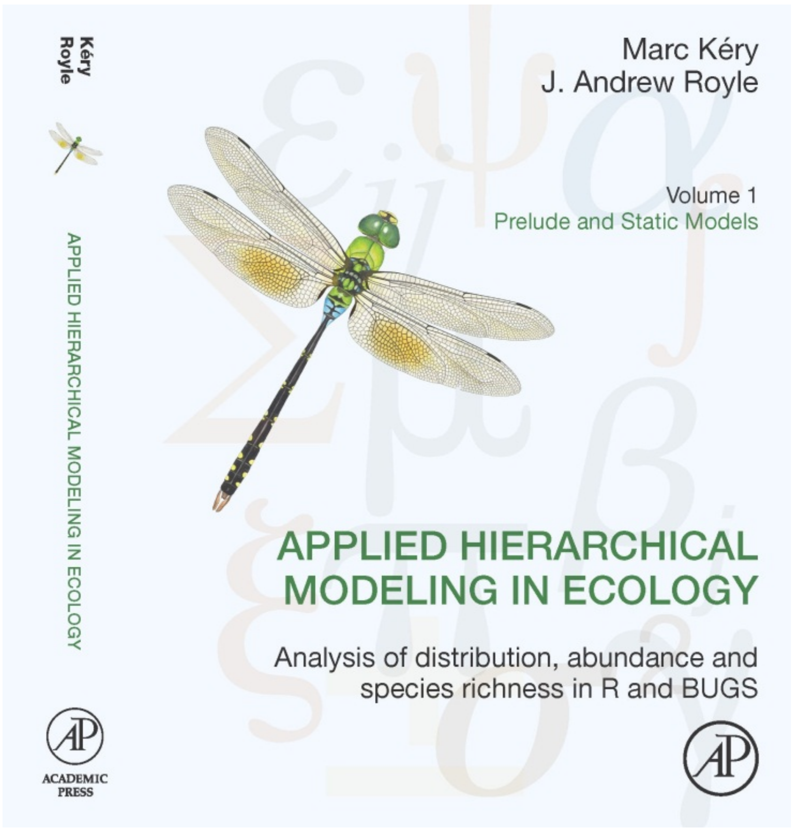
Applied hierarchical modeling in ecology

Modeling distribution, abundance and species richness using R and BUGS

Volume 1: Prelude and Static models,

This document contains part of Chapters 3–5

Marc Kéry & J. Andy Royle


Kéry
Royle
APPLIED HIERARCHICAL MODELING IN ECOLOGY
ACADEMIC PRESS
Marc Kéry
J. Andrew Royle
Volume 1
Prelude and Static Models
APPLIED HIERARCHICAL MODELING IN ECOLOGY
Analysis of distribution, abundance and species richness in R and BUGS

7 September 2015

Academic Press, 2015

Table of contents

3. Linear models, generalised linear models (GLMs) and random effects models: the components of hierarchical models

3.1 Introduction

In chapter 2, we introduced hierarchical models (HMs) as consisting of a nested sequence of probability models for observed and unobserved random variables. We saw that the former are observed data and the latter latent or partially latent variables, or random effects, such as the occurrence or abundance state of a local population, but that a random effect may also be some hypothetical "group effect". Typically, we model structure in each random variable using link functions and linear functions of covariates, i.e., as simple weighted sums of covariate values (where coefficients represent the weights), exactly as in generalised linear models; GLMs (McCullagh & Nelder, 1989). Therefore, HMs can be described as a compound GLM – a sequence of two or more related GLMs. In this book, we use the R package `unmarked` (Fiske and Chandler 2011) and BUGS software (Lunn et al. 2000; Plummer 2003) for almost all fitting of HMs. Much of the power of HMs derives from the flexible ways in which we specify linear models at each level in the HM, e.g., for occupancy and for detection in an occupancy model. Thus, it is essential that you understand very well the specification of a wide range of linear models, in order to be an effective hierarchical modeler.

The main focus of this chapter is the linear model. We show how different linear models can be described in algebra and in the model definition language in R, e.g., using functions `lm` and `glm`, which we assume you are familiar with. When fitting HMs with `unmarked`, you will specify linear models in exactly the same way, regardless of the type of model. We also show how to specify the same linear models in the BUGS programming language. In so doing, we get a little ahead of ourselves, because we don't formally introduce BUGS until chapter 5. However, seeing the algebraic model description and the R and BUGS model descriptions alongside is illuminating, although you may perhaps fully understand the BUGS model specification only after you have read chapter 5. Ecologists can find an excellent introduction to linear models in chapter 6 of Evan Cooch's gentle introduction to program MARK (www.phidot.org/software/mark/docs/book).

We also provide a brief and applied introduction of two further crucial concepts in applied statistics that are almost always found in HMs: generalised linear models (GLMs) and random effects. Strictly speaking, random effects are nothing new in HMs: they are simply the outcome of an *unobserved* random variable, as opposed to the data, which are the outcome of an *observed* random variable. The presence of an unobserved random variable is one defining feature of a hierarchical or mixed model. However, in our experience ecologists often find random effects or mixed models confusing, hence, we attempt to shed some light on this class of models here. We focus on GLMs and random effects mostly as they are relevant for your understanding of the types of HMs in this book. Our coverage of both topics is much more summary than for instance in chapters 3 and 4 of Kéry & Schaub (2012) or in text books such as McCullagh & Nelder (1989), Dobson & Barnett (2008) or Ntzoufras (2009) for GLMs, or in books like Pinheiro & Bates (2000), McCulloch & Searle (2001), Lee et al. (2006), Gelman & Hill (2007) or Littell et al. (2008) and in Bolker et al. (2009) for random effects/mixed models.

Most ecologists learn statistical modeling best by example, hence, we use for illustration a toy data set and imagine that we had measured wing span and body length of a total of nine blue-eyed hooktail dragonflies (*Onychogomphus uncatus*; Fig. 1) in three populations in the Spanish Pyrenees (Navarra, Aragon, Catalonia). For each individual we also assessed sex, color intensity (proportion of body that is yellow as opposed to black), ectoparasite load (number of mites counted) and whether each of the four wings (2 hind, 2 front) was damaged or not. Part of this toy data set, and analysis, is taken from chapters 3 and 4 in Kéry & Schaub (2012).

In section 3.2, we illustrate linear models by investigating the relationships between wing span (a numerical, metric response) and four explanatory variables: population, sex, body length and color intensity. The first two are factors with three (population: 1 = Navarra, 2 = Aragon, 3 = Catalonia) and two (sex: 1= male; 2 = female) levels, respectively. Factors are categorical explanatory variables in which numbers have no quantitative meaning; they are merely labels or group names. In contrast, body length and color intensity are continuous explanatory variables, or covariates, where numbers do have a quantitative meaning. For instance, we can compute a difference in body length between two dragonflies, but we cannot compute a difference between their sex. In this chapter, we will distinguish continuous and categorical explanatory variables by use of the terms *'covariates'* and *'factors'*. We show how their pairwise effects are combined in linear models in an additive or an interactive fashion. Of course, in your modeling you will typically have more than just two explanatory variables, but the principles are best explained in the simplest possible setting.

In section 3.3, we will use parasite load and wing damage as response variables to illustrate generalised linear models (GLMs). Finally, in section 3.4., we revisit some models from sections 3.2. and 3.3. and turn them into random-effects, or hierarchical, or generalized linear mixed models (GLMMs).

Fig. 3–1: Male blue-eyed hooktail (*Onychogomphus uncatus*), Aragon/Spain, 2013 (Photo: M. Kéry)

```
# Define data
pop <- factor(c(rep("Navarra", 3), rep("Aragon", 3), rep("Catalonia", 3)),
levels = c("Navarra", "Aragon", "Catalonia"))         # Population
wing <- c(10.5, 10.6, 11.0, 12.1, 11.7, 13.5, 11.4, 13.0, 12.9) # Wing span
body <- c(6.8, 8.3, 9.2, 6.9, 7.7, 8.9, 6.9, 8.2, 9.2) # Body length
sex <- factor(c("M","F","M","F","M","F","M","F","M"), levels = c("M", "F"))
mites <- c(0, 3, 2, 1, 0, 7, 0, 9, 6)      # Number of ectoparasites
color <- c(0.45, 0.47, 0.54, 0.42, 0.54, 0.46, 0.49, 0.42, 0.57) # Color
intensity
damage <- c(0,2,0,0,4,2,1,0,1)                  # Number of wings damaged

cbind(pop, sex, wing, body, mites, color, damage) # Look at data
      pop sex wing body mites color damage
 [1,]   1   1 10.5  6.8     0  0.45      0
 [2,]    1    2 10.6  8.3     3  0.47       2
 [3,]    1    1 11.0  9.2     2  0.54       0
 [4,]    2    2 12.1  6.9     1  0.42       0
 [5,]    2    1 11.7  7.7     0  0.54       4
 [6,]    2    2 13.5  8.9     7  0.46       2
 [7,]    3    1 11.4  6.9     0  0.49       1
 [8,]    3    2 13.0  8.2     9  0.42       0
 [9,]    3    1 12.9  9.2     6  0.57       1
```

Internally in R, factor levels for `pop` and `sex` are expressed as integers ranging from 1 to the number of levels, but we can clearly see that R interprets these numbers as factor levels (i.e., names):

```
str(pop)
 Factor w/ 3 levels "Navarra","Aragon",..: 1 1 1 2 2 2 3 3 3
```

We plot the data for wing span, parasite load and damage (Fig. 2).

```
par(mfrow = c(1, 3), cex = 1.2)
colorM <- c("red", "red", "blue", "green", "green")  # Pop color code males
colorF <- c("red", "blue", "blue", "green", "green") # Pop color code females
plot(body[sex == "M"], wing[sex == "M"], col =colorM, xlim = c(6.5, 9.5), ylim =
c(10, 14), lwd = 2, frame.plot = FALSE, las = 1, pch = 17, xlab = "Body length",
ylab = "Wing span")
points(body[sex == "F"], wing[sex == "F"], col =colorF, pch = 16)
text(6.8, 13.8, "A", cex = 1.5)
plot(body[sex == "M"], mites[sex == "M"], col = colorM, xlim = c(6.5, 9.5), ylim
= c(0, 10), lwd = 2, frame.plot = FALSE, las = 1, pch = 17, xlab = "Body
length", ylab = "Parasite load")
points(body[sex == "F"], mites[sex == "F"], col = colorF, pch = 16)
text(6.8, 9.7, "B", cex = 1.5)
plot(body[sex == "M"], damage[sex == "M"], col = colorM, xlim = c(6.5, 9.5),
ylim = c(0, 4), lwd = 2, frame.plot = FALSE, las = 1, pch = 17, xlab = "Body
length", ylab = "Damaged wings")
points(body[sex == "F"], damage[sex == "F"], col = colorF, pch = 16)
text(6.8, 3.9, "C", cex = 1.5)
```

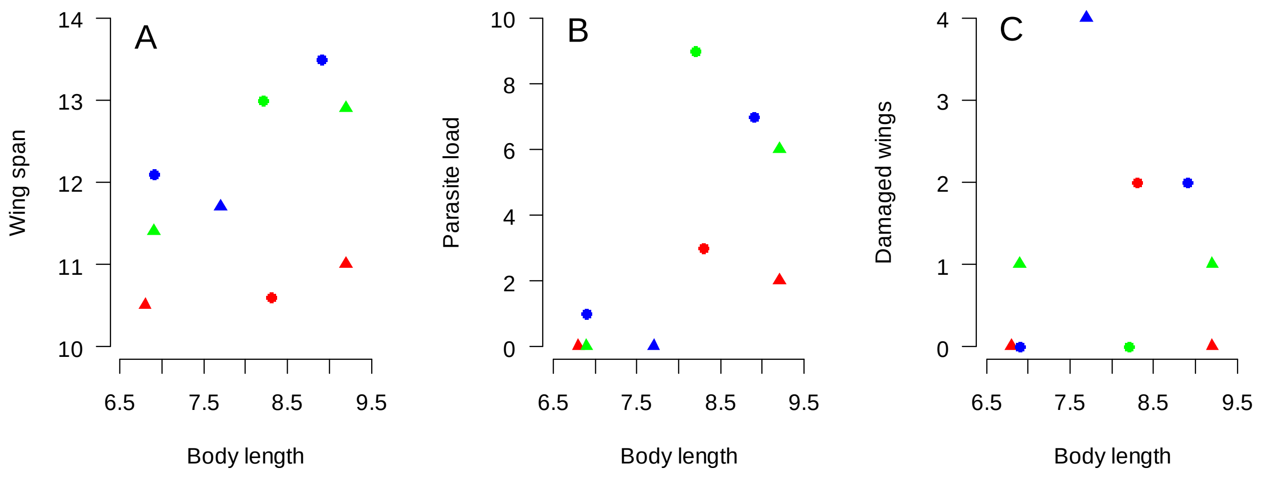


Fig. 3–2: Relationships between wing span (A), parasite load (B), and number of damaged wings (out of four; C), respectively, and body length, sex and population in the dragonfly toy data set. Colors code for the three populations (Navarra red, Aragon green and Catalonia blue); circles denote females and triangles males.

