A note on R:
Linear models and models with random effects

Torben Martinussen, Ib Skovgaard and Helle Sørensen
Department of Natural Sciences, Faculty of Life Sciences
January 2007

R is a statistical computer program. It has several advantages: it is well suited for programming and statistical computing, it easily produces high level plots, and it is freely available. You can read more about the program on the Rhome page `www.r-project.org`. This note gives some instructions on how to apply the program for doing statistical analyses corresponding to the topics covered in the course *Statistisk Dataanalyse 2* (SD2) offered at Faculty of Life Sciences (University of Copenhagen). Most of the examples given in this note are closely linked to the examples in Bibby et al. [2006] which is used as text material in SD2. Chapters 1–3 are copied from Martinussen and Skovgaard [2006] which was used in another course, *Statistisk Dataanalyse 1*. Moreover, there is a considerable overlap in topics between Chapter 5 from Martinussen and Skovgaard [2006] and Sections 4.1 of the present note.

Contents

1

Starting and working with R

We assume that R has already been installed, see Appendix A for details. You then start R by double-clicking on the R-icon (a capital R) on the desktop. The R-console window, see Fig. 1.1, then appears and you meet the prompt

```
>
```

which means that R is ready to receive a command. For example, you write `3+2`

```
> 3+2
[1] 5
```

and R responds with the second line when you press Enter.

Alternatively, and this is generally recommended, you can write your commands in a so-called R-script (or R-program) which is just a separate file and then run the commands from the script: choose `File -> New script` in the menu bar in order to open a new script, write commands in the script and run a command line by `Ctrl+R` (or click `Edit -> Run line or selection`) The command is then automatically transferred to the R-console and run, and the command as well as the result appear as before. Instead of the `Ctrl+R` command you may also use copy-paste from the R-script to the R-console.

Save the R-script when you finish the R-session (`Ctrl+S` or via the File menu), then you have the commands for later use. You may in principle use any text-editor (notepad for example but not Word as it does not produce plain text-files) to write your commands to R and then copy-paste the text into the R-console. It is most convenient to save all files (data files and R-programs) regarding a certain project (or course) in the same directory, and ask R to use this directory as working directory. This is done via the File menu.

R as a pocket calculator

R may be used as a simple pocket calculator. All standard functions (powers, exponential, square-root, logarithm, *etc.*) are of course built-in. For example,

Figure 1.1: The R console window.

```
> (17-3) / (3-1) + 1
[1] 8
> 3*4 - 7 + 1.5
[1] 6.5
> 1.8^2 * exp(0.5) - sqrt(2.4)
[1] 3.792664
```

The last example shows that $1.8^2 * e^{0.5} - \sqrt{2.4}$ equals 3.79.

Variables. You can also assign values to variables and use them for later computations. For example,

```
> x = 1.8^2
> y = exp(0.5)
> z = x*y - sqrt(2.4)
> z
[1] 3.792664
```

Note that when you just write `z`, `R` prints its value. Variable names may contain periods as in `fit.data1`. `R` is case sensitive, which means that `y` and `Y` will be different.

Vectors and matrices. One can also construct vectors in `R`. The `c(...)` is used to define the vector that is written within the `c(...)` construct:

```
> x=c(2,5,6,9,17)
> x
[1]  2  5  6  9 17
```

Some other convenient commands are

```
> y=1:5
> z=seq(1,15,3)
> w=rep(2,6)
> u=rep(c(1,2,3), each=4)
> y
[1] 1 2 3 4 5
> z
[1]  1  4  7 10 13
> w
[1] 2 2 2 2 2 2
> u
[1] 1 1 1 1 2 2 2 2 3 3 3 3
```

where `seq(1,15,3)` gives the natural numbers from 1 to 15 with an inter-distance of 3. Since the inter-distance between 13 and 15 is only 2, 15 is not included in the sequence. One can also do vectorized arithmetic and even apply functions to vectors as for example

```
> x/y
[1] 2.00 2.50 2.00 2.25 3.40
> sqrt(y)
[1] 1.000000 1.414214 1.732051 2.000000 2.236068
```

where the operations are carried out element-wise, that is, the first value of `x/y` is $2/1$ and so forth; similarly the square-root-function is applied to each value of the `y`-vector.

Matrices can be constructed using for example the function `matrix` as shown below

```
> A=matrix(1:6,2,3)
> B=matrix(1:6,3,2)
> A
     [,1] [,2] [,3]   # Matrix with 2 rows and 3 columns.
[1,]    1    3    5
[2,]    2    4    6
> B
     [,1] [,2]        # Matrix with 3 rows and 2 columns.
[1,]    1    4
[2,]    2    5
[3,]    3    6
> A%*%B               # Do the matrix multiplication AB
     [,1] [,2]
[1,]   22   49
[2,]   28   64
```

Note that what is written on a line after a `#` in `R` is ignored and may thus be used to write comments. We see from the above matrix constructions using the function `matrix` that the columns of the matrix are filled up first. You may use the `byrow=T` as in

```
> A=matrix(1:6,2,3,byrow=T)  # T is short for TRUE
> A
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

to change this to the rows. It is easy in `R` to do matrix calculations such as calculating the determinant and the inverse of matrix. Let us use the matrix `B` as example and calculate first B^TB

```
> Z=t(B)%*%B
> Z
     [,1] [,2]
[1,]   14   32
[2,]   32   77
```

and then the determinant and inverse of B^TB:

```
> det(Z)
[1] 54
> solve(Z)
           [,1]       [,2]
[1,]  1.4259259 -0.5925926
[2,] -0.5925926  0.2592593
```

2

Getting data into R

Suppose we want to use R to work with the data given in Tables 2.1 and 2.3 in Skovgaard et al. [1999] on milk yield from two groups of cows. The first group consists of 32 cows with observed milk yields $4132, 3672, \ldots, 5161$. The other group consists of 33 cows with observed milk yields $3860, 4130, \ldots, 6290, 5474$.

There are several ways of inputting data, we shall describe some of them here. For small datasets you may simply type the data directly into R as

```
> yield=c(4132,3672,3664, (More data here),6290,5474)
> group=c(1,1,1,(More data here),2,2)
```

In the definition of `group` the '1' is repeated 32 times and the '2' is repeated 33 times.

The above method will rarely be recommendable as (real) datasets often are more extensive and usually will be stored in a text-file or as an excel-file. Suppose the milk yield data are stored in the text-file `Milkyield.txt` as

```
Group Yield
    1  4132
    1  3672
    1  3664
    1  4292
    .
    .    (More data-lines here)
    .
    2  5034
    2  6290
    2  5474
```

then you may input the data using the `read.table` command as

```
> data=read.table("Milkyield.txt",header=TRUE)
> data
   Group Yield
1      1  4132
```

```
2      1  3672
3      1  3664
4      1  4292
       .
       .    (More data-lines here)
       .
63     2  5034
64     2  6290
65     2  5474
```

where `header=TRUE` is used if the first line in the text-file contains the names of the variables (as is the case for our dataset here). If this is not the case then it should be left out. If the dataset contains decimal numbers using commas as `4,2`, the above command needs to be modified to

```
> data=read.table("Milkyield.txt",header=TRUE,dec=',')
```

If decimal numbers are given as `4.2` then you don't have to use the `dec`-option.

If the text-file is not in the working directory but in another directory, a convenient way to locate it is to write

```
data=read.table(file.choose(),header=TRUE)
```

which gives you a window from where you can browse through the directories in a usual way.

If you have the information as in the text-file `Milkyield.txt` stored as an excel sheet then you may input the data to `R` by marking the data area of the excel sheet (including the first line with the variable names) and then choose `Ctrl+C` (or `Edit -> Copy`). Thereafter go to the `R Console` and issue

```
data=read.table('clipboard',header=TRUE)
```

where `header=TRUE` is used as above when reading from a text-file and similarly if the dataset contains decimal numbers.

A table of data read by `read.table` is called a data frame (a data frame may also be created in other ways). You can get various information about a data frame using the functions `dim()` and `names()`

```
> data=read.table("Milkyield.txt",header=TRUE)
> dim(data)
[1] 65  2
> names(data)
[1] "Group" "Yield"
```

giving the number of rows (experimental units) and columns (variables), and the names associated with the columns, respectively.

For other ways of inputting data into `R`, see for example Larsen and Sestoft [2005] or Dalgaard [2000].

3

Manipulating data and some simple statistics

Sometimes we may wish to do some data manipulations before doing a statistical analysis. With the milk yield data in mind, we may for example construct the two datasets consisting of group 1 and 2 only, as shown below where we also add the log-transformed yield data for group 1.

```
> data1=subset(data,Group==1)
> data1=transform(data1,logYield=log(Yield))
> data2=subset(data,Group==2)
> data1
   Group Yield logYield
1      1  4132 8.326517
2      1  3672 8.208492
3      1  3664 8.206311
4      1  4292 8.364508
       .
       .    (More data-lines here)
       .
30     1  4010 8.296547
31     1  5589 8.628556
32     1  5161 8.548886
```

The variables of the datasets can be accessed as

```
> data1$Yield
 [1] 4132 3672 3664 4292 4881 4287 4087 4551 4140 4635 4198 3407 4644 4089 5156
[16] 5348 5436 4911 4775 5931 5040 4520 5381 4787 4717 5716 5832 4865 4811 4010
[31] 5589 5161
```

or directly by its name, `Yield`, if the data frame `data1` has been attached beforehand. It reads

```
> attach(data1)
> Yield
 [1] 4132 3672 3664 4292 4881 4287 4087 4551 4140 4635 4198 3407 4644 4089 5156
[16] 5348 5436 4911 4775 5931 5040 4520 5381 4787 4717 5716 5832 4865 4811 4010
[31] 5589 5161
```

R is told not to look in the particular data frame by the detach-function and then you need the $-notation again to access variables as the following illustrates

```
> detach(data1)
> Yield
Error: Object "Yield" not found
```

Let us attach the data set (group 1) again:

```
> attach(data1)
> Yield
 [1] 4132 3672 3664 4292 4881 4287 4087 4551 4140 4635 4198 3407 4644 4089 5156
[16] 5348 5436 4911 4775 5931 5040 4520 5381 4787 4717 5716 5832 4865 4811 4010
[31] 5589 5161
```

One may access the entries in a vector using the [] notation, for example

```
> Yield[1]
[1] 4132
> Yield[32]
[1] 5161
```

gives the first and the 32th element of the data vector Yield. There are various ways of extracing specific elements of a vector as for example

```
> Yield[1:5]                  # The first five elements of Yield
[1] 4132 3672 3664 4292 4881
> Yield[c(3,10,20)]           # The 3rd, 10th and 20th element of Yield
[1] 3664 4635 5931
> Yield[-(1:20)]              # All but the 1st-20th
 [1] 5040 4520 5381 4787 4717 5716 5832 4865 4811 4010 5589 5161
```

Some functions giving information about a given vector are

```
> min(Yield)         # Minimum value of Yield
[1] 3407
> max(Yield)         # Maximum value of Yield
[1] 5931
> which.max(Yield)   # Gives index of greatest element of Yield
[1] 20
> which.min(Yield)   # Gives index of smallest element of Yield
```

```
[1] 12
> length(Yield)      # Length of the vector Yield
[1] 32
> sort(Yield)        # Sorts the elements of Yield in increasing order
 [1] 3407 3664 3672 4010 4087 4089 4132 4140 4198 4287 4292 4520 4551 4635 4644
[16] 4717 4775 4787 4811 4865 4881 4911 5040 5156 5161 5348 5381 5436 5589 5716
[31] 5832 5931
```

If `B` is a matrix then you access the elements of `B` using the `[,]` notation. For example, `B[1,2]` will give the element in the first row and second column of `B` (assuming that `B` has at least two columns).

Functions in `R`. Now, you have already seen some examples of functions, e.g. `sqrt()`, `read.table()`. A function is called by a name followed by one or more arguments written in parentheses and separated by commas. This is the typical way `R` works also for more complicated calculations and statistical analyses. Most functions will have some optional arguments typically set to a default value as illustrated below

```
> log(exp(1))     # The natural log-function has base argument set to exp(1).
[1] 1
> log(10,base=10) # Now we changed the base value to 10 ie the log10-fct
[1] 1
> log10(10)       # The log10-function has also its own name
[1] 1
```

Logical values. Certain expressions passed on to `R` give the answer `TRUE` or `FALSE` (depending of course on whether they are true or false), for example

```
> (Yield > 4746)
 [1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[13] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
[25] FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
```

The first value of `Yield` is 4132 and therefore the first value of `(Yield > 4746)` is `FALSE`, and so on. The value of `(Yield > 4746)` might be stored as vector

```
> vec=(Yield > 4746)
```

and then used to extract the values of `Yield` that fulfill the condition (that is where the vector `(Yield > 4746)` has the value `TRUE`)

```
> Yield[vec]
 [1] 4881 5156 5348 5436 4911 4775 5931 5040 5381 4787 5716 5832 4865 4811 5589
[16] 5161
```

The function `which()` returns the vector of the indices of `Yield` where the condition (`vec` in this case) is true

```
> which(vec)
 [1]  5 15 16 17 18 19 20 21 23 24 26 27 28 29 31 32
```

The logical operators are < (used above), <=, >, =>, ==, and != where == and != mean equal and not equal, respectively.

Logical expressions may be combined using the logical operators `&` (and) and `|` (or):

```
> (Yield > 4746)&(Yield <= 5000)
 [1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[13] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE
[25] FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE
> Yield[(Yield > 4746)&(Yield <= 5000)]
[1] 4881 4911 4775 4787 4865 4811
> (Yield <=4746)|(Yield > 5000)
 [1]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[13]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE
[25]  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE
> Yield[(Yield <=4746)|(Yield > 5000)]
 [1] 4132 3672 3664 4292 4287 4087 4551 4140 4635 4198 3407 4644 4089 5156 5348
[16] 5436 5931 5040 4520 5381 4717 5716 5832 4010 5589 5161
```

so `(Yield > 4746)&(Yield <= 5000)` has the value `TRUE` where both conditions are true while `(Yield <=4746)|(Yield >5000)` has the value `TRUE` where either of the two conditions are true.

Missing values. Often in real experiments some of the data may be missing for various reasons (experiment failed, measurement below detection limit, etc). This is indicated in `R` with `NA` (**N**ot **A**vailable). Suppose that the first recording of `Yield` is missing (it is not, but we define it to be)

```
> Yield[1]=NA
```

Operations on an element with a `NA`-value gives the value `NA` as for example

```
> Yield[1]+Yield[2]
[1] NA
```

Also a function, such as `sum()`, which normally calculates the sum of the components of the vector, applied to a data-vector will return the value `NA`. If we want to apply the function on the elements of the vector that is not missing, it can be done using `!is.na()` as illustrated below

```
> sum(Yield) # Computes the sum of a data-vector
[1] NA
> is.na(Yield)
 [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
> sum(Yield[ !is.na(Yield)])
[1] 146533
```

Simple descriptive statistics. Simple descriptive statistics such as a histogram may be calculated and plotted by

```
> hist(Yield,breaks=c(3400,3800,4200,4600,5000,5400,5800,6200))
```

see Fig. 3.1, which reproduces Fig. 2.1 in Skovgaard et al. [1999].

We can also calculate the empirical mean, variance, standard deviation, median and quantiles of a vector of numbers as

```
> mean1=mean(Yield) # Calculates the empirical mean
> mean1
[1] 4708.281
> s2.1=var(Yield)   # Calculates the empirical variance
> s.1=sqrt(s2.1)
> s.1
[1] 648.1547
>  sd(Yield)        # Calculates the empirical standard deviation
[1] 648.1547
> median(Yield)     # Calculates the median
[1] 4746
> quantile(Yield)
     0%     25%     50%     75%    100%
3407.00 4183.50 4746.00 5157.25 5931.00
```

The standard deviation is calculated both as the square root of the variance and by using the built in function `sd`. The function `summary()` summarizes most of the above results

```
> summary(Yield)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   3407    4184    4746    4708    5157    5931
```

The boxplot is a diagram giving information on the center, spread, skewness, and length of tails in a data set. It is invoked by

Figure 3.1: A histogram of the milk data.

