```
            else lines(new, col = cols[n * dt])
            start = FALSE
        }
        u = new
    }
    new
}
```

Try the following:

```
> out = diffuse()
> out = diffuse(dt = 0.25)
> out = diffuse(dt = 0.49)
> out = diffuse(dt = 0.5)
> out = diffuse(dt = 0.51)
> out = diffuse(dt = 0.05)
> out = diffuse(dt = 0.49, n.end = 10000)
```

Exercise 9.78 What happens when `dt` becomes 0.5? Explain.

Exercise 9.79 What happens when `dt` becomes larger then 0.5? Explain.

Exercise 9.80 What happens when `dt` becomes larger then 0.5?

Exercise 9.81 What is a solution to this problem?

Try

```
> out = diffuse(dt = 0.4, x.end = 100, n.end = 10000, dirichlet = TRUE)
> out = diffuse(dt = 0.4, x.end = 100, n.end = 10000, dirichlet = FALSE)
```

Exercise 9.82 What happens to the material released, at the last time step?

Exercise 9.83 What is the total (sum) of out, after the last time step, for the last two commands?

Exercise 9.84 What is meant with a Dirichlet boundary condition?