3.2 Linear models

In a linear model, the effects of covariates or factors on a response act in a purely additive fashion, i.e., they are expressed as simple weighted sums of the values of the covariates, with the weights being the coefficients or parameters of the linear model. The model is said to be linear in the parameters, but it does not necessarily represent a straight line when shown as a graph. For instance, models with polynomial, such as quadratic or cubic, terms do not translate into a straight-line graph, but they are linear models. A linear model can be described in a variety of ways:

(1) in words, e.g., when we say something like "population and body length act additively on wing span",

(2) using specific names or labels for least-squares techniques which *imply* a certain linear model, e.g., when we say "t-test", "linear regression", "ANOVA" or "ANCOVA" (Mead, 1988),

(3) in graphs, e.g., using line and bar plots,

(4) in algebra, e.g., $y_i \sim Normal(\alpha_{j(i)} + \beta * x_i, \sigma^2)$,

(5) using matrix algebra, e.g., $\mathbf{y} = \mathbf{X}\beta + \varepsilon$,

(6) as a system of equations,

(7) in the R model definition language, e.g., `lm(wing ~ pop + length)`,

(8) and finally, using the BUGS model definition language,
e.g.,
```
y[i] ~ dnorm(mu[i], tau)
mu[i] <- alpha[pop[i]] + beta * length[i].
```

In statistical modeling, most of us have become accustomed to defining certain types of linear models using point-and-click techniques available in many statistical packages, or using a slick model definition language such as the one in the R functions `lm` and `glm` or in all functions in package `unmarked`. Model definition languages make your life easier because they allow you to define a large range of linear models quickly and error-free. The big drawback is that it is easy to fit linear models without actually understanding them or knowing what their parameters mean. In addition, you may not be able to fit certain non-standard linear models, or only in a very complicated way. In contrast, when fitting models in BUGS, no such shortcuts to defining linear models are available. Hence, you must know exactly what kind of linear model you want to fit and in what parameterisation (see below). This is one of the main stumbling blocks for beginners in BUGS. On the other hand, it is a great advantage, because it forces upon you a much clearer understanding of what these models mean.

To illustrate, we will look at how to describe a certain linear model in all the different ways listed above, and then extend that example to summarize some typical linear models that can be fitted for covariates and factors. We will also see how to combine factors and continuous covariates in additive and interactive ways (see also chapter 6 in Kéry 2010). We focus on the algebraic and the R language descriptions of a linear model, because you are likely to know and use R already and because this will be the way in which you specify linear models within HMs in `unmarked`. In addition, we show the syntax for fitting the same linear models in the BUGS language.

3.2.1 Linear models with main effects of one factor and one continuous covariate

We use our dragonfly example to illustrate the description and fitting of a linear model where `pop` and `length` act in an additive fashion on wing span, namely according to the linear model underlying a least-squares technique known as a "main-effects analysis of covariance (ANCOVA)". In section 3.2.2, we extend the main-effects model to an interaction-effects ANCOVA model. We can fit the main-effects model in R by issuing the following commands:

```
summary(fm1 <- lm(wing ~ pop + body))

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)    6.5296     1.6437   3.973  0.01061 *
popAragon      1.8706     0.4521   4.138  0.00902 **
popCatalonia   1.7333     0.4489   3.861  0.01187 *
body           0.5149     0.1991   2.586  0.04908 *
[ ... ]
Residual standard error: 0.5498 on 5 degrees of freedom
Multiple R-squared:  0.8416,    Adjusted R-squared:  0.7465
F-statistic: 8.854 on 3 and 5 DF,  p-value: 0.01916
```

How many parameters does our model have ? Where are the estimates of these parameters presented in the R output ? What do these parameters mean ? To understand this, let's look at the description of this model in algebra. We can write this model as follows:

$$wing_i = \mu + \alpha_j + \beta * body_i + \varepsilon_i$$
$$\varepsilon_i \sim Normal(0, \sigma^2)$$

Here, $wing_i$ is the response of unit i (= wing length of dragonfly *i*, data point) and $body_i$ is the value of covariate `body length` for dragonfly i. The intercept μ is a constant shared by the response of all nine dragonflies. Vector α_j has three elements, corresponding to the number of levels, or populations, in the factor `pop`, and contains the *effects* of each population. (We could write $\alpha_{j(i)}$ to emphasize that population membership *j* is a function of the individual *i*.) For population 1 (Navarra) we have α_1, for population 2 (Aragon) α_2 and for population 3 (Catalonia) α_3. Parameter β is the expected change in wing span for a 1-unit change of body length. The unexplained part in the wing span of dragonfly *i*, which is not accounted for by population membership or body length of dragonfly *i*, is the residual ε_i. The residuals for all nine dragonflies are assumed to be independent draws from the same normal distribution with constant variance. The residual variance (σ^2) is also an estimated model parameter. In the R output (under `Residual standard error`), we see the square root of that variance estimate (σ).

Hence, our model has four parameters (α_j and β) to describe the expected response and one parameter (σ) for the variability of the response around that mean. Importantly, when fitting linear models, stats packages like R internally do not make a difference between the two types of explanatory variables (covariates and factors, although factors must be declared as such). All linear models are internally represented as a multiple linear regression, where the effects of a factor are broken apart into those of two or more indicator or dummy variables that contain 0's and 1's only. In our case for `pop`, we have three dummy variables, one for each level (population) of the `pop` factor, to code for the presence or absence of an effect of that population in the expected response of each dragonfly.

However, the model above is overparameterised or parameter-redundant (Mead 1988; Ntzoufras 2009): we try to estimate one parameter too many. Specifically, we cannot estimate both the intercept μ and a full parameter vector $\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, \alpha_3\}$ of length three. One of the population effects α_j has to be set to zero to make the model identifiable. Customarily in R, the parameter corresponding to the first factor level is set to zero, i.e., the constraint $\alpha_1 = 0$ is imposed. As a consequence, the intercept μ becomes the intercept of the regression line of dragonflies in population 1; this is the expected wing span of a (hypothetical) dragonfly of length zero in Navarra. The remaining two elements of the parameter vector $\boldsymbol{\alpha}$, which are not fixed but estimated, then become the differences between the intercepts of the regression lines in populations 2 and 3, relative to the intercept in population 1 (this is why seemingly only coefficient estimates for Aragon and Catalonia are shown in the R output above). In this model parameterisation, population 1 (Navarra) serves as a baseline or reference and the two parameters estimated for the population factor represent comparisons of the other populations with that one. This parameterisation of the effects of the factor `pop` may therefore be called the *treatment contrast* or *effects parameterisation*.

How do we write this model in the BUGS language ? We will see in much more detail later (starting in chapter 5), that BUGS is not a vectorized language: we cannot specify a linear model simply as `y ~ x` as in R. Rather, we have to specify the stochastic relationship between y and x for each element/observation in the two vectors by looping over them from the first until the last element. Elements of vectors are indexed by square brackets, hence, our BUGS code will be this (this is *not* executable as is; and the text after the hash mark # is an explanatory comment, not part of the model specification):

```
for(i in 1:9){                                     # Loop over each individual
   wing[i] ~ dnorm(mean[i], tau)                   # Stochastic relationship
   mean[i] <- mu + alpha[pop[i]] + beta * body[i] # Deterministic relationship
}
alpha[1] <- 0                                      # Fix effect of 1st level at 0
```

This says that the wing span of dragonfly *i* is a draw from a normal distribution with an expected value (or mean) called `mean[i]` and precision `tau`. In BUGS, the dispersion parameter of the normal distribution is not the variance or the standard deviation, but the precision (the reciprocal of the variance). The expected wing span of dragonfly *i* , `mean[i]`, is the sum of an intercept `mu`, a population-specific effect `alpha[pop[i]]`, and an effect `beta` of body length. Thus, inside of the loop we write exactly what we might write in algebra to describe that model (see also below for different algebraic ways of writing the same model, especially variant 2).

We make the following three remarks:

(1) In a statistical model, the observed data are assumed to be the outcome of a stochastic process (see Chapter 2). In line with this, observed data in BUGS must always be a stochastic quantity, i.e., stand on the left side of the twiddle or tilde symbol, which indicates a stochastic relationship. Residuals are only defined implicity, and although correct in algebra, it would *not* be correct in BUGS to write the following:

```
for(i in 1:9){
   wing[i] <- mu + alpha[pop[i]] + beta * body[i] + eps[i]
   eps[i] ~ dnorm(0, tau)            # In BUGS this is WRONG !
}
```

The reason is that you can't simply calculate an observed quantity such as `wing[i]`, it has to be stochastic, i.e. a draw from some distribution.

(2) To specify effects of factor levels, or more generally, parameters that are grouped in a vector or higher-dimensional array, we use *nested indexing* in the BUGS language: `alpha[pop[i]]`, where `pop` is the indexing variable. Nested indexing can be best understood when read from the inside out: for instance, dragonfly i=1 has (internally in R) a value of pop[1] = 1, hence its expected (mean) wing span will get a contribution of `alpha[1]`. Dragonflies 6 and 7 have (internal) values of the `pop` factor of 2 and 3, respectively. Hence, their expected wing span will get contributions of alpha[2] and alpha[3], respectively. Be careful with the indices when writing a model in algebra, and similarly, in the BUGS language indexing of arrays is one of the more complicated things for a beginner to grasp. As said above, we might have written $\alpha_{j(i)}$ rather than simply α_j to clarify the algebraic description of the model. This would emphasize that the information given by index *j* (population membership of a dragonfly) is equivalent to the factor `pop` in the analysis.
(3) Factor levels in BUGS cannot be letters or words such as in our example here, but they must be integers ranging from 1 to the number of distinct levels. Thus, in a BUGS analysis, we would have to convert our factors `pop` and `sex` into numerical values and only (1,2,3) and (1,2), respectively, will be correct, while, for instance, (1,2,4) for pop or (0,1) for sex will result in errors when using nested indexing.