```
> boxplot(Yield)
```

giving the plot in the left panel of Fig. 3.2. The middle line of the box is the median, the lower and upper lines of the box are the so-called lower and upper hinges (the median of the lower part of the data and the median of the upper part of the data, respectively), which usually are close to the first and third quartiles. The whiskers indicate the range of the data. However, the length of the whiskers are set to be no longer than 1.5 times the length of the box. Data values not contained in this range are marked separately with points (none for the Yield-data). To illustrate this let's make a transformed (right-skewed) data set based on the Yield data and do a boxplot of them:

```
> data.skew=exp((Yield-mean(Yield))/1000)
> boxplot(data.skew)
```

which gives the plot in the right panel of Fig. 3.2.

We have already seen that is quite easy to get high level plots using R. Let us further illustrate the plotting facilities in R using a dataset concerning mites on apple leaves, Table 2.2 in Skovgaard et al. [1999]. A copy of the table is given below.

No. of mites per leave	0	1	2	3	4	5	6	7	≥ 8
No. of leaves	70	38	17	10	9	3	2	1	0

The data are read into R as follows:

```
> mites=0:7
> leaves=c(70,38,17,10,9,3,2,1)
> mites
[1] 0 1 2 3 4 5 6 7
> leaves
[1] 70 38 17 10  9  3  2  1
```

A simple scatter plot and a reproduction of Fig. 2.2 in Skovgaard et al. [1999] given in the same display can be produced as

```
> par(mfrow=c(1,2)) #sets up the plotting window with two panels
> plot(mites,leaves)
> plot(mites,leaves,type='h')
```

which gives the plots in Fig. 3.3.

Figure 3.2: Boxplot of Yield data (left-panel). Boxplot of right skewed data (right-panel)

Figure 3.3: Mites data.

The help system. The `?` may be used to read more about specific functions and options in `R`. For example

```
> ?plot
  type what type of plot should be drawn.  Possible types are
          *  '"p"' for *p*oints,
          *  '"l"' for *l*ines,
          *  '"b"' for *b*oth,
          *  '"c"' for the lines part alone of '"b"',
          *  '"o"' for both "*o*verplotted",
          *  '"h"' for "*h*istogram" like (or "high-density")
             vertical lines,
          *  '"s"' for stair *s*teps,
          *  '"S"' for other *s*teps, see _Details_ below,
          *  '"n"' for no plotting.
```

shows possible choices for the `type`-option using the `plot`-function. The output shown above is only part of what is actually printed.

4

Linear models

Linear models are fitted with the `lm`-function. A call to `lm` produces an object which can then be analyzed with various other functions. For example, `summary` gives us parameter estimates (and their standard errors and the corresponding t-tests); `confint` produces confidence intervals for the parameters; and `anova` gives the analysis of variance table and can be used for testing nested linear models against each other.

In the following we show how to fit and analyze pure ANOVA models (Section 4.1) and models with covariates (Section 4.2). Then we show how to carry out model validation (Section 4.3) and how to estimate linear functions of the parameters (Section 4.4). Finally, Section 4.5 gives a summary of analysis with `lm`, with emphasis on model formulation.

4.1 Analysis of variance (ANOVA)

In this section we consider the class of ANOVA models, that is, models where all the explanatory variables are factors. First we analyse the one-way ANOVA, then the multi-way ANOVA.

4.1.1 One-way ANOVA

Consider Example 3.1 from Bibby et al. [2006] on chlorophyll production in winter wheat. We first read in the data in two vectors, `treat` and `chloro`. Furthermore, we may produce a plot of `chloro` against `treat` with the `plot`-command and a boxplot with the `boxplot`-command. The plots are shown in Figure 4.1.

```
> treat = c(1,1,1,1,2,2,2,2,3,3,3,3)
> chloro = c(54,46,44,53,62,65,74,68,76,69,84,74)
> plot(treat, chloro)
> boxplot(chloro~treat)
```

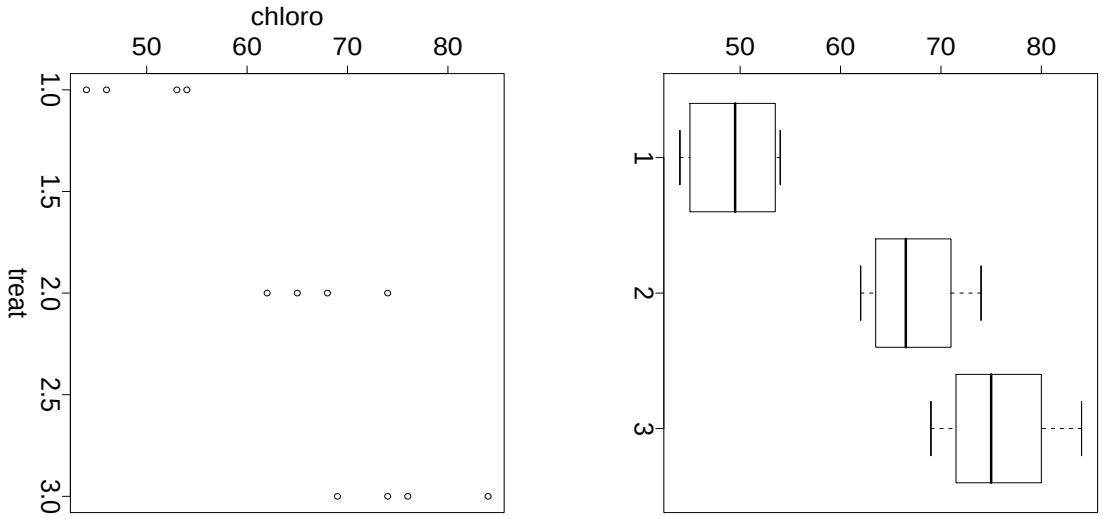

Figure 4.1: Scatter plot and boxplot of the Chlorophyll data.

Model fit, analysis of variance table and test for treatment effect. The one-way ANOVA model, model (3.1) from Bibby et al. [2006], is fitted with `lm`. The response (`chloro`) is written on the left hand side of a `~` (a "tilde"), the factor (`treat`) on the right hand side. We need to tell `R` to use `treat` as a factor rather than as a covariate (numerical variable); we do so with `factor`.

```
> treat = factor(treat)
> model1 = lm(chloro ~ treat)
```

The analysis of variance table is obtained by `anova`:

```
> anova(model1)
Analysis of Variance Table

Response: chloro
          Df  Sum Sq Mean Sq F value    Pr(>F)
treat      2 1464.67  732.33  24.389 0.0002324 ***
Residuals  9  270.25   30.03
```

From the table we read the residual sum of squares ($SS_e = 270.25$) the residual degrees of freedom ($DF_e = 9$) as well as the residual mean square error ($MS_e = SS_e / DF_e = 30.03$). Moreover, we see that there is a highly significant treatment effect. The test for treatment effect could also be carried out by fitting the model without treatment effect, model (3.3) in Bibby et al. [2006], and compare it to the model including the treatment effect by `anova` as follows:

```
> model2 = lm(chloro ~ 1)
> anova(model2,model1)
```

```
Analysis of Variance Table

Model 1: chloro ~ 1
Model 2: chloro ~ treat
  Res.Df     RSS Df Sum of Sq      F    Pr(>F)
1     11 1734.92
2      9  270.25  2   1464.67 24.389 0.0002324 ***
```

Parameter estimates and confidence intervals. The parameter estimates are extracted with `summary`, together with their standard errors and t-tests for the corresponding parameters being zero (some of the output has been suppressed):

```
> summary(model1)

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   49.250      2.740  17.975 2.32e-08 ***
treat2        18.000      3.875   4.645  0.00121 **
treat3        26.500      3.875   6.839 7.57e-05 ***

Residual standard error: 5.48 on 9 degrees of freedom
Multiple R-Squared: 0.8442,     Adjusted R-squared: 0.8096
F-statistic: 24.39 on 2 and 9 DF,  p-value: 0.0002324
```

We see that $\hat{\sigma} = 5.48$ (residual standard error). For the treatment effect, `R` uses the first level of `treat` as reference. That is, the first estimate is the estimate of $\alpha(1)$, $\hat{\alpha}(1) = 49.25$, then follows the estimate $\hat{\alpha}(2) - \hat{\alpha}(1) = 18.00$ and $\hat{\alpha}(3) - \hat{\alpha}(1) = 26.50$. Hence, in order to get an estimate of $\alpha(2)$, we compute

$$\hat{\alpha}(2) = \hat{\alpha}(1) + \big(\hat{\alpha}(2) - \hat{\alpha}(1)\big) = 49.25 + 18.00 = 67.25.$$

A slightly different way of thinking about the estimates is the following. Think about the model written with an intercept, μ:

$$Y_i = \mu + \beta(\texttt{treat}) + e_i$$

and fix $\beta(1)$ to zero. With this parameterization, $\hat{\mu} = 49.25$, $\hat{\beta}(2) = 18.00$, $\hat{\beta}(3) = 26.50$. Note that we could have got the α-estimates right away by fitting the model without intercept:

```
> model1a = lm(chloro ~ treat - 1)
> summary(model1a)
Coefficients:
       Estimate Std. Error t value Pr(>|t|)
treat1    49.25       2.74   17.98 2.32e-08 ***
```

```
treat2    67.25       2.74   24.55 1.48e-09 ***
treat3    75.75       2.74   27.65 5.14e-10 ***

Residual standard error: 5.48 on 9 degrees of freedom
Multiple R-Squared: 0.9947,     Adjusted R-squared: 0.9929
F-statistic: 563.3 on 3 and 9 DF,  p-value: 1.480e-10
```

It is important to realize that `model1` and `model1a` are the same; the only difference is the parameterization. In particular, of course, the two models give the same residual standard error.

Confidence intervals are easily obtained with `confint`. For the original model specification with group 1 as reference, we get

```
> confint(model1)
                2.5 %   97.5 %
(Intercept) 43.051960 55.44804
treat2       9.234648 26.76535
treat3      17.734648 35.26535
```

Change of reference group. By default, the first treatment group is used a reference, but the reference group can be changed with `relevel`:

```
> newtreat = relevel(factor(treat), ref=2)
> model1b = lm(chloro ~ newtreat)
> summary(model1b)
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   67.250      2.740  24.545 1.48e-09 ***
newtreat1    -18.000      3.875  -4.645  0.00121 **
newtreat3      8.500      3.875   2.194  0.05591 .

Residual standard error: 5.48 on 9 degrees of freedom
Multiple R-Squared: 0.8442,     Adjusted R-squared: 0.8096
F-statistic: 24.39 on 2 and 9 DF,  p-value: 0.0002324
```

In particular we see that the p-value for the hypothesis $\alpha(2) = \alpha(3)$ is just above 5%.

4.1.2 Two-way and multi-way ANOVA

In this section we will mainly use the data from Example 3.2 in Bibby et al. [2006] on decomposition of organic matter. First, read the data from the text-file `organic.txt` and attach the dataset. Next, make the time-variable, `tim`, a factor. The medicine factor, `vet`, is automatically a factor since its levels are not numeric.

```
> organic = read.table("organic.txt", header=T)
> attach(organic)
> tim = factor(tim)
```

Interaction, fit of model. First, some general comments: the syntax for the product of two factors, `A` and `B`, is `A:B`, whereas `A*B` is short for the collection of `A:B`, `A`, `B` and `0` (the trivial factor). Hence `model1`, `model1a` and `model1b` below all fit the two-way ANOVA model with interaction, model (3.10) in Bibby et al. [2006].