There are several ways to write what is essentially the same linear model, and these are called *parameterisations* of a model, for instance:

$$wing_i = \alpha_j + \beta * body_i + \varepsilon_i$$
$$\varepsilon_i \sim Normal(0, \sigma^2)$$

Here, everything is the same as before, except that the model no longer has the parameter μ, and the meaning of the parameters α_j representing the effects of the levels of factor `pop` has changed. Now, all three elements of this parameter vector are estimable, since there is no longer a separate intercept (μ). They now represent directly the intercepts of the three regression lines, i.e., the expected wing spans of a dragonfly at body length zero in the three populations. This parameterisation of the factor `pop` can be called the *means parameterization*. We can fit it in R by "subtracting the intercept" in the model formula of the function call:

```
summary(fm2 <- lm(wing ~ pop-1 + body))

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
popNavarra     6.5296     1.6437   3.973  0.01061 *
popAragon      8.4003     1.5916   5.278  0.00325 **
popCatalonia   8.2630     1.6437   5.027  0.00401 **
body           0.5149     0.1991   2.586  0.04908 *
[ ... ]
Residual standard error: 0.5498 on 5 degrees of freedom
Multiple R-squared:  0.9988,    Adjusted R-squared:  0.9979
F-statistic:  1053 on 4 and 5 DF,  p-value: 1.694e-07
```

This is exactly the same model as before: the parameter estimates are either the same (for popNavarra, body and the residual standard error) or they can be obtained by adding two of the parameters in the former parameterisation. Hence, the value of popAragon here is obtained by adding the value of the intercept and of popAragon in the previous parameterisation (i.e., 8.4002 = 6.5296 + 1.8706) and similar for popCatalonia (i.e., 8.2629 = 6.5296 + 1.7333; differences are due to rounding). The advantage of this parameterisation is perhaps ease of interpretation. However, it cannot be adopted for models with multiple factors, for which we must choose the effects parameterisation or impose sum-to-zero constraints (see below and Ntzoufras 2009).

In BUGS, the means parameterisation of the model is specified simply like this:

```
for(i in 1:9){
   wing[i] ~ dnorm(mean[i], tau)
   mean[i] <- alpha[pop[i]] + beta * body[i]
}
```

Hence, the BUGS code inside of the loop is again very similar to how we write the model in algebra. Indeed, it is identical to version 2 of the following three algebraically synonymous model descriptions:

(1) $wing_i \sim Normal(\alpha_j + \beta * body_i, \sigma^2)$

(2) $wing_i \sim Normal(\mu_i, \sigma^2)$, with $\mu_i = \alpha_j + \beta * body_i$

(3) $wing_i = \alpha_j + \beta * body_i + \varepsilon_i$, with $\varepsilon_i \sim Normal(0, \sigma^2)$

It is very important that you know how to write your linear models in algebra, because the BUGS language model description very much resembles the model's representation in algebra, so much so, indeed, that once you know how to write the model in algebra, you have almost written it in the BUGS language ! Furthermore, BUGS software can be sensitive to the parameterisation chosen: sometimes one may work well and the other not. Therefore, it is good to be able to switch between different model parameterisations and try which works (best).

Another parameterisation of a linear model for factors imposes a sum-to-zero constraint, i.e., the effects of the factor levels are estimated such that their combined effect is zero. They can then be interpreted as deviations from the common mean of zero; this again allows a separate intercept to be estimated, as for the effects parameterisation. This parameterisation may have some advantages for ease of interpretation in models with multiple factors and is easily coded in BUGS (see Ntzoufras 2009).

The intercept for each population is the expected wing span of a dragonfly of body length zero. This is of course nonsensical, and a more meaningful model is obtained when we regress wing span not on body length, but on *centered* body length, i.e., on a transformation of original body length obtained by subtracting the sample mean of body length. The intercept is then the expected wing span of a dragonfly at the average observed body length; a much more meaningful parameter. Moreover, centering covariates is often helpful to obtain convergence both in maximum likelihood and Bayesian MCMC analyses. Often, to avoid numerical problems in unmarked or BUGS, we have to standardize covariates, i.e., center them and then divide the result by some amount, typically the covariate sample standard deviation. Note that while centering only affects the intercept, standardizing affects both the intercept and the slope of a covariate. When scaling a covariate with the standard deviation of the observed values of a covariate, the slope will be the expected change in the response variable for a 1 unit change in the scaled covariate, which corresponds to a 1 standard deviation change of the original covariate.

After fitting a model we may often want to make predictions (to draw figures or even to understand what our model is telling us about the data), i.e., compute the expected response given the parameter estimates for certain values of the covariates. This can easily be done by hand or you can use the R function `predict` to do this for you automatically, including an assessment of prediction uncertainty (standard errors). It is trivial as well in BUGS, but there you have to do this "by hand". We will see many examples of predictions with R or BUGS in this book (e.g., this is emphasized in chapter 5), but here as a mini example we simply point out that the expected wing span of a dragonfly of body length 8 in Aragon can be estimated using the output of `fm1` as 8.4003 + 0.5149 * 8 = 12.5195.

In statistics books, you are likely to see linear models described in matrix algebra, for instance as $\mathbf{y} = \mathbf{X\beta} + \boldsymbol{\varepsilon}$. What does this mean ? Matrices and vectors are simply a manner in which numbers can be stored in an orderly way in an array. Hence, for our dragonfly example $\mathbf{y}$ is a vector of length nine containing the responses, i.e., the wing span of each dragonfly; $\mathbf{X}$ is the design matrix of the fitted model (also called model matrix or X matrix), $\boldsymbol{\beta}$ is the parameter vector, and $\boldsymbol{\varepsilon}$ is the vector of residuals. We call the result of the matrix multiplication of the design matrix and the parameter vector, $\mathbf{X}\beta$, the linear predictor; this is the expected wing span, given population membership and body length of each dragonfly, in the absence of the random noise $\boldsymbol{\varepsilon}$.

To better understand this, it is useful to write it out with the numbers of our data set plugged into the various matrices and vectors. We do this first for the effects parameterisation and then for the means parameterisation. We will see that the structure of the linear model is exactly defined by the design matrix $\mathbf{X}$ and that this, moreover, determines the interpretation of the fitted parameters. For the effects parameterisation, we have for $\mathbf{y} = \mathbf{X\beta} + \boldsymbol{\varepsilon}$:

$$\begin{pmatrix} 10.5 \\ 10.6 \\ 11.0 \\ 12.1 \\ 11.7 \\ 13.5 \\ 11.4 \\ 13.0 \\ 12.9 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 6.8 \\ 1 & 0 & 0 & 8.3 \\ 1 & 0 & 0 & 9.2 \\ 1 & 1 & 0 & 6.9 \\ 1 & 1 & 0 & 7.7 \\ 1 & 1 & 0 & 8.9 \\ 1 & 0 & 1 & 6.9 \\ 1 & 0 & 1 & 8.2 \\ 1 & 0 & 1 & 9.2 \end{pmatrix} \begin{pmatrix} \mu \\ \alpha_2 \\ \alpha_3 \\ \beta \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \varepsilon_4 \\ \varepsilon_5 \\ \varepsilon_6 \\ \varepsilon_7 \\ \varepsilon_8 \\ \varepsilon_9 \end{pmatrix}, \text{ with } \varepsilon_i \sim Normal(0, \sigma^2)$$

From left to right, we have response vector, design matrix, parameter vector and residual vector. The number of columns of the design matrix always matches the length of the parameter vector. The design matrix contains indicator variables (zeroes and ones) for the effects of factors and the measured covariate values for continuous or numerical explanatory variables. Here, the first column of the design matrix defines the intercept (which is equal to the expected wing span of an individual with body length zero in population Navarra) by a vector of all ones, while in the next two columns, ones indicate individuals in populations Aragon and Catalonia, respectively. In this description of the linear model, we recognize particularly clearly the structure of a multiple linear regression, with as many single-degree-of-freedom terms (scalar parameters) as columns in the design matrix.

The design matrix multiplied with the parameter vector produces another vector, η_i, called the linear predictor, which is the expected wing span for each dragonfly. For dragonfly 1, it is given by $1*\mu + 0*\alpha_2 + 0*\alpha_3 + 6.8*\beta$, hence, its expected wing span contains one contribution from μ and 6.8 contributions from β. Likewise, for dragonfly 9, it is $1*\mu + 0*\alpha_2 + 1*\alpha_3 + 9.2*\beta$. With fitted model `fm1`, we can obtain the expected wing length "by hand" (using the powerful `model.matrix` function, which we describe in more detail below) or using the `predict()` function after fitting the linear model using `lm()`; note that `%*%` denotes matrix multiplication:

```
cbind(model.matrix(~pop+body) %*% fm1$coef, predict(fm1)) # Compare two
solutions
```

```
      [,1]     [,2]
1 10.03068 10.03068
2 10.80297 10.80297
3 11.26635 11.26635
4 11.95280 11.95280
5 12.36469 12.36469
6 12.98252 12.98252
7 11.81550 11.81550
8 12.48482 12.48482
9 12.99968 12.99968
```

Here is the means parameterisation in the vector-matrix description:

$$\begin{pmatrix} 10.5 \\ 10.6 \\ 11.0 \\ 12.1 \\ 11.7 \\ 13.5 \\ 11.4 \\ 13.0 \\ 12.0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 6.8 \\ 1 & 0 & 0 & 8.3 \\ 1 & 0 & 0 & 9.2 \\ 0 & 1 & 0 & 6.9 \\ 0 & 1 & 0 & 7.7 \\ 0 & 1 & 0 & 8.9 \\ 0 & 0 & 1 & 6.9 \\ 0 & 0 & 1 & 8.2 \\ 0 & 0 & 1 & 9.2 \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \beta \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \varepsilon_4 \\ \varepsilon_5 \\ \varepsilon_6 \\ \varepsilon_7 \\ \varepsilon_8 \\ \varepsilon_9 \end{pmatrix} \text{, with } \varepsilon_i \sim Normal(0, \sigma^2)$$

The expected wing span of dragonfly 1 is still given by $1*\alpha_1 + 0*\alpha_2 + 0*\alpha_3 + 6.8*\beta$, which is the same as before, except for the small semantic change that we now call the intercept in population Navarra α_1 instead of μ. In contrast, for dragonfly 9 we get $0*\alpha_1 + 0*\alpha_2 + 1*\alpha_3 + 9.2*\beta$ and so we see that the intercept of the regression line in population 3 is now given by α_3, rather than by $\mu + \alpha_3$ as in the effects parameterisation of the model. When we fit a linear model, we implicitly solve a system of equations subject to some constraints on the noise terms ε_i:

$$\begin{aligned}
10.5 &= 1*\alpha_1 + 0*\alpha_2 + 0*\alpha_3 + 6.8*\beta + \varepsilon_1 \\
10.6 &= 1*\alpha_1 + 0*\alpha_2 + 0*\alpha_3 + 8.3*\beta + \varepsilon_2 \\
11.0 &= 1*\alpha_1 + 0*\alpha_2 + 0*\alpha_3 + 9.2*\beta + \varepsilon_3 \\
12.1 &= 0*\alpha_1 + 1*\alpha_2 + 0*\alpha_3 + 6.9*\beta + \varepsilon_4 \\
11.7 &= 0*\alpha_1 + 1*\alpha_2 + 0*\alpha_3 + 7.7*\beta + \varepsilon_5 \\
13.5 &= 0*\alpha_1 + 1*\alpha_2 + 0*\alpha_3 + 8.9*\beta + \varepsilon_6 \\
11.4 &= 0*\alpha_1 + 0*\alpha_2 + 1*\alpha_3 + 6.9*\beta + \varepsilon_7 \\
13.0 &= 0*\alpha_1 + 0*\alpha_2 + 1*\alpha_3 + 8.2*\beta + \varepsilon_8 \\
12.0 &= 0*\alpha_1 + 0*\alpha_2 + 1*\alpha_3 + 9.2*\beta + \varepsilon_9
\end{aligned}$$

Indeed, the least-squares model fitting criterion used by the R function `lm` chooses the parameter values such that the sum of the squared residuals is minimized (hence the name). This technique does not make any distributional assumptions about the residuals, but the parameter estimates and standard errors are identical to those obtained using the maximum likelihood method when we make the explicit assumption that the residuals have a normal distribution.

The design matrix lies at the heart of a linear model and it is imperative that we obtain a clear understanding of it when fitting linear models. R has the powerful `model.matrix` function, which yields the design matrix for any linear model specified in the model definition language of R. This can be very useful to understand a model, and can even be useful when fitting the model in BUGS, for instance by clarifying what a linear model looks like. Let's look at the design matrices of the two parameterisations of the main-effects ANCOVA model.

```
model.matrix(~ pop + body) # Effects parameterisation
  (Intercept) popAragon popCatalonia body
1           1         0            0  6.8
2           1         0            0  8.3
3           1         0            0  9.2
4           1         1            0  6.9
5           1         1            0  7.7
6           1         1            0  8.9
7           1         0            1  6.9
8           1         0            1  8.2
9           1         0            1  9.2

model.matrix(~ pop-1 + body) # Means parameterization
  popNavarra popAragon popCatalonia body
1          1         0            0  6.8
2          1         0            0  8.3
3          1         0            0  9.2
4          0         1            0  6.9
5          0         1            0  7.7
6          0         1            0  8.9
7          0         0            1  6.9
8          0         0            1  8.2
9          0         0            1  9.2
```

Finally, the geometrical repesentation of the main-effects (i.e., additive) ANCOVA model is that of a bundle of parallel regression lines, one for each population (Fig. 3–3A). They are parallel because the slope is shared among all three.

```
par(mfrow = c(1, 3), mar = c(5,4,2,2), cex = 1.2, cex.main = 1)
plot(body[sex == "M"], wing[sex == "M"], col = colorM, xlim = c(6.5, 9.5), ylim
= c(10, 14), lwd = 2, frame.plot = FALSE, las = 1, pch = 17, xlab = "Body
length", ylab = "Wing span")
points(body[sex == "F"], wing[sex == "F"], col = colorF, pch = 16)
abline(coef(fm2)[1], coef(fm2)[4], col = "red", lwd = 2)
abline(coef(fm2)[2], coef(fm2)[4], col = "blue", lwd = 2)
abline(coef(fm2)[3], coef(fm2)[4], col = "green", lwd = 2)
text(6.8, 14, "A", cex = 1.5)
```

The form of the design matrix determines the interpretation of the parameters. Table 1 gives an overview of a range of linear models that can be fitted for two explanatory variables when one is a factor and the other is a covariate. We also list the case of an interaction between `pop` and `body` and briefly review the important concept of interaction in the next section.

3.2.2 Linear models with interaction between one factor and one continuous covariate

When two terms in a linear model interact, the effect of one depends on the value of the other and vice versa. We illustrate this in the context of our ANCOVA linear model with `pop` and `body`, which, when the two interact, geometrically represents a bundle of non-parallel regression lines (Fig. 3–3B). Interaction also means that the effects of the terms are combined in a multiplicative, rather than in a purely additive manner. The design matrix then contains additional columns that represent the products of the columns that stand for the main effects. In a sense, the polynomial terms in the preceding section represent such interactions of a covariate with itself.

```
model.matrix(~ pop*body)  # Effects parameterisation
  (Intercept) popAragon popCatalonia body popAragon:body popCatalonia:body
1           1         0            0  6.8            0.0               0.0
2           1         0            0  8.3            0.0               0.0
3           1         0            0  9.2            0.0               0.0
4           1         1            0  6.9            6.9               0.0
5           1         1            0  7.7            7.7               0.0
6           1         1            0  8.9            8.9               0.0
7           1         0            1  6.9            0.0               6.9
8           1         0            1  8.2            0.0               8.2
9           1         0            1  9.2            0.0               9.2
```

Here, the main effect of pop is represented by columns 2 and 3, and that of body by column 4 and the interaction terms represent the product of columns 2 and 4 and 3 and 4, respectively.

```
model.matrix(~ pop*body-1-body)  # Means parameterisation
# Output slightly trimmed
  popNavarra popAragon popCatalonia popNa:body popAr:body popCat:body
1          1         0            0        6.8        0.0         0.0
2          1         0            0        8.3        0.0         0.0
3          1         0            0        9.2        0.0         0.0
4          0         1            0        0.0        6.9         0.0
5          0         1            0        0.0        7.7         0.0
6          0         1            0        0.0        8.9         0.0
7          0         0            1        0.0        0.0         6.9
8          0         0            1        0.0        0.0         8.2
9          0         0            1        0.0        0.0         9.2
```

Now, the main effect of pop is represented by columns 1 to 3, and the interactive effect with body directly by columns 4–6. Let's fit this latter parameterisation and plot the lines of best fit (Fig. 3–3B).

```
summary(fm3 <- lm(wing ~ pop*body-1-body))
Call:
lm(formula = wing ~ pop * body - 1 - body)
[ ... ]
Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
popNavarra          9.1296     2.7626   3.305   0.0456 *
popAragon           6.4553     3.2116   2.010   0.1380
popCatalonia        6.9217     2.9023   2.385   0.0972 .
popNavarra:body     0.1939     0.3385   0.573   0.6070
popAragon:body      0.7632     0.4077   1.872   0.1580
popCatalonia:body   0.6805     0.3559   1.912   0.1518
[ ... ]
Residual standard error: 0.5805 on 3 degrees of freedom
Multiple R-squared:  0.9992,    Adjusted R-squared:  0.9976
F-statistic: 629.9 on 6 and 3 DF,  p-value: 9.763e-05

# Plot
plot(body[sex == "M"], wing[sex == "M"], col = colorM, xlim = c(6.5, 9.5), ylim
= c(10, 14), lwd = 2, frame.plot = FALSE, las = 1, pch = 17, xlab = "Body
length", ylab = "")
points(body[sex == "F"], wing[sex == "F"], col = colorF, pch = 16)
abline(coef(fm3)[1], coef(fm3)[4], col = "red", lwd = 2)
abline(coef(fm3)[2], coef(fm3)[5], col = "blue", lwd = 2)
abline(coef(fm3)[3], coef(fm3)[6], col = "green", lwd = 2)
text(6.8, 14, "B", cex = 1.5)
```

In BUGS, the interactive model in the effects and in the means parameterisations just shown are specified as follows:

```
for(i in 1:9){   # Effects parameterisation
   wing[i] ~ dnorm(mean[i], tau)
   mean[i] <- mu.alpha + alpha[pop[i]] + mu.beta * body[i] + beta[pop[i]] *
body[i]
}
alpha[1] <- 0     # Impose 'corner constraints' for identifiability
beta[1] <- 0

for(i in 1:9){   # Means parameterisation
   wing[i] ~ dnorm(mean[i], tau)
   mean[i] <- alpha[pop[i]] + beta[pop[i]] * body[i]
}
```

We can also fit what might be called partially interactive models. We illustrate this and show how we can directly fit a model matrix using R functions such as `lm` or `glm`. Note first that the fitted regression lines in Catalonia and Aragon are rather similar, while that in Navarra is different (Fig. 3–3A). So let's create a design matrix for a partially interactive model, with identical slopes for Aragon and Catalonia and a separate slope for Navarra, while keeping the intercepts separate for all three populations. We can conduct a significance test to compare the two models (and find out that the partially interactive model is better) and then plot the best fit-lines under this model (Fig. 3–3C). We take the last design matrix, copy parts of the column associated with the slope in Catalonia into the slope column for Aragon and then delete the slope column for Catalonia.

```
# Create new design matrix
(DM0 <- model.matrix(~ pop*body-1-body)) # Original DM for means param
DM0[7:9,5] <- DM0[7:9,6]                 # Combine slopes for Ar and Cat
(DM1 <- DM0[, -6])                       # Delete former slope column for Cat
  popNavarra popAragon popCatalonia popNavarra:body popAragon:body
1          1         0            0             6.8            0.0
2          1         0            0             8.3            0.0
3          1         0            0             9.2            0.0
4          0         1            0             0.0            6.9
5          0         1            0             0.0            7.7
6          0         1            0             0.0            8.9
7          0         0            1             0.0            6.9
8          0         0            1             0.0            8.2
9          0         0            1             0.0            9.2

# Fit model with partial interaction
summary(fm4 <- lm(wing ~ DM1-1))

Call:
lm(formula = wing ~ DM1 - 1)
[ ... ]
Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
DM1popNavarra        9.1296     2.4018   3.801   0.0191 *
DM1popAragon         6.8230     1.8491   3.690   0.0210 *
DM1popCatalonia      6.6320     1.9106   3.471   0.0256 *
DM1popNavarra:body   0.1939     0.2943   0.659   0.5461
DM1popAragon:body    0.7162     0.2331   3.072   0.0372 *
[ ... ]
Residual standard error: 0.5047 on 4 degrees of freedom
Multiple R-squared:  0.9992,    Adjusted R-squared:  0.9982
F-statistic:  1000 on 5 and 4 DF,  p-value: 2.793e-06
```

```
# Do significance test
anova(fm3, fm4)                 # F test between two models
Analysis of Variance Table

Model 1: wing ~ pop * body - 1 - body
Model 2: wing ~ DM1 - 1
  Res.Df    RSS Df  Sum of Sq      F Pr(>F)
1      3 1.0109
2      4 1.0187 -1 -0.0078683 0.0234 0.8882

# Plot
plot(body[sex == "M"], wing[sex == "M"], col = colorM, xlim = c(6.5, 9.5), ylim
= c(10, 14), lwd = 2, frame.plot = FALSE, las = 1, pch = 17, xlab = "Body
length", ylab = "")
points(body[sex == "F"], wing[sex == "F"], col = colorF, pch = 16)
abline(coef(fm4)[1], coef(fm4)[4], col = "red", lwd = 2)
abline(coef(fm4)[2], coef(fm4)[5], col = "blue", lwd = 2)
abline(coef(fm4)[3], coef(fm4)[5], col = "green", lwd = 2)
text(6.8, 14, "C", cex = 1.5)
```

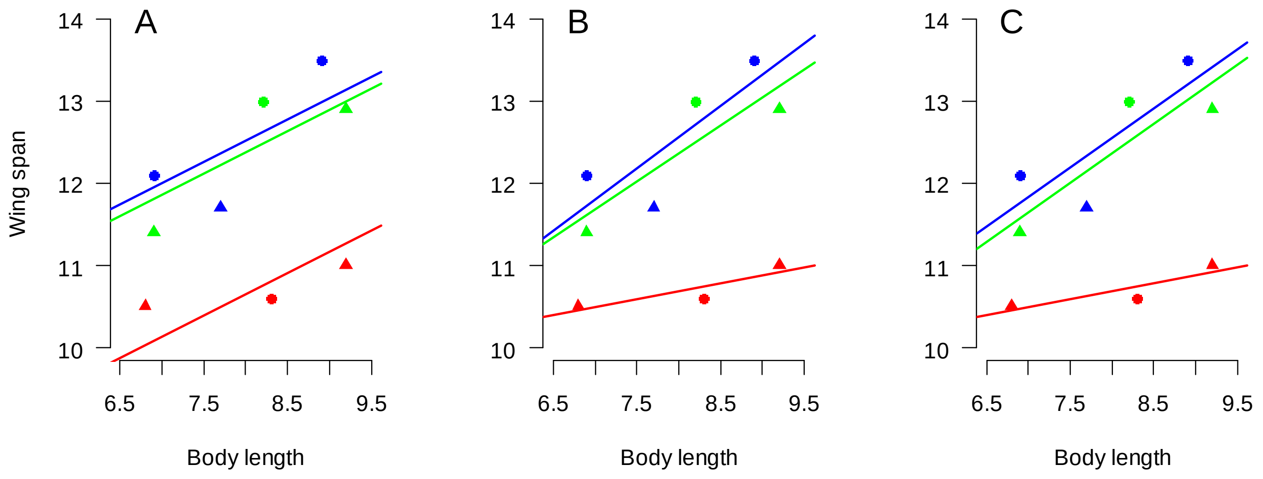


Fig. 3–3: Geometric representation of the main-effects ANCOVA model (additive effects of population and body length; A), the interaction-effects ANCOVA model (multiplicative effects of population and body length; B) and the partially interactive-effects ANCOVA model (C) in the *O. uncatus* toy data set. Color code and symbols as in Fig. 3–2.