```
> model1 = lm(y ~ A + B + A:B)
> model1a = lm(y ~ A*B)
> model1b = lm(y ~ A:B)
```

`model1` and `model1a` are exactly the same, whereas `model1b` fits the same model but with another parameterization. The latter might be useful for estimation purposes but for hypothesis testing `model1` or `model1a` is generally recommended. In particular, the output from `anova(model1b)` is almost never useful.

For the example, the model fit and the analysis of variance table goes as follows:

```
> model1 = lm(matter ~ vet + tim + vet:tim)
> anova(model1)
Analysis of Variance Table

Response: matter
          Df  Sum Sq Mean Sq F value    Pr(>F)
vet        1 1.73155 1.73155 79.0972 6.519e-10 ***
tim        2 0.76809 0.38405 17.5432 9.004e-06 ***
vet:tim    2 0.00767 0.00383  0.1751    0.8402
Residuals 30 0.65674 0.02189
```

In particular we find the variance estimate, $\hat{\sigma}^2 = 0.022$.

Hypothesis testing/model reduction. From the `anova(model1)`-output above we see right away that the interaction between `vet` and `tim` is not significant ($p = 0.84$). Alternatively we could have fitted the additive model, model (3.11) from Bibby et al. [2006], and used `anova` to compare the two models:

```
> model2 = lm(matter ~ vet + tim)
> anova(model2,model1)
Analysis of Variance Table

Model 1: matter ~ vet + tim
Model 2: matter ~ vet + tim + vet:tim
  Res.Df     RSS Df Sum of Sq      F Pr(>F)
1     32 0.66441
2     30 0.65674  2   0.00767 0.1751 0.8402
```

Of course, we get the same test! Note the slightly annoying fact that `R` uses the notation "`Model 1`" for the model under the hypothesis and "`Model 2`" for the full model, no matter what names we have used. In the same way, we test for the main effect of `tim`

by fitting the model with `vet` as the only factor and test it against the additive model. Similarly we test for the main effect of `vet`.

```
> model3 = lm(matter ~ vet)
> anova(model3,model2)
Analysis of Variance Table

Model 1: matter ~ vet
Model 2: matter ~ vet + tim
  Res.Df     RSS Df Sum of Sq      F    Pr(>F)
1     34 1.43250
2     32 0.66441  2   0.76809 18.497 4.586e-06 ***

> model4 = lm(matter ~ tim)
> anova(model4,model2)
Analysis of Variance Table

Model 1: matter ~ tim
Model 2: matter ~ vet + tim
  Res.Df     RSS Df Sum of Sq      F    Pr(>F)
1     33 2.39596
2     32 0.66441  1   1.73155 83.397 1.991e-10 ***
```

In conclusion, the additive model, `model2`, cannot be reduced any further, and is thus the final model.

Estimation. As usual the estimates from the final model are extracted by `summary`, and confidence intervals with `confint`:

```
> summary(model2)

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)    2.58329    0.04803  53.784  < 2e-16 ***
vetIvermectin  0.43863    0.04803   9.132 1.99e-10 ***
tim12         -0.19713    0.05883  -3.351  0.00208 **
tim16         -0.35715    0.05883  -6.071 8.83e-07 ***

Residual standard error: 0.1441 on 32 degrees of freedom
Multiple R-Squared:  0.79,      Adjusted R-squared: 0.7703
F-statistic: 40.13 on 3 and 32 DF,  p-value: 5.912e-11

> confint(model2)
                   2.5 %     97.5 %
(Intercept)    2.4854584  2.6811304
vetIvermectin  0.3407918  0.5364638
tim12         -0.3169575 -0.0773092
```

```
tim16          -0.4769741 -0.2373259
```

Estimates for the treatment differences and the time differences are immediately read:

$$\hat{\alpha}(\texttt{Ivermectin}) - \hat{\alpha}(\texttt{Control}) = 0.439$$
$$\hat{\beta}(12) - \hat{\beta}(8) = -0.197; \quad \hat{\beta}(16) - \hat{\beta}(8) = -0.357.$$

Interaction plots. Interaction plots are easily constructed in `R` with the function `interaction.plot`. The commands below produce the two graphs in Figure 4.2.

```
> interaction.plot(tim,vet,matter)
> interaction.plot(vet,tim,matter)
```

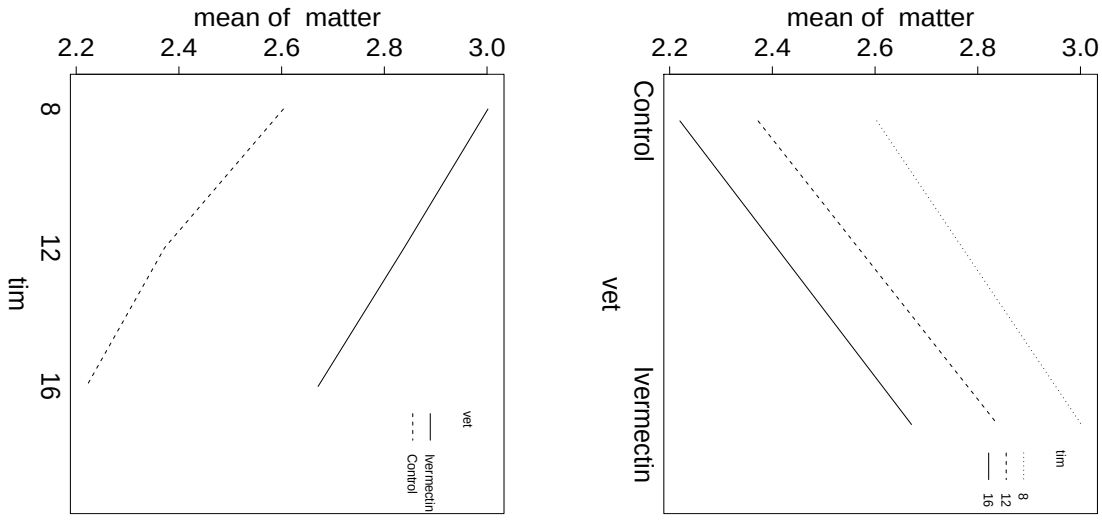

Figure 4.2: Interaction plots for the data on organic matter.

Note that `interaction.plot` always uses the first argument as a factor. Hence, for a factor with numeric values, `R` makes the x-axis equidistant (same distance between subsequent levels) even if the values are *not* equidistant. This means that the plots cannot always be used to assess linearity between the factor values used numerically and the response.

An example with three factors. Consider now example 3.4 from Bibby et al. [2006] on yield of spring wheat, and suppose that the data has been read into `R`, such that a numeric variable `yield` as well as three factors `weed`, `pattern` and `dens` are available. A number of models and tests are listed in Table 3.9 in Bibby et al. [2006]. Some of the models and some of the tests are carried out below (with the same notation as in Table3.9). Part of the output has been suppressed.

```
> model1 = lm(yield ~ weed*pattern*dens)

> model2 = lm(yield ~ weed*pattern + weed*dens + pattern*dens)
> anova(model2,model1)
  Res.Df     RSS Df Sum of Sq      F Pr(>F)
1     38 11.8765
2     36 11.3176  2    0.5589 0.8889 0.4199

> model3b = lm(yield ~ weed*pattern + pattern*dens)
> anova(model3b,model2)
  Res.Df     RSS Df Sum of Sq      F Pr(>F)
1     40 12.0194
2     38 11.8765  2    0.1429 0.2286 0.7967

> model4 = lm(yield ~ weed + pattern*dens)
> anova(model4,model3b)
  Res.Df     RSS Df Sum of Sq      F Pr(>F)
1     41 12.0682
2     40 12.0194  1    0.0488 0.1623 0.6892

> model5 = lm(yield ~ weed + pattern + dens)
> anova(model5,model4)
  Res.Df     RSS Df Sum of Sq      F Pr(>F)
1     43 13.4926
2     41 12.0682  2    1.4245 2.4197 0.1015

> model6a = lm(yield ~ pattern + dens)
> anova(model6a,model5)
  Res.Df     RSS Df Sum of Sq      F  Pr(>F)
1     44 15.6900
2     43 13.4926  1    2.1974 7.0028 0.01132 *
```

The last `anova`-call shows that there is a significant effect of `weed`. The main effects of `pattern` and `density` turn out to be significant as well, so `model5` is the final model. The estimates are as usual extracted with `summary`:

```
> summary(model5)
Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)      6.8525     0.1808  37.903  < 2e-16 ***
weedL.multi     -0.4279     0.1617  -2.646 0.011324 *
patternUniform   0.4454     0.1617   2.755 0.008582 **
dens449          0.8900     0.1980   4.494 5.21e-05 ***
dens721          0.7931     0.1980   4.005 0.000242 ***

Residual standard error: 0.5602 on 43 degrees of freedom
Multiple R-Squared: 0.475,      Adjusted R-squared: 0.4262
```

```
F-statistic: 9.726 on 4 and 43 DF,  p-value: 1.079e-05
```

The adjusted means reported below Table 3.9 in Bibby et al. [2006] are computed in Section 4.4 (Adjusted means with `estimable`).

The product factor in incomplete designs. Sometimes, not all combinations of the factors occur in designs with two (or more) factors. This causes a little trouble in `R`. Suppose not all combinations of `A` and `B` occur. Nevertheless the product factor `A:B` is coded in `R` with all possible levels, and `lm` gives a "strange" output because it tries to estimate all these although it is obviously not possible form the design. Fortunately, the solution is simple. Write

```
> AB = factor(A:B, exclude=NA)
```

and use this newly constructed product factor instead of `A:B` in the model. The new factor only contains the combinations that were actually present in the design.

4.2 Covariate models

We now turn to models which include covariates (numerical values as explanatory variables). First we consider the simple linear regression, then quadratic regression and finally models including both covariates and factors.

4.2.1 Simple linear regression

Suppose we have observed pairs (x_i, y_i) for $i = 1, \ldots, n$ and want to make a linear regression of y on x, $y_i = \alpha + \beta x_i + e_i$. The values below are taken from Example 12.2 in Skovgaard [2000] on the relation between digestion of fat and the amount of a particular acid. The model is fitted with `lm` as follows:

```
> x = c(29.8,30.3,22.6,18.7,14.8,4.1,4.4,2.8,3.8)
> y = c(67.5, 70.6, 72.0, 78.2, 87, 89.9, 91.2, 93.1, 96.7)
> model1 = lm(y~x)
```

The call is very similar to that of a one-way ANOVA, the only difference is that `x` is now a numeric variable (a covariate) whereas in the ANOVA setting the explanatory variable was a grouping variable (a factor).

As usual `anova` gives the analysis of variance table, and `summary` gives the estimates:

```
> anova(model1)
Analysis of Variance Table

Response: y
          Df Sum Sq Mean Sq F value    Pr(>F)
```

```
x          1 896.76  896.76  101.63 2.028e-05 ***
Residuals  7  61.76    8.82

> summary(model1)

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 96.53336    1.67518   57.63 1.24e-10 ***
x           -0.93374    0.09262  -10.08 2.03e-05 ***

Residual standard error: 2.97 on 7 degrees of freedom
Multiple R-Squared: 0.9356,     Adjusted R-squared: 0.9264
F-statistic: 101.6 on 1 and 7 DF,  p-value: 2.028e-05
```

We read the parameter estimates: $\hat{\alpha} = 96.5$, $\hat{\beta} = -0.93$, $\hat{\sigma} = 2.97$ or $\hat{\sigma}^2 = \mathrm{MS_e} = 8.82$. The test of the hypothesis $\beta = 0$ is carried out as a t-test by `summary` and as a F test by `anova` (the tests are of course equivalent as $F = t^2$). The F-test could also have been carried out by fitting the model *without* `x` and compare it to `model1`:

```
> anova(model2,model1)
Analysis of Variance Table

Model 1: y ~ 1
Model 2: y ~ x
  Res.Df    RSS Df Sum of Sq      F    Pr(>F)
1      8 958.53
2      7  61.76  1    896.76 101.63 2.028e-05 ***
```

More details on this example are given in Martinussen and Skovgaard [2006].

4.2.2 Quadratic regression, test for linearity

Consider the dataset from Example 5.5 from Bibby et al. [2006] on the relationship between height and diameter of Corsican pines. First, we type the data, make a new variable with the quadratic diameter and plot `h` against `d` (as in Figure 5.5 from Bibby et al. [2006]):

```
> h = c(32,31,30,29,29,28,25,23,20,18,17,17,16,16,15,13,11,11)
> d = c(22.7,22.7,22.6,22.6,21.9,21.9,21.8,21.0,20.4,18.6,19.2,18.9,18.5,
        18.1,17.7,17.2,16.5,15.5)
> d2 = d^2
> plot(d,h)
```

The quadratic as well as the linear regression of d on h are fitted and the linear model is tested against the quadratic (as a test for linearity):

```
> model1 = lm(h ~ d + d2)
> model2 = lm(h ~ d)
> anova(model2,model1)
Analysis of Variance Table

Model 1: h ~ d
Model 2: h ~ d + d2
  Res.Df    RSS Df Sum of Sq      F   Pr(>F)
1     16 4.4535
2     15 1.8640  1    2.5895 20.839 0.000372 ***
```

We conclude that the relationship is not linear, and extract the estimates from the quadratic regression model:

```
> summary(model1)
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  8.727814   1.001295   8.717 2.95e-07 ***
d            0.773303   0.100342   7.707 1.36e-06 ***
d2          -0.010489   0.002298  -4.565 0.000372 ***

Residual standard error: 0.3525 on 15 degrees of freedom
Multiple R-Squared: 0.9803,     Adjusted R-squared: 0.9777
F-statistic: 373.7 on 2 and 15 DF,  p-value: 1.599e-13
```

We see that $\hat{\beta}_0 = 8.73$, $\hat{\beta}_1 = 0.773$, $\hat{\beta}_2 = -0.0105$, $\hat{\sigma} = 0.353$, following the notation from Bibby et al. [2006].

4.2.3 Models with both factors and covariates, test for linearity

In this section we will consier Example 4.2 from Bibby et al. [2006] on hydrolysis of amino acids. The dataset consists of three variables: the response `serine` and two explanatory variables, `feed` and `hour`. The variables are as follows:

```
> serine
 [1] 4.47 4.34 4.22 4.10 3.48 4.46 4.30 4.19 4.08 3.53 4.23 4.09 4.00 3.80 3.29
[16] 4.10 4.13 3.92 3.79 3.20 5.26 5.09 4.94 4.79 4.02 5.30 5.08 5.00 4.75 4.03
[31] 4.46 4.29 4.13 4.05 3.38 4.43 4.23 4.15 3.95 3.38 5.61 5.47 5.31 5.08 4.43
[46] 5.70 5.46 5.35 5.12 4.28
> feed
 [1] barley barley barley barley barley barley barley barley barley barley
[11] fish   fish   fish   fish   fish   fish   fish   fish   fish   fish
[21] mais   mais   mais   mais   mais   mais   mais   mais   mais   mais
[31] meat   meat   meat   meat   meat   meat   meat   meat   meat   meat
[41] soy    soy    soy    soy    soy    soy    soy    soy    soy    soy
Levels: barley fish mais meat soy
```