In BUGS, we can use function `inprod` to directly fit a design matrix, which must be provided to BUGS as data.

```
for(i in 1:9){
   wing[i] ~ dnorm(mean[i], tau)
   mean[i] <- inprod(DM1[i,], beta[]) # Requires ncol(DM1) = length(beta)
}
```

The ability to directly fit a design matrix in BUGS, whether directly generated by `model.matrix` in R or after some modifications as in our example, can sometimes be quite useful.

3.3 Generalised linear models (GLMs)

Generalised linear models (GLMs) extend the concept of a linear effect of covariates to response variables for which a statistical distribution other than a normal may be assumed, such as the

Poisson, binomial/Bernoulli, gamma, or exponential distributions. Perhaps the key feature of GLMs is that the linear effect of covariates is expressed not for the expected response directly, but for a *transformation* of the expected response (Nelder & McCullagh 1989). That transformation is called the link function. Generally, we can describe a GLM for response y_i in terms of three components:

1. *Random part of the response (or the error structure* of the model*)*: a statistical distribution f with parameter(s) θ

$$y_i \sim f(\theta)$$

2. *A link function* g, which is applied to the expected response $E(y) = \mu_i$, with η_i known as the *linear predictor*:

$$g(E(y)) = g(\mu_i) = \eta_i$$

3. *Systematic part of the response* (or the *mean structure* of the model): the linear predictor (η_i), which contains a linear model, such as:

$$\eta_i = \alpha + \beta * x_i$$

We can combine elements 2 and 3 and define a GLM succinctly in just two lines:

$$y_i \sim f(\theta)$$

$$g(\mu_i) = \alpha + \beta * x_i$$

Hence, a response y follows a distribution f with parameter(s) θ, and a transformation g of the expected, or mean, response is modelled as a linear function of covariates; that's the GLM in a nutshell. And this is exactly the way in which a GLM is specified in the BUGS language and the reason for why BUGS is so great to help us *really* understand GLMs.

The GLM concept gives you considerable creative freedom in combining the three components of a GLM, but there are typically pairs of response distributions and link functions that go together particularly well. These latter are called the *canonical* link functions. They are the identity link for normal responses ($\eta_i = \mu_i$), the log link for Poisson responses ($\eta_i = \log(\mu_i)$) and the logit link for binomial or Bernoulli responses ($\eta_i = \log(\mu_i / (1 - \mu_i))$); 'identity' simply means that the expected response is not transformed but is directly modeled as a linear function of explanatory variables. Together, these three standard GLMs make up a vast number of statistical methods used in ecology and elsewhere; for overviews, see Dobson & Barnett (2008) or Ntzoufras (2009).

The vast scope of the GLM in applied statistical analysis is one reason for the great importance of the GLM to you. A second one is that GLMs unify a very large number of seemingly unrelated models and analysis techniques; hence, understanding GLMs helps you achieve a synthetic understanding of very many techniques and models. And finally, as we will see many times in this book, GLMs are the main building blocks for most hierarchical models that we develop in later chapters, including occupancy and N-mixture models first introduced in chapter 2. Many exciting ecological models for inference about populations or communities can be viewed simply as a sequence of coupled GLMs (Royle & Dorazio, 2008; Kéry & Schaub, 2012).

In section 3.2, we described what is essentially a normal-response GLM, and here we illustrate three of the most typical GLMs with non-normal responses: Poisson, Bernoulli and binomial GLMs. All three are of fundamental importance for the HMs in this book, because the Poisson GLM is our typical model for spatio-temporal variation of abundance, and the Bernoulli GLM plays the analogous role for occurrence ("presence/absence" or distribution data). Furthermore, in models with population dynamics (see chapters 14–17), the Poisson is a natural model for the recruitment and the binomial for the survival process. Finally, our canonical description of the observation process governed by false-negatives is the Bernoulli or the binomial distribution (Royle & Dorazio 2008; Kéry & Schaub 2012), although sometimes a Poisson, or a Bernoulli that *implies* an underlying Poisson process, may be more adequate; see Efford et al. (2013), Royle et al. (2014) and also 3.3.6 and exercises 3.3 and 4. 8. If you understand Poisson and Bernoulli GLMs, you are in good shape to understand all the HMs in this book.

Technically, GLMs are defined for all members of statistical distributions belonging to the so-called exponential family (McCullagh & Nelder 1989; Dobson & Barnett 2008; Ntzoufras 2009), which includes normal, Poisson, binomial/Bernoulli, multinomial, negative binomial, beta, gamma, lognormal, exponential and Dirichlet. Thus, the principles of linear modeling can be carried over to a vast number of "error models" other than the normal, i.e., to a very large number of stochastic processes that produce random outcomes which are predictable in some average sense only.

3.3.1 Poisson generalised linear model (GLM) for unbounded counts

The Poisson distribution is the number of "things" for a Poisson process, which is a natural model for independent discrete "things" indexed by space or time (Gelman et al. 2014). "Things" is a placeholder for much of what scientists usually count. A Poisson GLM is a natural choice for unbounded counts, i.e., non-negative integers that have no logical upper bound (unlike binomial counts, which cannot be greater than their sample size; see section 3.3.7). We will use the parasite load in our toy dragonflies for illustration. For the number of mites counted on dragonfly *i*, C_i, we can write the main-effects ANCOVA linear model for factor `pop` and the body length covariate `body` within a Poisson GLM as follows:

1. Random part of the response (statistical distribution for the randomness in the response):

$$C_i \sim \text{Poisson}(\lambda_i)$$

2. Link of random and systematic part in response (link function):

$$\log(E(C)) = \log(\lambda_i) = \eta_i$$

3. Systematic part of the response (linear predictor η_i for link-transformed mean response):

$$\eta_i = \alpha_j + \beta * body_i$$

Here, λ_i is the expected (or mean) response for dragonfly i on the arithmetic scale ($E(C)$), η_i is the expected count on the log link scale, also called the linear predictor, $body_i$ is the body length of dragonfly i and α and β are the two parameters (α being vector-valued) of the log-linear regression of the counts on factor `pop` and covariate `body`.

We fit this model in R and give the BUGS code for fitting it. Once again, we will see that the model descriptions in algebra and in the BUGS language are essentially identical. Here is the model in algebra in short; note that C corresponds to the variable `mites` in our toy data.

$$C_i \sim \text{Poisson}(\lambda_i)$$
$$\log(\lambda_i) = \alpha_j + \beta * body_i$$

```
summary(fm10 <- glm(mites ~ pop-1 + body, family = poisson))

Call:
glm(formula = mites ~ pop - 1 + body, family = poisson)
[ ... ]
Coefficients:
             Estimate Std. Error z value Pr(>|z|)
popNavarra     -6.634      2.325  -2.854  0.00432 **
popAragon      -5.878      2.216  -2.653  0.00799 **
popCatalonia   -5.519      2.290  -2.410  0.01596 *
body            0.845      0.261   3.237  0.00121 **
[ ... ]
(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 59.658  on 9  degrees of freedom
Residual deviance: 15.467  on 5  degrees of freedom
AIC: 42.591

Number of Fisher Scoring iterations: 6
```

We can specify the linear model for the factor `pop` in the means or in the effects parameterisation, exactly as before. The methods for fitting and interpreting covariates, including two factors such as `pop` and `sex`, are exactly analogous for linear models and GLMs. To specify a Poisson GLM in the BUGS language using the means parameterisation, we write:

```
for(i in 1:9){
   mites[i] ~ dpois(lambda[i])
   log(lambda[i]) <- alpha[pop[i]] + beta * body[i]
}
```

This is *exactly* how we just wrote the model in algebra, where `pop[i]` now takes the place of the index *j* for the intercept α , i.e., α_j in algebra is written as `alpha[pop[i]]` in the BUGS language.

3.3.5 Bernoulli GLM: logistic regression for a binary response

We use a Bernoulli distribution to describe patterns in binary responses, i.e., the outcome of coin flip-like random processes. The Bernoulli distribution is also important because the perhaps more commonly encountered binomial distribution with trial size *N* is the sum of the outcomes of *N* independent Bernoulli trials. In this book, the Bernoulli (or alternatively, the binomial; see 3.3.7) is the canonical description of the measurement error process involved in all ecological studies of distribution or abundance, where the presence of a species or of an individual can be overlooked with probability equal to 1 minus the detection probability. That is, the probability of measurement error is the converse of detection probability. Unlike in most sciences where measurement error is typically modelled as a continuous random process (for instance using a normal distribution), for the measurement of distribution or abundance, measurement error is a binary process at the scale of each individual or the presence/absence state of species at a site.

For our example data set, we can define a binary random variable, where the state "parasite presence" corresponds to a parasite load greater than 0. The event of a dragonfly with "parasite presence" is a so-called "success" (albeit less so for the dragonfly) and coded as 1, while the converse, "parasite absence", is traditionally called a "failure" and coded as 0. We call y the indicator for a parasite load greater than 0; it contains a 1 if a dragonfly has at least one parasite and a 0 if it is parasite-free. Our Bernoulli GLM for the occurrence of at least one parasite on a dragonfly in relationship to population and body length is then:

$$y_i \sim \text{Bernoulli}(p_i)$$

$$\text{logit}(p_i) = \log\left(\frac{p_i}{1-p_i}\right) = \alpha_j + \beta * body_i$$

Here, p_i is the expected proportion of dragonflies with presence of parasites on the probability scale, i.e., with a parasite load 1 or greater. This is the mean response at the level of each individual dragonfly. The linear model is applied to the logit transformation of that expected response. Thus, in a Bernoulli GLM we are modeling a proportion or probability.

```
presence <- ifelse(mites > 0, 1, 0)  # convert abundance to presence/absence
summary(fm11 <- glm(presence ~ pop-1 + body, family = binomial))
Call:
glm(formula = presence ~ pop - 1 + body, family = binomial)
[...]
Coefficients:
             Estimate Std. Error z value Pr(>|z|)
popNavarra    -17.263     10.974  -1.573    0.116
popAragon     -16.710     10.557  -1.583    0.113
popCatalonia  -17.273     10.948  -1.578    0.115
body            2.297      1.416   1.622    0.105

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 12.4766  on 9  degrees of freedom
Residual deviance:  6.6209  on 5  degrees of freedom
AIC: 14.621

Number of Fisher Scoring iterations: 5
```

There is nothing new here in terms of the model specification in R. To specify this model in BUGS, we again write almost the identical thing as in algebra:

```
for(i in 1:9){
   presence[i] ~ dbern(p[i])
   logit(p[i]) <- alpha[pop[i]] + beta * body[i]
}
```

The logit link is the typical link employed in binomial or Bernoulli GLMs. Other link functions include the probit and the complementary log-log (cloglog), which are both available in R and BUGS. They give very similar results especially in the range of $0.1 < p < 0.9$, although the `cloglog` is asymmetric about 0.5. In custom-made MCMC algorithms, the probit can enjoy considerable efficiency benefits (Dorazio & Rodriguez 2012; Johnson et al. 2012). The `cloglog` link leads to an astonishing interpretation of presence/absence data in terms of an underlying abundance; we describe this next.