```
> hour
 [1]  8 16 24 32 72  8 16 24 32 72  8 16 24 32 72  8 16 24 32 72  8 16 24 32 72
[26]  8 16 24 32 72  8 16 24 32 72  8 16 24 32 72  8 16 24 32 72  8 16 24 32 72
```

`feed` is string valued and is thus automatically used as a factor by `R`. `hour` takes numeric values, and can (and will) be used both as a factor and as a covariate. We keep the `hour`-variable as numeric and make a new variable, `hourfac`, as a factor. Moreover, a log-transformation turns out to be appropriate, so we construct a new variable `logserine` with the log 10-transformed values:

```
> hourfac = factor(hour)
> hourfac
 [1] 8  16 24 32 72 8  16 24 32 72 8  16 24 32 72 8  16 24 32 72 8  16 24 32 72
[26] 8  16 24 32 72 8  16 24 32 72 8  16 24 32 72 8  16 24 32 72 8  16 24 32 72
Levels: 8 16 24 32 72
> logserine = log10(serine)
```

Model fit. The time variable may be used either as a factor (`hourfac`) or as a covariate (`hour`). First, the factor models with and without interaction between `hourfac` and `feed`:

```
> model1 = lm(logserine ~ feed:hourfac)
> model2 = lm(logserine ~ feed + hourfac)
> model4 = lm(logserine ~ feed)
> model5 = lm(logserine ~ hourfac)
```

The names of the models correspond to the names in Bibby et al. [2006], and the test for interaction, say, is carried out by `anova` as usual:

```
> anova(model2,model1)
Analysis of Variance Table

Model 1: logserine ~ feed + hourfac
Model 2: logserine ~ feed:hourfac
  Res.Df        RSS Df  Sum of Sq      F Pr(>F)
1     41 0.00076722
2     25 0.00050543 16 0.00026179 0.8093 0.6643
```

Next, we turn to models with the time variable as covariate. `model3` is is the model with different intercepts but a common slope for all feeds as in Bibby et al. [2006] (parallell lines). `model6` is the model with different intercepts and different slopes (not discussed in Bibby et al. [2006]):

```
> model3 = lm(logserine ~ feed + hour)
> model6 = lm(logserine ~ feed + hour + feed*hour)
```

Test for linearity. Any two linear models where one is a submodel of the other can be compared by `anova`. Hence, for example, `model3` may be tested against `model2` as in Bibby et al. [2006]. This is a test of the hypothesis that the time-logserine relation is linear.

```
> anova(model3,model2)
Analysis of Variance Table

Model 1: logserine ~ feed + hour
Model 2: logserine ~ feed + hourfac
  Res.Df        RSS Df  Sum of Sq      F Pr(>F)
1     44 0.00080247
2     41 0.00076722  3 0.00003525 0.6279 0.6012
```

Had the interaction between `feed` and `hour` been significant, such that `model2` had been rejected, a test for linearity could have been carried out by testing `model6` against `model1`. Tests of quadratic models, say, are carried out in the same way: fit the model with the quadratic term(s) and test it against the relevant factor model.

Estimation. The model with parallell lines, `model3`, turns out to be the final model. If we want the estimates as in Table 4.6 of Bibby et al. [2006], it is useful to fit the model without intercept. Estimates are extracted with `summary`, confidence intervals could be extracted with `confint` (not shown):

```
> model3a = lm(logserine ~ feed + hour - 1)
> summary(model3a)
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
feedbarley  0.6668516  0.0015819  421.54   <2e-16 ***
feedfish    0.6380504  0.0015819  403.33   <2e-16 ***
feedmais    0.7354692  0.0015819  464.92   <2e-16 ***
feedmeat    0.6588011  0.0015819  416.45   <2e-16 ***
feedsoy     0.7664038  0.0015819  484.47   <2e-16 ***
hour       -0.0017684  0.0000271  -65.25   <2e-16 ***

Residual standard error: 0.004271 on 44 degrees of freedom
Multiple R-Squared:     1,      Adjusted R-squared:     1
F-statistic: 1.886e+05 on 6 and 44 DF,  p-value: < 2.2e-16
```

In Section 4.4 we show how to make pairwise comparisons of the feed types with the `estimable` function.

4.3 Model validation

In this section we will use the hydrolysis data again, see Section 4.2.3. Most often model validation is carried out for the initial model which in this case was two-way analysis

with interaction between `feed` and `hourfac` (`model1`). Here we will, however, follow the approach from Bibby et al. [2006] and do it in the final model. This is mostly a matter of taste. The final model was the linear regression model with the time variable `hour` as covariate, and with different intercepts but the same slope for all levels of the feed factor, `feed`. The model was fitted with

```
> model3 = lm(logserine ~ feed + hour)
```

4.3.1 Analysis of residuals

The ingredients for the residual analysis are easily extracted from `model3`. The predicted values are extracted with `predict` (or `fitted`), the raw residuals with `residuals (or redis)` and the standardized residuals with `rstandard`.

```
> pred3 = predict(model3)
> res3 = residuals(model3)
> sres3 = rstandard(model3)
```

The objects may then be used for residual plots (use `plot`) and QQ-plots (use `qqnorm`). The `qqnorm`-call below plots the quantiles of `sres` against those of the standard normal distribution. The line with zero intercept and slope one makes the comparison easier. For the residual plots, one may add a horizontal line at zero level with `abline(h=0)`.

```
> plot(pred3, res3)
> plot(pred3, sres3)
> qqnorm(sres3)
> abline(0,1)
```

The plots are shown in Figure 4.3, and look quite okay. Note that QQ-plot is not identical to that in Figure 5.3 in Bibby et al. [2006] where the quantiles of the standardized residuals are compared to a t-distribution rather than the normal distribution.

4.3.2 Transformation, Box-Cox analysis

Transformation. As explained in Bibby et al. [2006] a transformation of the response and/or the covariates may help remedy problems with the model assumptions. As an example, consider the hydrolysis data once again and remember that the original response, `serine`, was log-transformed. Consider for a moment the untransformed response, with the same model as above, and construct the residual plot:

```
> model3.orig = lm(serine ~ feed + hour)
> plot(predict(model3.orig), rstandard(model3.orig))
```

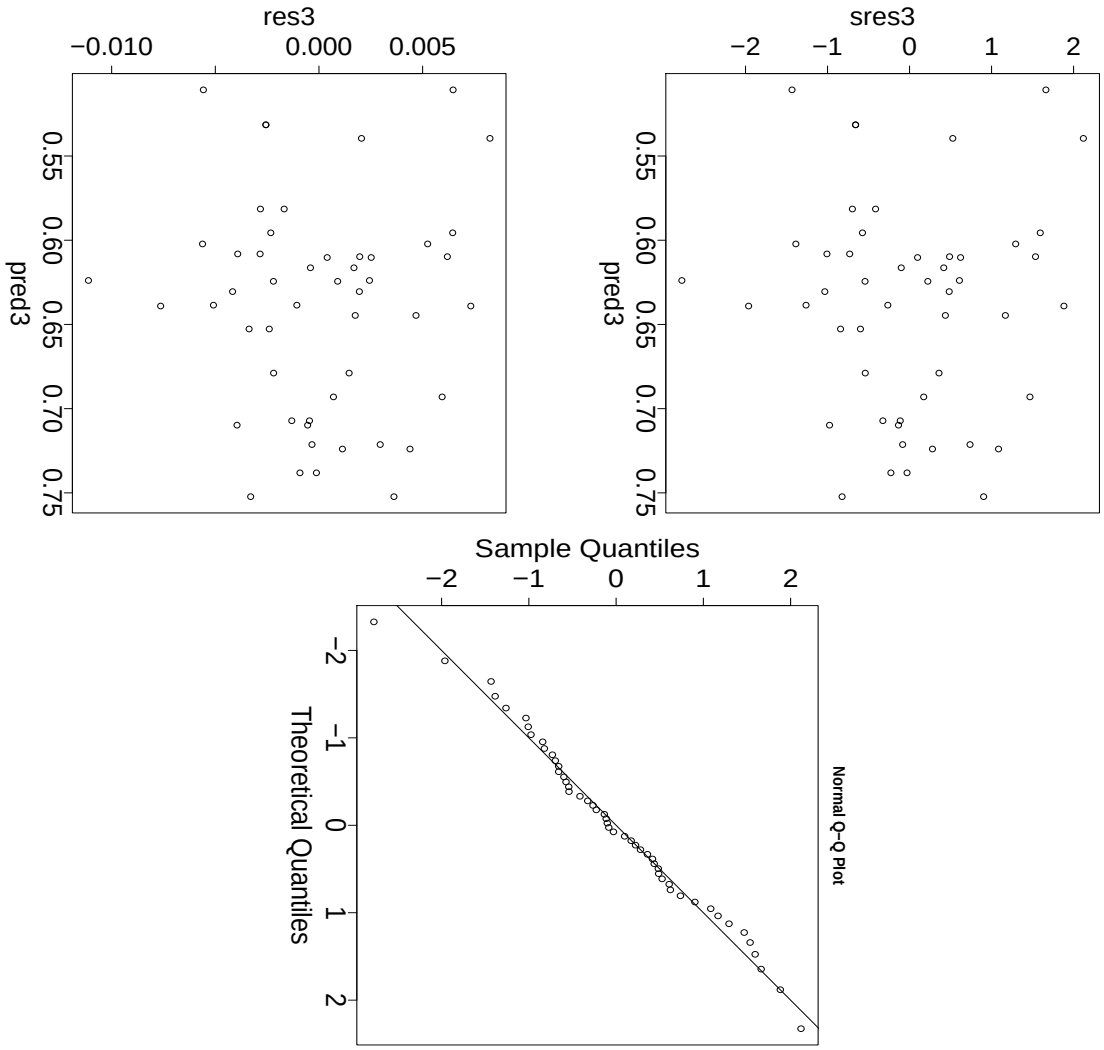

Figure 4.3: Model validation for the hydrolysis data.

The left part of Figure 4.4 shows the plot. There is a clear pattern of large positive residuals for small and large predicted values and many negative residuals for medium predicted values, so the model clearly does not catch the variation in the data. Compare with the upper right graph in Figure 4.3 where there is no such pattern. Hence, the log-transformation seems appropriate for these data.

Box-Cox analysis. A Box-Cox analysis can be useful in order to choose a transformation of the data. It compares, in a certain way, power transformations (y^λ) and the logarithmic transformation, and chooses "the best" of all those. A Box-Cox analysis is easily carried out in `R` with the `boxcox`-function. This function is not part of the base package of `R`, but is part of the add-on package `MASS`. Hence, this package should be loaded before `boxcox` can be used; see Appendix B for details on how to use add-on packages.

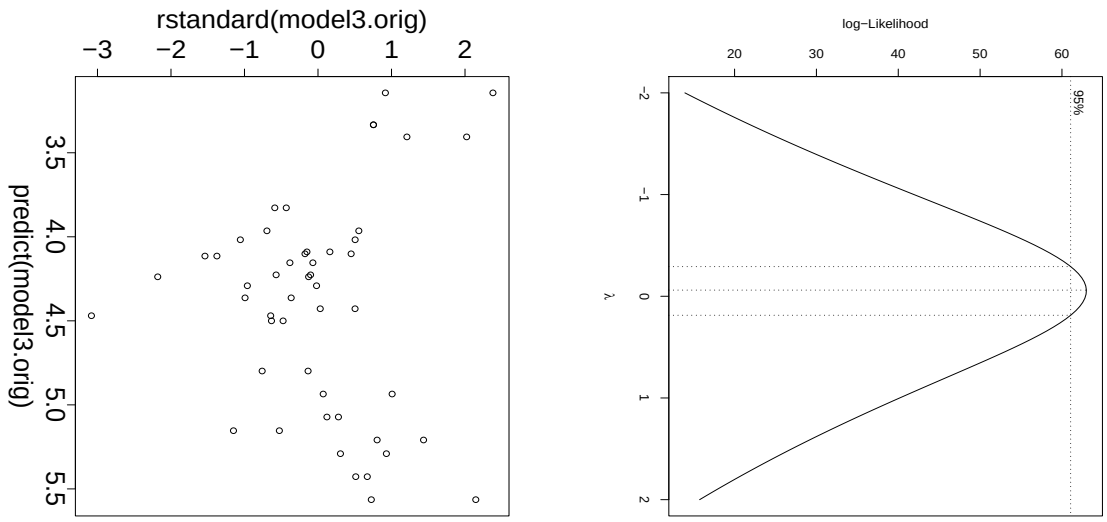

Figure 4.4: Residual plot and Box-Cox plot for the hydrolysis data (untransformed data).

As argument, `boxcox` takes a `lm`-like model formulation. The output is a graph giving the optimal power where zero corresponds to the log-transformation. For the hydrolysis data we write as follows and get the right part of Figure 4.4:

```
> library(MASS)
> boxcox(serine ~ feed + hour)
```

Indeed the optimal value is very close to zero, suggesting a log-transformation (for this particular model).

4.4 Estimation of contrasts

We are often interested in estimating certain functions of the parameters of a model. In particular we may be interested in linear functions, also called contrasts. We easily get the estimates themselves "by hand" from the parameter estimates by simply applying the function to the estimates. Usually we want standard errors and/or confidence limits as well for the contrasts, which are not so easy to compute. We want `R` to help us!

Consider for example the hydrolysis data again, and assume that we are interested in the difference in log-serine amount between feed types "fish meal" and "maize", that is, we want an estimate of $\alpha(\texttt{mais}) - \alpha(\texttt{fish})$. From the `summary`-output in the end of Section 4.2 we easily get

$$\hat{\alpha}(\texttt{mais}) - \hat{\alpha}(\texttt{fish}) = 0.735 - 0.638 = 0.097$$

but how about a standard error, a confidence interval or a test for the hypothesis of no difference?

For simple contrasts as this one we easily reparametrize (use one of the groups as reference) and use `relevel` as explained in Section 4.1.1. For more complicated contrasts as adjusted means, the `estimable`-function can do the job.

The `relevel`-function. We change the reference group to `fish` and fit the model once again with the new parameterization. Remember that the model is unchanged, only the parameterization is changed. We get the following:

```
> newfeed = relevel(factor(feed), ref="fish")
> model3.contr = lm(logserine ~ newfeed + hour)
> summary(model3.contr)
Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)    0.6380504  0.0015819  403.33  < 2e-16 ***
newfeedbarley  0.0288013  0.0019099   15.08  < 2e-16 ***
newfeedmais    0.0974188  0.0019099   51.01  < 2e-16 ***
newfeedmeat    0.0207508  0.0019099   10.87 4.86e-14 ***
newfeedsoy     0.1283535  0.0019099   67.21  < 2e-16 ***
hour          -0.0017684  0.0000271  -65.25  < 2e-16 ***
> confint(model3.contr)
                    2.5 %       97.5 %
(Intercept)    0.634862176  0.641238563
newfeedbarley  0.024952192  0.032650345
newfeedmais    0.093569720  0.101267872
newfeedmeat    0.016901682  0.024599835
newfeedsoy     0.124504407  0.132202560
hour          -0.001823011 -0.001713775
```

We find the expected estimate (0.097) as well as a t-test for the hypothesis of no difference ($t = 51.0$, $p = 0$) and a 95% confidence interval $(0.094, 0.101)$.

The `estimable`-function. The `estimable`-function is not part of the standard part of `R`, but is part of the add-on package `gmodels`. Hence, this package should be loaded before `estimable` can be used, see Appendix B for details on how to use add-on packages.

One parameterization of the model is given in `model3a`:

```
> model3a = lm(logserine ~ feed + hour - 1)
> summary(model3a)
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
feedbarley  0.6668516  0.0015819  421.54   <2e-16 ***
feedfish    0.6380504  0.0015819  403.33   <2e-16 ***
feedmais    0.7354692  0.0015819  464.92   <2e-16 ***
feedmeat    0.6588011  0.0015819  416.45   <2e-16 ***
feedsoy     0.7664038  0.0015819  484.47   <2e-16 ***
hour       -0.0017684  0.0000271  -65.25   <2e-16 ***
```

The model is written as $\alpha(\texttt{feed}_i)+\beta\cdot\texttt{hour}_i+e_i$ and the estimates are given in the above list. We are interested in $\hat{\alpha}(\texttt{mais})-\hat{\alpha}(\texttt{fish})$, which we may also write as

$$0\cdot\hat{\alpha}(\texttt{barley})-1\cdot\hat{\alpha}(\texttt{fish})+1\cdot\hat{\alpha}(\texttt{fish})+0\cdot\hat{\alpha}(\texttt{meat})+0\cdot\hat{\alpha}(\texttt{soy})+0\cdot\hat{\beta}$$

We need to tell `R` the coefficients (the 1, the -1 and the zeros) of this linear combination.

Below, `contr` defines the proper linear combination and associates a name, `fish-mais` to it. The order of the parameters is the same as in the above `summary`-output. More precisely, the `rbind`-command makes a row-matrix (a matrix with just one row) and assigns the name `fish-mais` to this row. The `contr`-object can then be used as argument to the `estimable`-function.

```
> library(gmodels)
> contr = rbind('fish-meat' = c(0,-1,1,0,0,0))
> estimable(model3a,contr,conf.int=0.95)
           Estimate  Std. Error  t value DF Pr(>|t|)   Lower.CI  Upper.CI
fish-meat 0.0974188 0.001909863 51.00826 44        0 0.09356972 0.1012679
```

Of course we get the same as with the `relevel`-method.

Suppose now that we want to estimate the expected serine amount corresponding to a hydrolysis time of 16 hours, for the feed types "barley" and "meat and bone meal". The expected values could be computed by hand or extracted with `predict`. The corresponding standard errors and confidence intervals may be computed with `estimable` as follows:

```
> barley.est = c(1,0,0,0,0,16)
> meat.est = c(0,0,0,1,1,16)
> est = rbind(barley.est, meat.est)
> estimable(model3a, est, conf.int=0.95)
            Estimate  Std. Error  t value DF Pr(>|t|)  Lower.CI  Upper.CI
barley.est 0.6385574 0.001405733 454.2523 44        0 0.6357243 0.6413904
meat.est   1.3969107 0.002263106 617.2537 44        0 1.3923497 1.4014717
```

In this case the `rbind`-command (`rbind` for "row-bind") constructs a 2×6-matrix (one row per parameter function) with the coefficients, and a name is associated with each row (each parameter function). The model uses the log-transformed variable as response, so the estimates and confidence limits should now be "back-transformed" (with 10^x) in order to get values on the original scale.

Adjusted means with `estimable`. Adjusted means are special linear functions of the parameters and may thus be computed with `estimable`. Consider the spring wheat data from Section 4.1.2 (Example 3.4 from Bibby et al. [2006]). Recall that the final model was the additive model with an effect of weed, pattern and density, that is,

$$y_i=\mu+\alpha(\texttt{weed}_i)+\beta(\texttt{pattern}_i)+\gamma(\texttt{dens}_i)+e_i$$

which was fitted as follows:

```
> model5 = lm(yield ~ weed + pattern + dens)
> summary(model5)
Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)      6.8525     0.1808  37.903  < 2e-16 ***
weedL.multi     -0.4279     0.1617  -2.646 0.011324 *
patternUniform   0.4454     0.1617   2.755 0.008582 **
dens449          0.8900     0.1980   4.494 5.21e-05 ***
dens721          0.7931     0.1980   4.005 0.000242 ***
```

Let us compute the adjusted means for `weed`. For `c.album` we are interested in

$$d = \mu + \alpha(\texttt{c.album}) + \bar{\beta} + \bar{\gamma}$$

where $\bar{\beta}$ is the average of the two β-parameters and γ is the average of the three γ-parameters. Rewrite d as

$$d = 1 \cdot \mu + 1 \cdot \alpha(\texttt{c.album}) + 0 \cdot \alpha(\texttt{l.multi}) + \frac{1}{2}\big(\beta(\texttt{row}) + \beta(\texttt{uniform})\big) + \frac{1}{3}\big(\gamma(204) + \gamma(449) + \gamma(721)\big)$$

Note the parameterization used by `R`, where the combination `C.album, row, 204` is used as reference. Another way to think about this is that the parameters $\alpha(\texttt{c.album})$ $\beta(\texttt{row})$ and $\beta(207)$ have been fixed to zero. Coefficients for these parameter should *not* be included in the list of coefficients. Hence, the relevant vector of coefficients is $(1, 0, 1/2, 1/3, 1/3)$. Similarly for the weed `l.multi`.

```
> library(gmodels)
> adj.c.album = c(1,0,1/2,1/3,1/3)
> adj.l.multi = c(1,1,1/2,1/3,1/3)

> adj = rbind(adj.c.album, adj.l.multi)
> estimable(model5, adj, conf.int=0.95)
             Estimate Std. Error  t value DF Pr(>|t|) Lower.CI Upper.CI
adj.c.album 7.636250  0.1143427 66.78386 43        0 7.405656 7.866844
adj.l.multi 7.208333  0.1143427 63.04146 43        0 6.977739 7.438927
```

Note that the actual parameterization of the model is not important (since we are only considering estimable contrasts), but the coefficients change according to the parameterization. Therefore, always make a `summary` of the model before you specify the coefficients. Moreover, it is always a good idea to compute the estimate by hand and compare with the `estimable`-output in order to check if the vectors of coefficients have been specified correctly.

4.5 Summary: analysis with `lm`

This chapter ends with a summary of how to use `lm` for analysis of linear models.

Description	Model term	R-syntax
Intercept	μ	`1`
Main effect of `A`	$\alpha(\mathtt{A}_i)$	`A`
Product factor (use for interaction)	$\gamma(\mathtt{A}_i, \mathtt{B}_i)$	`A:B`
x as covariate	$\beta \cdot \mathtt{x}_i$	`x`
Slope depends on level of `A`	$\beta(\mathtt{A}_i) \cdot x_i$	`A:x`

Table 4.1: R-syntax for various model terms.

Description	Model for Ey_i	R-syntax to `lm`
One-way ANOVA	$\mu + \alpha(\mathtt{A}_i)$	`y~A`
	$\alpha(\mathtt{A}_i)$	`y~A-1`
Two-way ANOVA w. interact.	$\mu + \gamma(\mathtt{A}_i, \mathtt{B}_i)$	`y~A:B`
	$\mu + \alpha(\mathtt{A}_i) + \beta(\mathtt{B}_i) + \gamma(\mathtt{A}_i, \mathtt{B}_i)$	`y~A+B+A:B` or `y~A*B`
	$\gamma(\mathtt{A}_i, \mathtt{B}_i)$	`y~A:B-1`
Two-way ANOVA, additive	$\mu + \alpha(\mathtt{A}_i) + \beta(\mathtt{B}_i)$	`y~A+B`
Linear regression	$\mu + \beta \cdot x_i$	`y~x`
Linear regr. through $(0, 0)$	$\beta \cdot x_i$	`y~x-1`
Diff. intercept, diff. slopes	$\mu + \alpha(\mathtt{A}_i) + \beta \cdot x_i + \gamma(\mathtt{A}_i) \cdot x_i$	`y~A+x+A:x` or `y~A*x`
	$\alpha(\mathtt{A}_i) + \gamma(\mathtt{A}_i) \cdot x_i$	`y~A*x-1-x`
Diff. intercept, same slopes	$\mu + \alpha(\mathtt{A}_i) + \beta \cdot x_i$	`y~A+A:x`
	$\alpha(\mathtt{A}_i) + \beta \cdot x_i$	`y~A+A:x-1`
Same intercept, diff. slopes	$\mu + \gamma(\mathtt{A}_i) \cdot x_i$	`y~A:x`
One common mean	μ	`y~1`

Table 4.2: The R-syntax for some common models.

Model formulation. A call to `lm` includes a two-sided expression with the response on the left hand side of a `~` (a "tilde") and a number of model terms of the right hand side, for example,

```
> lm(y ~ A*B)
```

In the following, let `y` be a response variable, `A` and `B` factors and `x` a covariate. Table 4.1 summarizes the translation of certain effects in a model to R-syntax. Moreover, notice the different meanings of `:` and `*` in the model formulation. For factors `A` and `B`, `A:B` is the product factor whereas `A*B` is short for `A+B+A:B`. For a factor `A` and a covariate `x`, `A:x` means that the coefficient to `x` depends on the level of `A` whereas `A*x` is short for `A+x+A:x`. As default, R includes an intercept term in the model; add `-1` to the model formula if you want the model without intercept.

Table 4.2 gives the R-syntax for a number of commonly used models, some of them for different parameterizations (recall that different parameterizations may be useful for different purposes). Of corse, more factors and/or covariates may be used in the model formulation.

Estimation and confidence limits. For a given fitted model, `model = lm(...)` we can use `summary(model)` to extract the parameter estimates along with their standard errors and t-tests for each parameter being equal to zero. Confidence intervals for the parameters are computed with `confint(model)`. The default confidence level is 95%, but can be changed to 90%, say, with `confint(model, level=0.90)`.

For a factor, `R` chooses one of the levels as the reference group. The reference group can be changed with `relevel` as illustrated in Sections 4.1.1 and 4.4.

Contrasts (linear parameter functions) can be estimated with `estimable` from the `gmodels`-package as illustrated in Section 4.4. Remember to load the `gmodel` before calling `estimable`. Generally, remember that the coefficients for a certain contrast depend on the specific parameterization of the model, so it is generally recommended to make a `summary(model)` before constructing the vector of coefficients. Also, remember that coefficients for parameters set to zero by `R` should not be included in the vector of coefficients.

Analysis of variance table. The analysis of variance table is found by `anova(model)`. From the output one can read the residual sum of squared (SS_e), the residual degrees of freedom (DF_e), and the residual mean square error (MS_e). Also, sequantial tests are performed, from the bottom and upwards.

Hypothesis testing. As noted above some hypotheses in `model` can be tested in the analysis of variance table obtained by `anova(model)`. More generally, we recommend to test hypotheses in linear models as follows: Fit the full model (`full.model`) and the model under the hypothesis (`hyp.model`). Then carry out the appropriate F-test for `hyp.model` against `full.model` by `anova(hyp.model,full.model)`.

Predicted values, residuals. Predicted values are extrected with `predict(model)` or `fitted(model)`. Raw residuals are extracted by `residuals(model)` and the standardized residuals by `rstandard(model)`.

5

Models with random effects

In this chapter we will show how to use `R` for analysis of Gaussian models with random effects and a linear fixed part, the so-called mixed linear models. We will mainly use the `lme`-function which works for balanced as well as unbalanced data, but also give some comments on analysis with two other functions: `lmer` which is a new version of `lme` where all the facilities from `lme` are not yet built in and `aov` which is for balanced data only. Except for the random part, `lme` works quite similarly to `lm`. The fixed part is specified in the same way, estimates are extracted with `summary` and tests can be performed with `anova`. Annoyingly, confidence intervals are computed with another function, though, namely `intervals`. Before we get started with the analyses, we give some general comments on hypothesis testing in models with random effects.

5.1 F-tests and likelihood ratio tests

As described in Bibby et al. [2006] hypothesis tests can be carried out as F-tests if the design is balanced in a certain sense. In this case the F-statistic is computed as the fraction of two mean square errors (MSE's). The factor diagram shows which MSE should appear in the denominator as well as the degrees of freedom for the F-test. The `aov`-function can be used to analyse data in this way.

Many experiments are not balanced, though, due to the design itself or due to missing values. Then the exact distributions for the test statistics are no longer F-distributions, and we need approximate methods. There is a lot of debate going on among experts about which approximations to use. Often approximate F-tests are used, with various approximations of the denominator degrees of freedom. `R` does not support this solution, though (for many reasons).

Instead, we will carry out likelihood ratio (LR) tests. The rationale for the LR test is the following: For a given model, the maximum of the likelihood function measures (in a certain sense) how well the model fits the data. Hence, if we compare the maximum of the likelihood function under a given model with the maximum of the likelihood function under a null model (assuming a hypothesis to be true), we have a measure of the discrepancy of the models. In other words we measure how much worse the null model

fits the data compared to the full model. This should be compared to the difference in dimensions of the two models: how many more parameters are used in the full models compared to the null model?

To be precise, we use

$$LR = 2 \cdot \log L(\text{full model}) - 2 \cdot \log L(\text{null model})$$

as a test statistic. This statistic is approximately χ^2-distributed; the degrees of freedom is equal to the decrement in model dimensions (number of parameters in the model) from the full model to the null model. The χ^2-approximation to the distribution of LR is good when there are "many" observations (whatever that means). For small and moderately-sized datasets, however, the experience is that these approximate p-values tend to be too small, thereby sometimes overestimating the importance of certain effects. Therefore it is sometimes recommended to compute a better approximation to the p-value by so-called parametric bootstrap (or simulation) if the approximate p-value is below the significance level, but not very small. This is, as we shall see, quite easy to do with `lme`.

Some comments on estimation methods: In order to make likelihood ratio tests *for fixed effects* it is absolutely essential that the models are fitted with the maximum likelihood (ML) method. However, it is well-known that the Restricted Maximum Likehood (REML) method generally produced better estimates. Hence, we recommend to always *take the estimates for the final model from the REML-fit rather than the ML-fit*; also if the model reduction has been carried out by likelihood ratio test using ML. To complicate matters, it is usually recommended to use REML estimation for testing hypotheses about the random part of the model, ie. use

$$REML.LR = 2 \cdot \log ReL(\text{full model}) - 2 \cdot \log ReL(\text{null model})$$

where $\log ReL$ stands for the restricted log-likelihood function. Note that `R` as default uses REML estimation.

`lme` (or `lmer`) can be used in this way for any mixed model, the design being balanced or not. However, the problems concerning the approximations that we use, make it preferable to use the exact F-tests from `aov` for the balanced cases. It may not be easy, though, to find out if the experiment is balanced in the appropriate sense, so if you are in doubt then we recommend that you use the general method (`lme`/`lmer`).

5.2 Analysis of linear mixed models (`lme` and `lmer`)

5.2.1 A single random factor.

In this section we will use the dataset on tenderness of pork chops from Example 2.7 and 8.1 in Bibby et al. [2006]. Suppose that the four relevant variables are available: the response variable `tender` as well as the factors `Porker` with levels $1, 2, \ldots, 24$, `pH` with levels `low` and `high`, and `chilling` with levels `fast` and `tunnels`.

Model fit. First the package `nlme` is loaded so we can use `lme`. Second, we use `lme` to fit model (8.1) from Bibby et al. [2006] with interaction between `pH` and `chilling`. `Porker` is used as a random factor. The fixed effect part of a model is written in the usual `lm`-way. The random part is specified by a one-sided expression followed by some grouping variables after the `|`. Here, we have only one grouping variable, `Porker`. For a start, we use REML-estimation (which is default, so the option `method=REML` could be skipped).

```
> library(nlme)
> model1 = lme(tender ~ chilling + pH + pH:chilling, random =~ 1 | Porker,
               method = "REML")
```

Estimation, confidence intervals The `summary-function` is used in the usual way to extract the estimates for the parameters from the fixed part of the model as well as for the variance parameters from the random part:

```
> summary(model1)
Linear mixed-effects model fit by REML

Random effects:
 Formula: ~1 | Porker
        (Intercept)  Residual
StdDev:    1.118207 0.6813447

Fixed effects: tender ~ chilling + pH + pH:chilling
                          Value Std.Error DF   t-value p-value
(Intercept)           7.010000 0.3780010 22 18.544922  0.0000
chillingtunnel        0.212500 0.2781578 22  0.763955  0.4530
pHlow                -1.545833 0.5345742 22 -2.891710  0.0085
chillingtunnel:pHlow  0.165000 0.3933745 22  0.419448  0.6790
```

The fixed part parameters are interpreted in the usual way. Moreover we see that $\hat{\sigma}_\eta = 1.12$ (the standard deviation for the random `Porker`-factor and $\hat{\sigma} = 0.68$ (the residual standad deviation). If we want the squared estimates directly we can use `VarCorr`:

```
> VarCorr(model1)
Porker = pdLogChol(1)
            Variance  StdDev
(Intercept) 1.2503870 1.1182070
Residual    0.4642305 0.6813447
```

Finally, `intervals` extract the confidence intervals for all parameters (both fixed and random):

```
> intervals(model1)
Approximate 95% confidence intervals

 Fixed effects:
                           lower      est.      upper
(Intercept)            6.2260738  7.010000  7.7939262
chillingtunnel        -0.3643639  0.212500  0.7893639
pHlow                 -2.6544724 -1.545833 -0.4371943
chillingtunnel:pHlow -0.6508088  0.165000  0.9808088

 Random Effects:
  Level: Porker
                     lower     est.    upper
sd((Intercept)) 0.7840528 1.118207 1.594774

 Within-group standard error:
    lower      est.     upper
0.5070156 0.6813447 0.9156139
```

Hypothesis testing. We use `anova` to test a model against another. Remember to fit the model with ML rather than REML, see Section 5.1. Here we test the additive model (no interaction `pH` and `chilling`) against the full model:

```
> model1.ML = lme(tender ~ chilling+pH+pH:chilling, random =~ 1 | Porker,
                  method = "ML")
> model2.ML = lme(tender ~ chilling + pH, random =~ 1 | Porker, method="ML")
> anova(model1.ML,model2.ML)
          Model df      AIC      BIC    logLik   Test   L.Ratio p-value
model1.ML     1  6 151.7097 162.9369 -69.85483
model2.ML     2  5 149.9008 159.2568 -69.95041 1 vs 2 0.1911661  0.6619
```

From the output we see that the maximum of the log-likelihood function is -69.85 in the model with interaction (`model1.LM`) and -69.95 in the additive model (`model2.LM`). This gives $LR = 0.19$ which, evaluated in the $\chi^2(1)$-distribution, amounts to a p-value of 0.66. Similarly we test for the effect of `chilling` ($p = 013$) and thereafter for the effect of `pH` ($p = 0.0048$):

```
> model3.ML = lme(tender ~ pH, random =~ 1 | Porker, method="ML")
> anova(model3.ML,model2.ML)
          Model df      AIC      BIC    logLik   Test  L.Ratio p-value
model3.ML     1  4 150.2197 157.7045 -71.10983
model2.ML     2  5 149.9008 159.2568 -69.95041 1 vs 2 2.318832  0.1278

> model4.ML = lme(tender ~ 1, random =~ 1 | Porker, method="ML")
> anova(model4.ML,model3.ML)
```

```
           Model df      AIC      BIC    logLik   Test  L.Ratio p-value
model4.ML      1  3 156.1908 161.8044 -75.09538
model3.ML      2  4 150.2197 157.7045 -71.10983 1 vs 2 7.971096  0.0048
```

Computation of p-value with parametric bootstrap (simulation). The above p-values are either well above or well below 5%, so we do not doubt the conclusions based on the χ^2-approximations. For later use let us show how to compute a more accurate approximation to the p-value, anyway.

Take the test of `model3.ML` against `model2.ML`, say, where we got the value 2.319 of LR. First, `simulate.lme` is used to to simulate 1000 datasets from the null model, using the estimates from the real dataset. In our case the null model corresponds to `model3.ML`. For each simulated dataset, the null as well as the alternative model, here given by `model2.ML` are fitted. `R` saves the maximum values of the log-likelihood function for each simulated dataset in a list. This takes roghly half a minute (depending on the computer, of course). We plug out the relevant values, compute the LR test statistic in `lrsim` and finally compute the frequency of simulated LR-values that are larger than our observed value, 2.319. In this case out bootstrap p-value is 0.133, slightly larger than the approximate value 0.1278.

```
> sim = simulate.lme(model3.ML, m2=model2.ML, nsim=1000, method="ML")
> lrsim = 2*(sim$alt$ML - sim$null$ML)
> psim = sum(lrsim > 2.319)/1000
> psim
[1] 0.133
```

Analysis of the final model. We conclude that $y_i = \alpha(\texttt{pH}_i) + \eta(\texttt{Porker}_i) + e_i$ (`model3.ML`) is the final model. We fit the model with REML and extract the estimates:

```
> summary(model3)
Linear mixed-effects model fit by REML

Random effects:
 Formula: ~1 | Porker
        (Intercept)  Residual
StdDev:    1.116364 0.6873579

Fixed effects: tender ~ pH
                Value Std.Error DF  t-value p-value
(Intercept)  7.116250 0.3514849 24 20.24624  0.0000
pHlow       -1.463333 0.4970748 22 -2.94389  0.0075
```

As for linear models, we may of course reparameterize the model in order to obtain estimates of other conrasts. Moreover, the `estimable`-function works for `lme`-objects, too.

5.2.2 Two or more random factors

Two nested random factors. Consider Example 7.2 from Bibby et al. [2006] on concentration of vitamin E in meat samples. The dataset contains three variables, `Lab`, `Sample` and `E.vit`. In the example the squareroot of `E.vit` is used as response, so we construct a new variable with those values. Moreover, we construct factors of the original numerical variables:

```
> sqrtEvit = sqrt(E.vit)
> L = factor(Lab)
> S = factor(Sample)
```

The start model has `S` as fixed factor, and `L` and `L:S` as random factors. `lme` will not accept product factors with : in the random part, so the product factor is constructed and called `LS` before the `lme`-call. Naturally, `L` is coarser then `LS`; this is indicated to `R` with `L/LS` in the random part of the model specification.

```
> LS = L:S
> library(nlme)
> model1 = lme(sqrtEvit ~ S, random = ~ 1| L/LS)
```

It turns out that the effect of `S` is indeed significant (try it!) so the model cannot be reduced. In order to get the estimates from page 126 in Bibby et al. [2006] we fit the model without intercept:

```
> model1a = lme(sqrtEvit ~ S - 1, random = ~ 1| L/LS)
> summary(model1a)
Linear mixed-effects model fit by REML

Random effects:
 Formula: ~1 | L
        (Intercept)
StdDev:   0.2452144

 Formula: ~1 | LS %in% L
        (Intercept)  Residual
StdDev:   0.1347001 0.0677408

Fixed effects: sqrtEvit ~ S - 1
      Value Std.Error DF   t-value p-value
S1 1.177684 0.1269398 16  9.277495       0
S2 2.541707 0.1269398 16 20.022923       0
S3 2.131801 0.1269398 16 16.793791       0
S4 1.777835 0.1269398 16 14.005333       0
S5 2.642682 0.1269398 16 20.818383       0

> VarCorr(model1a)
```

```
             Variance     StdDev
L =          pdLogChol(1)
(Intercept) 0.060130084  0.2452144
LS =         pdLogChol(1)
(Intercept) 0.018144124  0.1347001
Residual    0.004588815  0.0677408
```

The general case, analysis with `lmer`. More generally, the random factors may be non-nested and there may be more than two random factors. `lme` can handle such cases, too, but it is not much fun; actually, the syntax is quite complicated. In such cases it is much easier to use the `lmer`-function from the `lme4`-package. It is of course also applicable in the simple situations, but note that there is no such function as `simulata.lmer` for parametric bootstrap of the p-value and that the `estimable`-function does not seem to work, either.

Let us consider the Example 7.5 from Bibby et al. [2006] on sweetness of chocolate. The dataset contains four variables: `product`, `session`, `assessor` and sweetness score `score`. First, the explanatory variables are made factors and the score is transformed with arcsin as suggested in the example.

```
> y = asin(sqrt(score/15))
> A = factor(assessor)
> P = factor(product)
> S = factor(session)
```

Then consider model (7.14) from Bibby et al. [2006] with `P` as fixed effect and `A`, `S`, `A` $\times$ `P`, `P` $\times$ `S` and `A` $\times$ `S` as random factors. The syntax for the fixed part of the model is the usual. The random factors are given as `(1|random.factor)`:

```
> library(lme4)
> model1 = lmer(y~P + (1|A) + (1|S)+(1|A:P)+(1|P:S)+(1|A:S), method="REML")
Warning message:
Estimated variance for factors 'P:S', 'S' is effectively zero
in: `LMEoptimize<-`(`*tmp*`, value = list(maxIter = 200, tolerance = 1.49011
611938477e-08,
> summary(model1)
Linear mixed-effects model fit by REML
Formula: y ~ P + (1 | A) + (1 | S) + (1 | A:P) + (1 | P:S) + (1 | A:S)
  AIC   BIC logLik MLdeviance REMLdeviance
 92.1 122.9 -36.05      60.05         72.1
Random effects:
 Groups   Name        Variance   Std.Dev.
 A:P      (Intercept) 7.2735e-02 2.6969e-01
 A:S      (Intercept) 7.6564e-03 8.7501e-02
 P:S      (Intercept) 2.2309e-11 4.7232e-06
 A        (Intercept) 6.6916e-02 2.5868e-01
```

```
 S        (Intercept) 2.2309e-11 4.7232e-06
 Residual             4.4618e-02 2.1123e-01
number of obs: 160, groups: A:P, 40; A:S, 32; P:S, 20; A, 8; S, 4

Fixed effects:
            Estimate Std. Error t value
(Intercept)   0.8533     0.1382   6.176
P2           -0.3469     0.1448  -2.396
P3           -0.3124     0.1448  -2.157
P4            0.1418     0.1448   0.979
P5           -0.3581     0.1448  -2.473
```

R writes a warning to us that two of the variance components are extremely close to zero. This is also seen from the summary-output: the estimates for σ^2_{PS} and σ^2_{A} are 10^{-11}! Hence, we fit the model without these two factors (model2 and model2.ML below). We are interested in the effect of product, so we carry out a likelihood ratio test for the effect. Remember that we should use the ML-method for estimation.