3.3.7 Binomial GLM: logistic regression for bounded counts

The binomial distribution is the standard model for counts of independent and identically distributed discrete "things" that come in one of two types and which have a fixed total number

(Gelman et al. 2014). As for the Poisson, "things" is again a very generic placeholder. Type refers to binary distinctions such as "presence/absence", "male/female", "dead/alive", "here/away", "reproductive/non-reproductive" or "detected/non-detected". The binomial distribution is adopted as a model for random variables that can be considered as a sum of *N* independent Bernoulli trials, where *N* is called the sample or trial size. Typical examples may be the number of female nestlings in a nest with N nestlings, the number of individuals parasitized in a group of N individuals scored for some parasite, the number of individuals dying over a time period in a group of N individuals or the number of individuals detected among those present in a population of size N. A binomial count can never exceed the sample size, hence, is naturally bounded. This is a major distinction from Poisson counts. Note also that the binomial is *not* an adequate model for every ratio of two numbers. For instance, it would not generally be adequate for a ratio of two areas, e.g., when your interest lies in the proportion of the leaf area chewed by some beetle. The binomial is a model for the ratio of two counts only.

We will see binomial GLMs many times throughout this book, since they, or the elemental Bernoulli GLMs, are our canonical descriptions of the observation process for distribution and abundance. At the level of the observed data, most models in this book have a binomial distribution, hence, there is a sense in which they can all be considered logistic regressions, albeit, owing to the latent variables such as presence or abundance, they typically have some fairly complicated random effects structures.

For our toy dragonflies, we model as a binomial count the number of damaged wings in individual *i* from among the four wings (variable $damage_i$), in relation to population and body length:

$$damage_i \sim \text{Binomial}(N, p_i)$$

$$\text{logit}(p_i) = \log\left(\frac{p_i}{1-p_i}\right) = \alpha_j + \beta * body_i$$

Here, N = 4 is the value of the binomial sample size, corresponding to the four wings. At the elemental level of a single wing, the expected response of this binomial count is p_i; it is the expected proportion of damaged wings of an individual on the probability scale. Again, from observing counts we model an underlying probability. The linear model is applied to the logit transformation of that expected response or probability. Note, however, that the expected value of damage is N * p or the sum of N Bernoulli trials each with probability p. In R, the response of a binomial GLM with sample size greater than 1 (i.e., not Bernoulli), is specified as the pair (number of successes, number of failures).

```
summary(fm12 <- glm(cbind(damage, 4-damage) ~ pop + body -1, family = binomial))

Call:
glm(formula = cbind(damage, 4 - damage) ~ pop + body - 1, family = binomial)
[...]
Coefficients:
             Estimate Std. Error z value Pr(>|z|)
popNavarra    -4.7643     3.9351  -1.211    0.226
popAragon     -3.0047     3.6658  -0.820    0.412
popCatalonia  -4.7601     3.9255  -1.213    0.225
body           0.3837     0.4623   0.830    0.407

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 29.819  on 9  degrees of freedom
Residual deviance: 17.457  on 5  degrees of freedom
```

```
AIC: 32.833

Number of Fisher Scoring iterations: 4
```

In BUGS, a binomial GLM is specified exactly as in algebra above, except for the order of the arguments: success probability comes before the binomial sample size:

```
for(i in 1:9){
   damage[i] ~ dbin(p[i], 4)
   logit(p[i]) <- alpha[pop[i]] + beta * body[i]
}
```

3.4 Random effects (mixed) models

We previously defined a hierarchical model as a series of probability models for linked random variables, where the outcome of one random variable is observed (this is the data) and the outcome of one or more additional random variables is not observed (or only partially so) and thus latent. The outcomes of these (partially) latent random variables are called *random effects*. As we argued in chapter 2, in hierarchical models in this book random effects arise naturally under a mechanistic view of the stochastic processes that underlie the observed data on occurrence and abundance in site-structured populations. A population at site *i* has a certain state (e.g., a species is present or absent, or we have abundance state N_i), which is observed imperfectly due to the peculiar, binary kind of measurement error induced by imperfect detection. The latent occurrence or abundance state of a local population is viewed as the outcome of a stochastic process, whose features we want to model: for instance, we want to express the occurrence probability or the expected abundance as a function of environmental covariates. Hence, it is natural to treat the population state as a random effect governed by some probability distribution. Notably, this type of random effect has a clear ecological meaning as one of the fundamental states in ecology: occurrence (or the element of a species distribution) and abundance.

In contrast, in the vast majority of examples of random effects- (or mixed) models in ecology, the random effects do *not* have a clear ecological interpretation. Rather, they are merely abstract constructs invoked to explain the fact that some measurements are more similar to each other than others, i.e., to model correlations in the observed data. For instance, in a randomized-block ANOVA model, random "group effects" are introduced into the linear predictor for a response to accommodate the fact that measurements taken in the same block or group all have a tendency to be higher or lower than the overall average (Mead 1988; Littell et al. 2008). The shared group effect induces a correlation in the responses, and is treated as the outcome of a random process, typically described by a normal distribution. Thus, random effects are effects, or parameters, that one may or may not want to estimate individually, and that are given a (prior) distribution (e.g., a normal), which itself has (hyper-)parameters, such as the mean and the variance, that are estimated. We make the assumption that the group effects are *exchangeable*, which for practical purposes means independent and identically distributed.

The motivations to declare a set of effects as random, i.e., as the outcome of a stochastic process, are many and varied; see Kéry & Schaub, 2012, p. 77–82. We briefly summarize in the context of our dragonfly example from sections 3.2.1. and 3.2.2, as far as possible:

- Increased scope of inference and assessment of variability: Treating the effects of population as random allows us to make an inference not only about the three particular populations sampled, but about an entire "population of populations", from which the three populations are a sample. For instance, we can estimate the mean and the variability among the populations represented by our sample of three.
- Accounting for hidden structure in the data: The classical example is the block effect in a randomised block ANOVA or in similar random-effects, one-way ANOVA models (Mead

1988; Kéry, 2010, chapter 9). Accounting for such hidden structure accommodates the correlations that exist and prevents pseudo-replication (Hurlbert 1984).

- Partitioning of variability: Not only can we estimate the variability among populations, but we may also start to explain the differences among populations by measured covariates. Thus, whenever we want to *model* some parameters as a function of covariates, it is natural to assume that they are the outcome of a random process (i.e., random effects), the hyperparameters of which we can then model as being affected by those explanatory variables.
- Modeling of correlations among parameters: Not only can we model variability among a single set of parameters (e.g., population effects), but we can also model underlying correlations among two or more sets of parameters, such as survival and reproduction (Cam et al. 2002; Schaub et al. 2013) or among juvenile and adult survival over time (Kéry & Schaub, 2012, p. 204–208).
- Modeling of spatial, temporal or spatio-temporal correlation: Spatial or temporal autocorrelation can be accommodated by adding correlated spatial or temporal random effects, with the correlation being a function of their distance in space or time (Banerjee et al. 2004; Cressie & Wikle, 2011; also see chapter 18).
- Partial pooling, borrowing strength and 'shrinkage': Treating population effects as random can be seen as a compromise between assuming all populations are equal (complete pooling of effects; corresponding to a model without pop effects at all) on the one hand and assuming that they are totally unrelated (no pooling) on the other (Gelman, 2006; Gelman & Hill, 2007), corresponding to a model with fixed pop effects. By treating the effects as exchangeable, i.e., as similar, but not identical, the estimate of each population is not only affected by the dragonflies measured in it, but also by the other 6 dragonflies in the other two populations. In this way, the estimate for each population 'borrows strength' from the ensemble of the populations. This means that random-effects estimates of pop will not be the same as fixed-effects estimates; rather, they will be pulled in towards their overall mean and this is called 'shrinkage'.
- Combining information: Treating a set of parameters as random is a way of combining their information. For instance, parameters may be estimates from different studies. Treating them as random allows one to estimate their mean hyperparameter, which is a single measure of effect size that combines the information from all the studies (Schaub & Kéry 2012). This is called a meta-analysis.

Treating sets of parameters as random has many advantages, but should perhaps not be done by default (though see Gelman, 2007): if the assumption of exchangeability does not hold, we will mix apples and oranges and may obtain meaningless estimates of hyperparameters and hence random-effects estimates that are shrunk towards a nonsensical overall mean. Further, if interest lies in measuring the variation among random effects, a certain number is required; for instance, it does not make much sense to estimate a variance among populations in our dragonfly example with only three populations; we simply do this for illustrative purposes. To obtain an adequate estimate of the among-population heterogeneity, that is, the variance parameter, at least 5-10 populations might be required. And finally, random-effects models are typically computationally (much) more expensive, both in a frequentist and a Bayesian mode of analysis. Hence, there may be good reasons to not always treat all factor levels as random.

In the remainder of this chapter we illustrate traditional random, or mixed-effects models, first for a normal and then for a Poisson response. These are mixed-effects models because they contain both random and fixed effects, and we call them traditional, because they represent the typical motivation for random effects to account for some hidden structure in the data: by treating the population effects as random, we account for correlated measurements among dragonflies in the same population. These population effects are not something with a clear ecological interpretation, but rather some elusive tendency for some populations to be above and others to be below the population average. Since these random population effects are continuous quantities, we assume a normal distribution for them. This is what is done in the vast majority of mixed-effects models in ecology. However, as we have seen in the site-occupancy and the N-mixture models in chapter 2, random effects may be discrete and have distributions other than a normal, for instance a Bernoulli or a Poisson, respectively.

3.4.1 Random effects for a normal data distribution: normal-normal generalised linear mixed model (GLMM)

The distribution assumed for the observed random variable, i.e., the data, is often called the data distribution or observation model. We will use our toy dragonfly example to illustrate a random effects version of the linear regression model and, especially, shrinkage; compared with their fixed-effects counterparts, random effects estimates are pulled in (or 'shrunk') towards the mean of their prior distribution. Here, we revisit the ANCOVA example from section 3.2.1 and re-create the figure from before, but now without keeping track of the sex of the nine individuals. This is a normal-normal mixed or hierarchical model, because the distributions of the data and of the random population effects are both normal.

```
# Plot data without distinguishing sex
plot(body, wing, col = rep(c("red", "blue", "green"), each = 3), xlim = c(6.5,
9.5), ylim = c(10, 14), cex = 1.5, lwd = 2, frame.plot = FALSE, las = 1, pch =
16, xlab = "Body length", ylab = "Wing span")
```

The model adopted in section 3.2 assumed a linear relationship between wing span and length with a different baseline in each population and with residuals ε_i coming from a zero-mean normal distribution with variance σ^2.

$$wing_i = \alpha_j + \beta * length_i + \varepsilon_i$$
$$\varepsilon_i \sim Normal(0, \sigma^2)$$

As this model is written so far, the population effects α_j are estimated as completely unrelated numbers. On the other hand, if we assume that they are exchangeable, we can make the assumption that they are draws from a common distribution with (hyper-) parameters to be estimated. That is, we can add to the model the following assumption:

$$\alpha_j \sim Normal(\mu_\alpha, \sigma_\alpha^2)$$

This assumption is the only difference between a fixed-effects version of the "ANCOVA" and a random-effects version of it. We now use R to fit both models and plot the resulting regression lines (Fig. 3-4). For the random-effects model we use the REML method, a variant of maximum likelihood that is better suited for mixed models (McCullough & Searle, 2001; Littell et al. 2008), and fit the model with the function `lmer` in the R package lme4, which we load first. Note that in `lmer` (and function `glmer`, see below), terms appearing after the tilde (~) sign are assumed to be fixed effects except when they are between parentheses. For instance, a random intercept term for the levels of the factor `pop` is specified as (1 | pop) and a model with random intercepts *and* random slopes for each population is specified as (body | pop). By default in `lmer`, the latter defines a model where the covariance between intercepts and slopes is also an estimated parameter, i.e., where pairs of intercepts and slopes for the populations are treated as draws from a bivariate normal distribution. See below for how to specify the simpler model with two separate, univariate normal distributions, one for the intercepts and one for the slopes. Let's now fit the two models first with α_j assumed fixed.

```
summary(lm <- lm(wing ~ pop-1 + body))     # Same as fm2
[ ... ]
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
popNavarra     6.5296     1.6437   3.973  0.01061 *
popAragon      8.4003     1.5916   5.278  0.00325 **
popCatalonia   8.2630     1.6437   5.027  0.00401 **
body           0.5149     0.1991   2.586  0.04908 *
[ ... ]
```

Next, we fit the model with population effects assumed to be random. With the function `lmer`, this model is fit in the following parameterisation:

$$wing_i = \mu_\alpha + \gamma_j + \beta * length_i + \varepsilon_i$$

$$\varepsilon_i \sim Normal(0, \sigma^2)$$

$$\gamma_j \sim Normal(0, \sigma_\alpha^2)$$

Here, the mean μ_α is 'pulled out' from the random effects α_j and instead, the random effects γ_j are expressed as zero-mean deviations from the intercept μ_α; hence, $\alpha_j = \mu_\alpha + \gamma_j$.

```
library(lme4)
summary(lmm1 <- lmer(wing ~ (1|pop) + body))  # Fit the model
ranef(lmm1)                                   # Print random effects

Linear mixed model fit by REML ['lmerMod']
Formula: wing ~ (1 | pop) + body
[ ... ]
Random effects:
 Groups   Name        Variance Std.Dev.
 pop      (Intercept) 0.9861   0.9930    # This is sigma_alpha,
 Residual             0.3023   0.5498    # ... sigma,
Number of obs: 9, groups: pop, 3

Fixed effects:
            Estimate Std. Error t value
(Intercept)   7.7830     1.7034   4.569  # ... mu,
body          0.5084     0.1989   2.556  # ... beta,

> ranef(lmm1)                            # ... and these are the gamma_j.
$pop
          (Intercept)
Navarra    -1.0894238
Aragon      0.6062096
Catalonia   0.4832142
```

We recover $\alpha_j = \mu + \gamma_j$ and compare fixed- and random-effects estimates of the population-specific intercepts.

```
alpha_j <- fixef(lmm1)[1]+ranef(lmm1)$pop[,1]
cbind(fixed = coef(lm)[1:3], random = alpha_j)
                fixed   random
popNavarra   6.529633 6.693583
popAragon    8.400262 8.389216
```

```
popCatalonia 8.262966 8.266221
```

To better compare the fixed and random intercept (population) estimates, we plot the lines of best fit under the two versions of the model.

```
par(lwd = 3)
abline(lm$coef[1], lm$coef[4], col = "red", lty = 2)
abline(lm$coef[2], lm$coef[4], col = "blue", lty = 2)
abline(lm$coef[3], lm$coef[4], col = "green", lty = 2)
abline(alpha_j[1], fixef(lmm1)[2], col = "red")
abline(alpha_j[2], fixef(lmm1)[2], col = "blue")
abline(alpha_j[3], fixef(lmm1)[2], col = "green")
abline(fixef(lmm1), col = "black")
legend(6.5, 14, c("Catalonia", "Aragon", "Navarra"), col=c("blue", "green",
"red"), lty = 1, pch = 16, bty = "n", cex = 1.5)
```

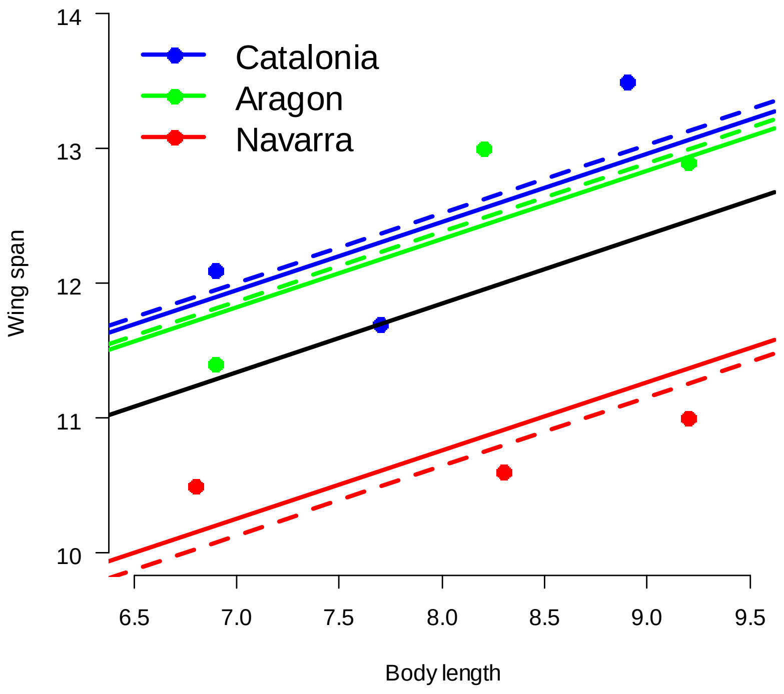


Fig. 3–4: Comparison of lines of best fit under a pop+body linear model with fixed pop effects (dashed) and random pop effects (solid), respectively. The fixed pop effects (α_j) are estimated as three completely unrelated parameters, while when assumed random, they are estimated subject to the additional assumption $\alpha_j \sim Normal(\mu_\alpha, \sigma_\alpha^2)$, i.e., as related quantities. The random-effects lines are pulled in ("shrunk") towards the grand mean, μ_α, which is represented by the solid black line.

In Fig. 3-4, you can see how the random-effects regression lines are less extreme than the fixed-effects regression lines, they are pulled in towards their grand mean (i.e., the mean hyperparameter of the assumed distribution for the random effect, represented by the black line); this is shrinkage in action. In a normal-normal random effects model, the degree of shrinkage, or pulling in towards the grand mean, depends on the ratio between the population variance σ_α^2 and the residual variance σ^2. If the population variance σ_α^2 is relatively large we don't see much shrinkage, while if the residual variance σ^2 is relatively large there is much shrinkage. Fixed-effects estimates can also be described as random-effects estimates with infinite variance hyperparameter σ_α^2.

To fit the random-intercepts model in BUGS, we write the following:

```
for(i in 1:9){       # Data model, loop over the individuals
   wing[i] ~ dnorm(mean[i], tau)
   mean[i] <- alpha[pop[i]] + beta * body[i]
}
for(j in 1:3){       # Parameter (random effects) model, loop populations
   alpha[j] ~ dnorm(mu.alpha, tau.alpha) # Mean and precision = 1/variance
}
```

We could also treat the slopes as realisations of a random variable. This would result in the following model (note that now the slope β is indexed by *j*, i.e., it can vary by population).

$$wing_i = \alpha_j + \beta_j * length_i + \varepsilon_i$$

$$\alpha_j \sim Normal(\mu_\alpha, \sigma_\alpha^2)$$ # Intercepts as random effects

$$\beta_j \sim Normal(\mu_\beta, \sigma_\beta^2)$$ # Slopes as random effects

$$\varepsilon_i \sim Normal(0, \sigma^2)$$ # Same old residual "random effects"

Another way of describing this model is that we fit three separate normal distributions, of which one has the mean fixed at zero. We can fit this model in R as follows:

```
summary(lmm2 <- lmer(wing ~ body + (1|pop) + (0+body|pop)))
```

This notation specifies a model with random intercepts (`(1|pop)`) and random slopes (`(0+body|pop)`), but without a correlation between the α_j and the β_j. For our minute data set, this complex model does not make sense, and we don't fit it here (but see exercise 3).

To fit the random-intercepts model in BUGS, we write the following:

```
for(i in 1:9){       # Data model, loop over the individuals
   wing[i] ~ dnorm(mean[i], tau)
   mean[i] <- alpha[pop[i]] + beta[pop[i]] * body[i]
}
for(j in 1:3){       # Parameter (random effects) model, loop populations
   alpha[j] ~ dnorm(mu.alpha, tau.alpha) # Model for intercepts
   beta[j] ~ dnorm(mu.beta, tau.beta)    # Model for slopes
}
```

3.4.2 Random effects for a Poisson data distribution: normal-Poisson generalised linear mixed model (GLMM)

Finally, we illustrate a traditional Poisson random effects, or mixed, model by fitting to the mite counts a Poisson generalised linear mixed model (GLMM) with random intercepts. This is a normal-Poisson mixed or hierarchical model, because the distribution of the data is Poisson and the random population effects are assumed to be draws from a normal:

$$mites_i \sim \text{Poisson}(\lambda_i)$$
$$\log(\lambda_i) = \alpha_j + \beta * length_i$$
$$\alpha_j \sim Normal(\mu_\alpha, \sigma_\alpha^2)$$

```
summary(glmm <- glmer(mites ~ body + (1|pop), family = poisson))
Generalized linear mixed model fit by maximum likelihood ['glmerMod']
 Family: poisson ( log )
Formula: mites ~ body + (1 | pop)
[ ... ]
Random effects:
 Groups Name        Variance Std.Dev.
 pop    (Intercept) 0.08535  0.2922
Number of obs: 9, groups:  pop, 3

Fixed effects:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -5.8718     2.2206  -2.644  0.00819 **
body          0.8351     0.2558   3.265  0.00110 **
```

The code in BUGS looks virtually identical to the algebraic description of the model:

```
for(i in 1:9){       # Data model, loop over the individuals
   mites[i] ~ dpois(lambda[i])
   log(lambda[i]) <- alpha[pop[i]] + beta * body[i]
}
for(j in 1:3){       # Parameter (random effects) model, loop populations
   alpha[j] ~ dnorm(mu.alpha, tau.alpha) # Mean and precision = 1/variance
}
```

3.6 Exercises

1. In 3.2.1. and 3.2.2., fit ANCOVA models with `sex` instead of `pop` and `color` instead of `body`.
2. Fit some or all of the linear models in sections 3.2.1.-3.2.4. to a Poisson, a Bernoulli and a binomial response, i.e., to the mite counts, the indicator for heavy parasitization and the damage variable (as per 3.3.1.-3.3.3.).
3. See whether you can fit a model with random intercepts and random slopes (as in 3.4.1.) to a clone of the toy data set that is 10 times larger. Here it is:

```
# Define and plot data (10 times larger data set than the toy data set)
clone.size <- 10              # clone size
pop <- factor(rep(c(rep("Navarra", 3), rep("Aragon", 3), rep("Catalonia", 3)),
levels = c("Navarra", "Aragon", "Catalonia"), clone.size))
wing <- rep(c(10.5, 10.6, 11.0, 12.1, 11.7, 13.5, 11.4, 13.0, 12.9), clone.size)
body <- rep(c(6.8, 8.3, 9.2, 6.9, 7.7, 8.9, 6.9, 8.2, 9.2), clone.size)
sex <- rep(factor(c("M","F","M","F","M","F","M","F","M"), levels = c("M", "F")),
clone.size)
mites <- rep(c(0, 3, 2, 1, 0, 7, 0, 9, 6), clone.size)
color <- rep(c(0.45, 0.47, 0.54, 0.42, 0.54, 0.46, 0.49, 0.42, 0.57),
clone.size)
damage <- rep(c(0,2,0,0,4,2,1,0,1), clone.size)
```

4. Fit the analogous Poisson GLMM to the mite counts for the larger clone of the data set.
5. Fit a random intercepts and a random intercepts and slopes model for a binomial response (for `damage`).