```
> model2 = lmer(y~P + (1|A) + (1|A:P) + (1|A:S))
> model2.ML = lmer(y~P + (1|A) + (1|A:P) + (1|A:S), method="ML")
> model3.ML = lmer(y~1 + (1|A) + (1|A:P) + (1|A:S), method="ML")
> anova(model3.ML, model2.ML)
Data:
Models:
model3.ML: y ~ 1 + (1 | A) + (1 | A:P) + (1 | A:S)
model2.ML: y ~ P + (1 | A) + (1 | A:P) + (1 | A:S)
          Df     AIC     BIC  logLik  Chisq Chi Df Pr(>Chisq)
model3.ML  4  85.237  97.537 -38.618
model2.ML  8  75.793 100.394 -29.896 17.444      4   0.001584 **
```

The model without product effect is rejected so model2 is the final model. The estimates are extracted with summary:

```
> summary(model2)
Linear mixed-effects model fit by REML

Random effects:
 Groups   Name        Variance  Std.Dev.
 A:P      (Intercept) 0.0729008 0.270001
 A:S      (Intercept) 0.0076823 0.087649
 A        (Intercept) 0.0664757 0.257829
 Residual             0.0445939 0.211173
number of obs: 160, groups: A:P, 40; A:S, 32; A, 8

Fixed effects:
            Estimate Std. Error t value
(Intercept)   0.8533     0.1380   6.182
```

```
P2          -0.3469     0.1450  -2.393
P3          -0.3124     0.1450  -2.155
P4           0.1418     0.1450   0.979
P5          -0.3581     0.1450  -2.470
```

Note that no p-values are associated with the t-values. This is because, in general, these statistics do not vary according to a t-distrution. Still, the t-values gives an idea about the significance of the parameters. For the same reason there is no function like `intervals` applicable for `lmer`-objects. Note also that `estimable` is not immediately applicable for `lmer`-objects.

If we wanted to reproduce the estimates Bibby et al. [2006] we should fit the model without intercept:

```
> model2a = lmer(y~P-1 + (1|A) + (1|A:P) + (1|A:S))
> summary(model2a)
Linear mixed-effects model fit by REML

Random effects:
 Groups   Name        Variance Std.Dev.
 A:P      (Intercept) 0.072805 0.269824
 A:S      (Intercept) 0.007667 0.087562
 A        (Intercept) 0.066751 0.258362
 Residual             0.044607 0.211204

Fixed effects:
    Estimate Std. Error  DF t value  Pr(>|t|)
P1   0.85332    0.13812 155  6.1779 5.477e-09 ***
P2   0.50639    0.13812 155  3.6662 0.0003377 ***
P3   0.54090    0.13812 155  3.9161 0.0001346 ***
P4   0.99516    0.13812 155  7.2049 2.377e-11 ***
P5   0.49522    0.13812 155  3.5854 0.0004505 ***
```

5.3 F-tests for balanced data (`aov`)

As mentioned in Section 5.1 exact F-tests can be carried out when the design is suitably balanced. These F-tests are performed with `aov`. Note, however, that `aov` does not check whether the design is balanced or not and that the analysis is wrong if this is not the case. Hence, if you are in doubt then use the general methos with `lme` or `lmer`.

The tenderness of pork chops data. Consider again the dataset from Section 5.2.1 which is indeed balanced. First, the model with interaction between `pH` and `chilling` is fitted, and the test for interaction is extracted with `summary`:

```
> model1.aov = aov(tender ~ chilling + pH + pH:chilling + Error(Porker))
> summary(model1.aov)

Error: Porker
          Df Sum Sq Mean Sq F value   Pr(>F)
pH         1 25.696  25.696  8.6665 0.007508 **
Residuals 22 65.230   2.965

Error: Within
            Df  Sum Sq Mean Sq F value Pr(>F)
chilling     1  1.0443  1.0443  2.2495 0.1479
chilling:pH  1  0.0817  0.0817  0.1759 0.6790
Residuals   22 10.2131  0.4642
```

Note that the `summary`-output is quite different for `aov`-objects than for `lm`- and `lme`-objects. An analysis of variance table is given for each stratum, that is, for each random factor. `Error: Within` corresponds to the residual stratum from which we see that the interaction between `pH` and `chilling` is not significant ($F = 0.1759$, $p = 0.68$), so we fit the model without interaction:

```
> model2.aov = aov(tender ~ chilling + pH + Error(Porker))
> summary(model2.aov)
Error: Porker
          Df Sum Sq Mean Sq F value   Pr(>F)
pH         1 25.696  25.696  8.6665 0.007508 **
Residuals 22 65.230   2.965

Error: Within
          Df  Sum Sq Mean Sq F value Pr(>F)
chilling   1  1.0443  1.0443  2.3331 0.1403
Residuals 23 10.2948  0.4476
```

As we know from the factor diagram (page 142 in Bibby et al. [2006] `pH` is in the `Porker` stratum and `chilling` is in the residual stratum. We see that `chilling` is not significant ($p = 0.14$) whereas `pH` is significant ($p = 0.0075$). Since the two effects are in different stratums we can use both tests without re-estimating the model.

The conclusion from the `aov`-analysis is the same as in the `lme`-analysis from Section 5.2.1. The p-values are close, but not exactly the same. This is because the discrepancy between two models is measured in differently in the two approaches (and furhermore because the p-value for the likelihood ratio test is only approximate).

The Vitamin E data. Let us briefly re-visit the Vitamin E data from Section 5.2.2. The `aov`-fit looks as follows:

```
> model1.aov = aov(sqrtEvit ~ S + Error(L+L:S))
> summary(model1.aov)
```

```
Error: L
          Df  Sum Sq Mean Sq F value Pr(>F)
Residuals  4 2.56871 0.64218

Error: L:S
          Df  Sum Sq Mean Sq F value    Pr(>F)
S          4 14.3465  3.5866  87.742 1.130e-10 ***
Residuals 16  0.6540  0.0409

Error: Within
          Df   Sum Sq  Mean Sq F value Pr(>F)
Residuals 25 0.114720 0.004589
```

Again, the conclusion is the same as in the `lme`-analysis: there is a highly significant effect of `S` (sample).

6

Repeated measurements

This chapter is about the analysis of repeated measurements. Throughout the chapter we will use the data from Example 10.1 in Bibby et al. [2006] on the growth of goats.

6.1 Preliminaries

6.1.1 The dataset

Suppose that the dataset `goatdata` is available:

```
> goatdata
    goat feed   w0 day weight
1      1    1 20.4   0   20.4
2      1    1 20.4  26   21.0
3      1    1 20.4  45   21.5
4      1    1 20.4  61   21.3
5      1    1 20.4  91   22.3
6      2    1 10.3   0   10.3
7      2    1 10.3  26   11.4
.
.
139   28    4 11.3  61   10.5
140   28    4 11.3  91   11.0
```

The dataset has five variables, each with 140 observations: `goat`, `feed`, `day`, `w0`. Note that `w0`, the weight at the beginning of the experiment, is the same for all measurements for each goat (as it should be). The variables `goat`, `feed` and (sometimes) `day` are to be used as factors. We construct the factors "inside" the dataset — this is useful as we later use subsets of the dataset:

```
> attach(goatdata)
goatdata$feedfac = factor(feed)
goatdata$goatfac = factor(goat)
goatdata$dayfac = factor(day)
```

6.1.2 Profile plots

In order to get an overview of the data it is quite important to make some plots of repeated measurements data. It is recommended to make *subject profiles* (individual profiles, here one per goat) as well as *average profiles* (one per treatment, average over goats). The subject profiles give an impression of the "typical" time-response relationship whereas the average profiles illustrate, among others, potential interactions between treatment and time.

Subject profiles. In order not to let the variation of weights at day 0 blur the picture too much, we choose to plot weight increments from day 0, rather than the actual weights. For the subject profiles we plot the increment against day for each goat, either in the same plot or in one plot per treatment. Such a plot is most easily made with the `interaction.plot` function:

```
> interaction.plot(day,goat,weight-w0)
```

For this particular dataset there is not the same time distance between any two subsequent measurements, and the above plot is therefore not very useful for evaluating the time vs. weight increment relationship. In order to make the time axis "correct", we need to construct the plot manually, and the following program lines produce the left part of Figure 6.1:

```
> plot(day,weight-w0, type="n")
> for (i in 1:28) points(day[goat==i], weight[goat==i]-w0[goat==i], type="l",
                         col=feed[goat==i][1])
```

Some explanation may be needed here: the `plot`-command produces an "empty plot", due to `type=''n''`. By an empty plot we mean a plot with axes and labels but no points or lines or similar. Then, for each goat, the subject profile is added to this plot. The `points` works like `plot`, except that it adds something to an existing plot instead of making a new one.

For the i'th goat the increments are computed an plotted. The points are joined due to `type="l"`. The colours are black (`col=1`), red (`col=2`), green (`col=3`), blue (`col=4`) for treatments 1, 2, 3 and 4, respectively. The colouring is quite useful — unless of course you are printing without colours. In that case substitute `col` by `lty` in order to obtain different line types for the treatments.

Note that we have included day zero in the plot although we intend to use the measurement from day 0 as a covariate rather than as part of the response vector. This is a matter of taste.

Average treatment profiles. The average profiles are also illuminating. They illustrate treatment differences, both at separate time points and over time, the latter corresponding to interaction between day and treatment. Again, `interaction.plot` would do the job were the measurements equally spaced:

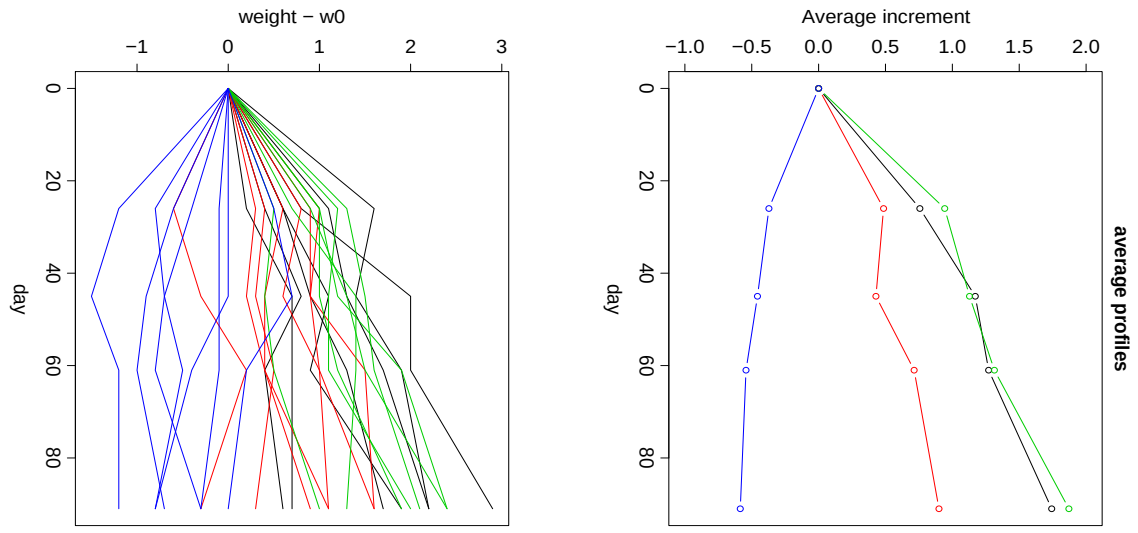

Figure 6.1: Subject and profile plots for the goats data.

```
> interaction.plot(day,feed,weight-w0)
```

If we want the time axis correct, as in the right panel of Figure 6.1, we can use fhe following program lines:

```
> ave = matrix(0,4,5)
> for (i in 1:4)
    {
     mean0 = mean(weight[feed == i & day==0])
     ave[i,1] = 0
     ave[i,2] = mean(weight[feed==i & day==26]) - mean0
     ave[i,3] = mean(weight[feed==i & day==45]) - mean0
     ave[i,4] = mean(weight[feed==i & day==61]) - mean0
     ave[i,5] = mean(weight[feed==i & day==91]) - mean0
    }
> plot(c(0,26,45,61,91),ave[1,],type="n", ylim=c(-1,2),
     xlab="day", ylab="Average increment",)
> for (i in 1:4) points(c(0,26,45,61,91),ave[i,],type="b",col=i)
```

First, a 4 by 5 matrix with all zeros is constructed for the average values, one row per treatment and one column per day (including day zero). Then, for each treatment (the `for`-loop) `mean0` is just the baseline weight, and the average increment for each day is computed and saved in the matrix. Then an empty plot is made with `plot` (due to `type="n"`). Note that the axes and labels on the axes are defined here. Finally, the average profiles are added to the plot with `points`.

6.2 Analysis of summary measures

In Bibby et al. [2006] the increment in weight from day 26 to day 91 is used a summary measure. Alternatively, one could have used the increment from day zero to day 91, or some other summary measure, but let us follow the analysis from the example. Hence, we need to construct a variable with these increments as well as explanatory variables (feed and the day zero measurement) of the right length, one value per day. The new variables are called `myincr`, `myfacfeed` and `myw0`. There are many ways to construct these variables so, here is one such way:

```
> goat91 = subset(goatdata, day==91)
> goat26 = subset(goatdata, day==26)

> myincr = goat91$weight - goat26$weight
> myfeedfac = goat91$feedfac
> myw0 = goat91$w0
```

The subdatasets `goat26` and `goat91` contain only measurements from days 26 and 91, respectively. The increments are computed from the weights in these two datasets, and the explanatory variables are taken from one of them. It is always a good idea to check that the variables are as we want them: correctly:

```
> myincr
 [1]  1.3  1.1  2.1  1.1  0.6  0.2  0.5  0.5  0.6  0.3  0.6  0.7 -0.1  0.3  0.6
[16]  0.9  0.8  0.8  0.5  1.5  1.4  0.0 -0.2 -0.5 -0.1 -0.4 -0.8  0.5
> myfeedfac
 [1] 1 1 1 1 1 1 1 2 2 2 2 2 2 2 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4
Levels: 1 2 3 4
> myw0
 [1] 20.4 10.3 12.5 10.8 13.6 19.0 10.5 11.2 17.0 19.2  7.0 11.0 11.6 10.0 11.2
[16] 14.7 10.7 10.6 20.5 18.0  8.1 18.0  9.3  8.8 16.9 10.5  9.3 11.3
```

After having defined the relevant variables, `lm` is used for the analysis as usual. For example, the model with feed type and the day zero weight as explanatory variables is fitted like this:

```
> modelA = lm(myincr ~ myw0 + myfeedfac)
```

This model may then be used for analysis in the usual way (do it!).

6.3 Analysis of the complete dataset

In the following we will use the complete dataset. We use the weights from days 26, 45, 61 and 91 as responses and the weight at day zero as an explanatory variable. Hence, the day zero weights should be excluded from the response vector. That is, we use the data set `goatdata1`:

```
> goatdata1 = subset(goatdata, day>0)

> goatdata1
  goat feed   w0 day weight feedfac goatfac dayfac
2    1    1 20.4  26   21.0       1       1     26
3    1    1 20.4  45   21.5       1       1     45
4    1    1 20.4  61   21.3       1       1     61
5    1    1 20.4  91   22.3       1       1     91
7    2    1 10.3  26   11.4       1       2     26
8    2    1 10.3  45   11.6       1       2     45
.
.
139   28    4 11.3  61   10.5       4      28     61
140   28    4 11.3  91   11.0       4      28     91
```

6.3.1 The random intercepts model

The random intercepts model uses the complete dataset and has `goat` as a random factor. Hence, the `lme`-function is used for the analysis. For example, the model with interaction between `feedfac` and `dayfac` and baseline measurements (`w0`) is fitted with

```
> library(nlme)
> modelC = lme(weight~w0 + feedfac*dayfac, random=~1|goat)
```

This model may then be used for analysis in the usual way (do it!).

6.3.2 Models with seriel correlation structure

Fit of the Diggle model. Models with seriel correlation structure are fitted by including a `corr`-statement in the `lme`-call, for example as follows:

```
> modelF = lme(weight ~ w0 + feedfac*dayfac,
               random = ~ 1 | goat,
               method = "REML",
               corr = corGaus(form = ~ day | goat, nugget=T))
```

`corGaus` gives the correlation structure from the Diggle model (there are other possibilities, see below): `goat` identifies goats as the experimental units on which repeated measurements are taken, `day` is indeed the day variable in this dataset, and `nugget=T` includes the measurement error variance σ^2 (which is set to zero otherwise).

Estimation. Estimates for the parameters (both for fixed and random effects) are obtained by `summary`. In this case `summary` gives a lot of output; we only show what is relevant at the moment:

```
> summary(modelF)
Linear mixed-effects model fit by REML
 Data: NULL
       AIC      BIC    logLik
  140.7827 194.4141 -49.39134

Random effects:
 Formula: ~1 | goat
        (Intercept)  Residual
StdDev:    0.399672 0.3026485

Correlation Structure: Gaussian spatial correlation
 Formula: ~day | goat
 Parameter estimate(s):
     range     nugget
36.3141567  0.3420622

Fixed effects: weight ~ w0 + feedfac * dayfac
                       Value Std.Error DF  t-value p-value
(Intercept)        1.5468990 0.3577340 72  4.32416  0.0000
w0                 0.9430660 0.0218743 23 43.11289  0.0000
feedfac2          -0.3535762 0.2698255 23 -1.31039  0.2030
feedfac3           0.1588740 0.2681716 23  0.59243  0.5593
feedfac4          -1.2343060 0.2710350 23 -4.55405  0.0001
dayfac45           0.4142857 0.1143474 72  3.62304  0.0005
dayfac61           0.5142857 0.1391742 72  3.69527  0.0004
dayfac91           0.9857143 0.1595969 72  6.17627  0.0000
feedfac2:dayfac45 -0.4714286 0.1617116 72 -2.91524  0.0047
feedfac3:dayfac45 -0.2285714 0.1617116 72 -1.41345  0.1618
feedfac4:dayfac45 -0.5000000 0.1617116 72 -3.09192  0.0028
feedfac2:dayfac61 -0.2857143 0.1968221 72 -1.45164  0.1509
feedfac3:dayfac61 -0.1428571 0.1968221 72 -0.72582  0.4703
feedfac4:dayfac61 -0.6857143 0.1968221 72 -3.48393  0.0008
feedfac2:dayfac91 -0.5714286 0.2257041 72 -2.53176  0.0135
feedfac3:dayfac91 -0.0571429 0.2257041 72 -0.25318  0.8009
feedfac4:dayfac91 -1.2000000 0.2257041 72 -5.31670  0.0000
```

The interpretation of the fixed effects parameters is the usual one, but some explanation for the random effects parameters is needed: The square of the intercept standard deviation (0.3997^2) is the estimate of ν^2. The square of the residual standard deviation (0.3026^2) is the estimate of $\sigma^2 + \tau^2$. The `range` parameter estimate (36.31) is the estimate of ϕ. The nugget parameter estimate (0.3421) is the estimate of $\sigma^2/(\sigma^2 + \tau^2)$. Hence, we have the equations:

$$\hat{\nu}^2 = 0.3997^2 = 0.1598, \quad \hat{\sigma}^2 + \hat{\tau}^2 = 0.3026^2 = 0.0916, \quad \hat{\phi} = 36.31, \quad \frac{\hat{\sigma}^2}{\hat{\tau}^2 + \hat{\sigma}^2} = 0.3421$$

Solving for the original parameters, we find

$$\hat{\nu}^2 = 0.1598, \quad \hat{\tau}^2 = 0.0603, \quad \hat{\phi} = 36.31, \quad \hat{\sigma}^2 = 0.0313.$$

These REML-estimates are not identical to those on page 192 in Bibby et al. [2006] which are ML-estimates.

Semi-variogram. A semi-variogram for the model is quite easily made in `R`. The following command produces the plot in Figure 6.2:

```
> modelF1 = lme(weight ~ w0 + feedfac*dayfac,
                random = ~ 1 | goat,
                method = "ML",
                corr = corGaus(form = ~ day | goat, nugget=T))

> plot(Variogram(modelF1, form= ~ day))
```

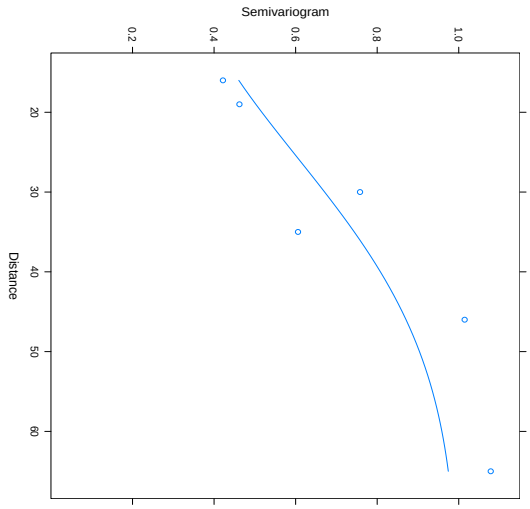

Figure 6.2: Semi-variogram for the Diggle model applied to the goats data.

Note that `R` (unfortunately) scales the y-axis such that the model-based estimate (the curve) approaches one as the day distance gets very large. Hence, the plot cannot be used for finding preliminary estimates for the variance parameters as described in the Bibby et al. [2006].

Comparing different correlation structures. Sometimes we are interested in comparing different correlation structures in order to choose the model in which we will carry out the analysis of the fixed effects (which is what we are really interested in).

Since the random intercepts model (`modelC`) is a special case of the Diggle model (`modelF`) it is, in principle, possible to carry out a test for model reduction from `modelF` to `modelC`). Unfortunately the likelihood ratio test statistic is not χ^2-distributed, however,

not even for a very large sample size, and the `simulate.lme`-function does not apply to models with a seriel correlation structure. Hence, there is no easy way to obtain a reliable p-value.

Instead, such models are often compared through the AIC-values For a given model the AIC value measures, in a certain sense, how well does the model fit the data, taking into account the number of parameters in the model (the complexity of the model). The comparison is easily performend in `R`: fit the models with the exact same fixed effects (and REML), use `anova`, and choose the model with the lowest value of AIC:

```
> anova(modelC,modelF)
       Model df      AIC      BIC    logLik   Test  L.Ratio p-value
modelC     1 19 143.6393 192.1630 -52.81966
modelF     2 21 140.7827 194.4141 -49.39134 1 vs 2 6.856638  0.0324
```

Here the AIC value is lower for the Diggle models than for the random intercepts model, suggesting that the Diggle model is the better one. Unfortunately there are not really any guidelines as to when a difference in AIC is large enough to justify that a model is really more apprpriate thah another, so the AIC analysis should be used a a guidelins only.

Reduction of random part of the model. Somedays we are interested in testing hypotheses in the random part of the model. Such tests are usually carried out before attempts are made to reduce the fixed effects structure of the model.

When testing hypotheses in the *random effects part* of the model it is generally recommended to use REML-estimation, that is, fit the two models with REML and compare their restricted log-likelihoods. This is not true for hypotheses in the fixed effects part of the model where is is absolutely essential that ML-estimation is used. For the same reason, when we use the restricted log-likelihoods to compare two models with different variance structure, it is very important that they have the same fixed effects.

In this case we can test the random intercepts model against the Diggle model. That is, test whether the correlation over all day distances is the same or actually decreases as the day distance increases. This amounts to testing `modelC` from Section 6.3.1 against the above `modelF`. Both are fitted with REML, and we use `anova` to carry out the test:

```
> anova(modelC,modelF)
       Model df      AIC      BIC    logLik   Test  L.Ratio p-value
modelC     1 19 143.6393 192.1630 -52.81966
modelF     2 21 140.7827 194.4141 -49.39134 1 vs 2 6.856638  0.0324
```

Unfortunately, the `simulate.lme`-function does not apply to models with a seriel correlation structure, so there is no easy way to get a more precise approximation to the p-value than the above χ^2-approximation. In any case, the test indicates that the Diggle model is more appropriate for the data than the random intercepts model. Comparing with the semi-variogram in Figure 6.2, this is perhaps not too surprising as the random intercepts model corresponds to a horizontal line which does not seem to fit very

well with the empirically fitted points. Hence, we continue the analysis with the Diggle model.

Reduction of fixed part of the model. Hypotheses about the fixed effects are tested as usual: first both the "full" model as well as the model under the hypothesis are fitted with ML; then they are compared with `anova`.

Consider for example the hypothesis that the time-response relationship is linear. First `modelF` with baseline measurements and the interaction between treatment and the day factor is fitted, this day with ML. Then a corresponding model, but now with day as a covariate, is fitted. This corresponds to a linear day-weight relationship between time and weight with slope-dependent intercepts and slopes.

```
> anova(modelG1, modelF1)
        Model df      AIC      BIC    logLik   Test  L.Ratio p-value
modelG1     1 13 89.71611 125.0566 -31.85805
modelF1     2 21 96.85771 153.9462 -27.42886 1 vs 2 8.858394  0.3544
```

The hypothesis is clearly not rejected. Hence, it makes sense to test hypotheses about the dependence of treatment on intercepts and slopes for the regression lines (do it!).

Other correlation structures. Above we have considered the random intercepts model and the Diggle model, but there are many other models for the correlation structure. For example one where the correlation decreases exponentially with the time distance rather than with the squared time distance (`modelI`) or the unstructured model where the correlation depends on the time distance only, with no further restrictions (`modelJ`):

```
> modelI = lme(weight ~ w0 + feedfac + dayfac + feedfac:dayfac,
               random = ~ 1 | goat,
               method = "REML",
               corr = corSymm(form = ~ 1 | goat),
               data=goatdata1)

> modelJ = lme(weight ~ w0 + feedfac + dayfac + feedfac:dayfac,
               random = ~ 1 | goat,
               method = "REML",
               corr = corExp(form = ~ day | goat, nugget=T))
```

An AIC comparison shows that there is not much difference in the ability to describe the data of the Diggle model and the exponential model:

```
> anova(modelJ, modelF,modelI)
       Model df      AIC      BIC    logLik   Test   L.Ratio p-value
modelJ     1 21 140.7574 194.3888 -49.37872
modelF     2 21 140.7827 194.4141 -49.39134
modelI     3 25 148.5294 212.3763 -49.26468 2 vs 3 0.2533164  0.9926
```

A

Installation of R

To install R under Microsoft Windows do as follows:

1. Go to `http://mirrors.sunsite.dk/cran/`
2. Click on **Windows (95 and later)** under Precompiled Binary Distributions.
3. Click on **base**.
4. Click on **R-2.3.1-win32.exe** and save the file (e.g. on the desktop).
5. When the file has been downloaded then double click on `R-2.3.1-win32`, accept the license and install the program.

B

Add-on packages

The base package of `R` contains most of the functions that we need, such as `lm`, `anova`, `summary` *etc.* In these notes we have, however, also used functions from add-on packages. There is a large number of such `R`-packages available which are not automatically installed with the base package. A package is simply a collection of `R`-functions. For example, we have used `lme` from the `nlme`-package and `estimable` from the `gmodels`-package.

To access the functions from a package, the package should once and for all be installed on your local computer. On a computer with internet connection, click "Packages" in the R menu and you get the option "Install package(s) from CRAN" which lists the possible packages. Click the wanted package and it is installed! This only needs to be carried out once on your computer. To actually use the functions from the add-on package (and for the help information to be visible) the package must also be loaded, either via the "Packages" menu or by the `library`-command

```
> library(packagename)
```

where `packagename` is the name of the `R`-package. You need to do this every time `R` is re-started.

Some packages, for example `gmodels` and `nlme`, are already installed with the base package, but still need to be loaded in each `R` -session. Hence, all you need to do for these to packages, is to write

```
> library(gmodels)
> library(nlme)
```

C

Getting help in R

To get help on how to use a certain function in R, for example `read.table`, you may simply write

```
> ?read.table
```

and R will open a window with some text about how to use the function. However, this only works when you know what function to use and remember its name. Quite often you remember that something can be done but have forgotten how. The *reference card* (Short, 2005) is very well suited for this purpose. If, for example, you have forgotten the name of the normal distribution function, you find it under the heading "Distributions" in the reference card. There is also, of course, a chance that you may find it in the present note.

To expand your knowledge in R it is usually better to use either a book on R, for example Dalgaard (2002), or one of the numerous R guides or manuals on the web. Thus, until you are experienced you may find "An Introduction to R" useful. To find it you select "R project home page" from the help menu, click on "Manuals" and then you may select to browse it (using the html version). For example, if you are looking for the normal distribution you may click on "Probability distributions" and you will jump to the right section.

Finally remember to share your knowledge with other R users.

Bibliography

Bo Martin Bibby, Torben Martinussen, and Ib Skovgaard. *Experimental Design in the Agricultural Sciences.* Samfundslitteratur, KVL-bogladen, 2006.

Peter Dalgaard. *Introductory Statistics with R.* Springer Verlag, New York, 2000.

Morten Larsen and Peter Sestoft. *Noter on R.* Department of Natural Sciences, KVL, 2005.

Torben Martinussen and Ib Skovgaard. *A note on R: Basic statistical analyses.* www.matfys.kvl.dk/stat/kurser/statdata1/Rnotat.pdf, 2006.

Ib Skovgaard. *Basal Biostatistik 2.* Samfundslitteratur, KVL-bogladen, 2000.

Ib Skovgaard, Henrik Stryhn, and Mats Rudemo. *Basal Biostatistik 1.* DSR Forlag, 1999.