4. Introduction to data simulation

4.1 What do we mean by data simulation and why is it so tremendously useful ?

By data simulation, we simply mean the generation of random numbers from a stochastic process which is described by a series of distributional statements, such as $\alpha_i \sim Normal(\mu, \sigma_\alpha^2)$ and $y_{ij} \sim Normal(\alpha_i, \sigma^2)$ for a normal-normal mixed model; see section 3.4.1. Data simulation is so exceedingly useful for your work as a quantitative ecologist and, moreover, is done so frequently in this book, that we dedicate a whole chapter to it. Here is why:

1. Truth is known: We know the values of simulated parameters, and we can therefore compare the model estimates to those. Fitting a model and obtaining parameter estimates that resemble the input values is a good check that your coding in BUGS (or any other language) is right and that the algorithmic MCMC black box in BUGS is doing the right thing.
2. Calibrate derived model parameters: Sometimes the effect of one or more paramaters on the desired output of a model is unclear, such as the values of apparent survival and recruitment on the extinction probability of a population. Being able to tune the parameters such that meaningful data sets arise can be useful. Related to that, simulations can be regarded like controlled experiments, as simplified versions of a real system in order to test how varying certain parameters affects estimates of other parameters, e.g., for a sensitivity analysis (T.L. Crewe, *pers. comm.*). Controlled experiments are generally impossible in many ecological studies, and simulation is one way to look into things.
3. Sampling error can be described as the natural variability of the data and of statistics computed from them, such as sample means or parameter estimates (we find "error"a slightly misleading term). This variability arises because we have not measured every member in a heterogeneous statistical population of interest. The magnitude of sampling error determines our measures of statistical precision (e.g., standard errors, confidence intervals), which can be computed based on theory even from a single sample, i.e., from a single data set. In our experience, most ecologists find it very challenging to grasp the concept of sampling error, i.e., the natural variability among a hypothetical large collection of replicate data sets, when all you have is usually just a single replicate. Repeatedly turning the handle on the data-generating stochastic process represented by your data simulation code and observing the resulting variability among simulated data sets, or things computed from them such as parameter estimates, is a fantastic learning experience, because it allows you to actually *observe* sampling error directly.
4. Check the frequentist operating characteristics of estimators (e.g., bias, precision): It can be very useful to see how good your estimates are expected to be for given sample sizes and parameter values. To assess the estimator bias (or "is my estimate on target on average ?") or precision (or "how repeatable, or how variable around their average, are the estimates ?"), it is most straightforward to analyse a large number of simulated data sets for different choices of sample size or parameter values and compute the difference between the mean of the estimates and truth (bias) and the variance of the estimates (precision).
5. Power analyses: Power is the probability to detect an effect in the data when it is really there. Analysing a large number of simulated data sets is the most flexible way to estimate the power of a sampling design and associated analysis method. A closely related problem is the determination of necessary sample size to detect an effect of a certain magnitude with a chosen probability.
6. Check the identifiability/estimability of model parameters: The mere fact that we can fit a certain model to a certain data set and get some numbers out does not guarantee that we actually have the "right" kind of data to inform every parameter. A parameter may be intrinsically or extrinsically unidentifiable, which means that there is no information in our data set to obtain an estimate of it. Intrinsic unidentifiability refers to the case where the structure of a model does not

allow us in principle to estimate a certain parameter from a certain type of data set. A famous example in ecological statistics is in the Cormack-Jolly-Seber model with full time-dependence, where the last survival and recapture probabilities are confounded and only their product can be estimated (Kéry & Schaub, 2012, p. 217–220). With extrinsic unidentifiability we cannot estimate a certain parameter because of the vagaries of a particular data set. For example, in an interaction-effects ANOVA model, we will not be able to estimate all interaction terms if some combinations of the factor levels are not observed. Identifiability is always a worry with complex models (Cressie et al. 2009; Lele 2010) and it is compounded in Bayesian analyses where we will always obtain an estimate: when there is no information in the data set to inform the parameter estimate, the estimate will simply be determined by the prior (see 5.5.2 for a striking example). Hence, in a strictly technical sense all parameters are always identifiable in a Bayesian analysis when proper prior distributions are used. However, practically, we have exactly the same problem, because a parameter estimate entirely determined by the prior will likely be unsatisfactory to most. To make things more complicated, there are intermediates between estimability and non-estimability (Catchpole et al. 2001). For simple models or for certain classes of models a lot is known about the parameters that are intrinsically identifiable (Cole 2012) and for some, relatively well-worked out methods exist to check parameter identifiability (Catchpole & Morgan 1997; Catchpole et al. 2002; Choquet & Cole 2012; Cole 2012). However, this is not the case for the vast majority of hierarchical models, and extrinsic unidentifiability can never be known beforehand. Hence, perhaps the most straightforward approach to check estimability in practice is to generate many replicate data sets under a model for various sets of parameter values, estimate the parameters and see whether the estimates cluster around the data generating values.
6. Check for the robustness of estimators and for the effects of assumption violations: An assumption violation can be loosely defined as the presence of an "important" effect in the data-generating model and its absence in the data-analysis model. "Important" means that this absence has a noticeable effect on the quality of the desired inference. A straightforward way to gauge the robustness of our model to the violation of an assumption such as "Y does not vary among individuals" is to: generate data under the more general model where Y *does* vary among individuals and then analyse the data with the restricted model where Y does *not* vary among individuals. Repeating this a large number of times (e.g., 100–1000) will allow us to say how influential the violation of this assumption is for our inference. Often, as few as 5–10 simulation replicates may already be enough to give you a pretty good idea of the broad patterns.
7. Finally, data simulation provides proof that you understand a model: If you can simulate data under a certain model, then it is likely that you really understand that model. Simulating data sets is the opposite of analyzing a data set: you assemble a data set in the data simulation procedure and then break it down again using the analysis procedure. Analyzing data is like fixing a broken motor-bike: to be able to fix the bike you must really know all its parts and how they relate to each other. If you can take apart a motor-bike and then re-assemble the parts into a functioning bike, you prove to yourself that you understand how the bike works, and you will be better able to diagnose and fix problems when they arise. In a similar vein, if you can assemble a data set using its "ingredients" (parameters, covariates), you will certainly understand what the parameter estimates mean that come out of the analysis.

In the next section, we will use R to walk you through the generation of a data set in great detail. We build a typical point count data set, where we have one count from each of a number of sites for each of a number of surveys that are conducted over a short time period, such as a breeding season. After that, we package the essential parts of that R code into a function. Functions make it much easier to repeatedly execute the simulation code, and allow you to easily change key settings such as sample size or parameter values without altering the underlying code.

4.2 Generation of a typical point count data set

The data set we assemble is perhaps the canonical example of a count data set in nature as we see it, and for which we employ hierarchical models to learn about the features of the two processes that have generated them: the ecological and the observation models. Hence, in this chapter we also illustrate, using data simulation, the way in which observed counts arise as a result of the action of an ecological process governing spatio(-temporal) variation in abundance and of an observation process embodied by two possible forms of binary measurement error. We also emphasize that occurrence is simply an information-reduced summary of abundance, something we have seen before (e.g., in section 3.3.6). In essence, we generate a prototypical data set that embodies our view of how spatially replicated observations of the biological abundance state arise.

To be more concrete, we give a name to our imaginary species and call it a great tit (*Parus major*; Fig. 4–1), a small passerine widespread in Eurasia. We generate a data set that contains *J* replicate counts at each of *M* sites under the 'closure' assumption: that the counts at a site take place in such a short time that abundance *N* at the site does not change. See 6.9 for a typical breeding bird survey that produces data of this kind.

Fig. 4–1: The famous Great tit (*Parus major*; J. Peltomäki).

4.3 Packaging everything in a function

It can be very useful to package a simulation that you run repeatedly into a function. This will make your programming more concise and flexible, and it makes more transparent the settings used for data generation. We define a function to generate the same kind of data that we just created, assigning function arguments to any part of the simulation code that we might want to flexibly modify among simulated datasets, such as sample size, parameter values, presence/absence/magnitude of interaction terms or detection error.

```
# Function definition with set of default values
data.fn <- function(M = 267, J = 3, mean.lambda = 2, beta1 = -2, beta2 = 2,
beta3 = 1, mean.detection = 0.3, alpha1 = 1, alpha2 = -3, alpha3 = 0, show.plot
= TRUE){
#
```

```
# Function to simulate point counts replicated at M sites during J occasions.
# Population closure is assumed for each site.
# Expected abundance may be affected by elevation (elev),
# forest cover (forest) and their interaction.
# Expected detection probability may be affected by elevation,
# wind speed (wind) and their interaction.
# Function arguments:
#     M: Number of spatial replicates (sites)
#     J: Number of temporal replicates (occasions)
#     mean.lambda: Mean abundance at value 0 of abundance covariates
#     beta1: Main effect of elevation on abundance
#     beta2: Main effect of forest cover on abundance
#     beta3: Interaction effect on abundance of elevation and forest cover
#     mean.detection: Mean detection prob. at value 0 of detection covariates
#     alpha1: Main effect of elevation on detection probability
#     alpha2: Main effect of wind speed on detection probability
#     alpha3: Interaction effect on detection of elevation and wind speed
#     show.plot: if TRUE, plots of the data will be displayed;
#        set to FALSE if you are running simulations.

# Create covariates
elev <- runif(n = M, -1, 1)                          # Scaled elevation
forest <- runif(n = M, -1, 1)                        # Scaled forest cover
wind <- array(runif(n = M*J, -1, 1), dim = c(M, J)) # Scaled wind speed

# Model for abundance
beta0 <- log(mean.lambda)               # Mean abundance on link scale
lambda <- exp(beta0 + beta1*elev + beta2*forest + beta3*elev*forest)
N <- rpois(n = M, lambda = lambda)      # Realised abundance
Ntotal <- sum(N)                        # Total abundance (all sites)
psi.true <- mean(N>0)                   # True occupancy in sample

# Plots
if(show.plot){
par(mfrow = c(2, 2), cex.main = 1)
devAskNewPage(ask = TRUE)
curve(exp(beta0 + beta1*x), -1, 1, col = "red", main = "Relationship lambda-
elevation \nat average forest cover", frame.plot = F, xlab = "Scaled elevation")
plot(elev, lambda, xlab = "Scaled elevation", main = "Relationship lambda-
elevation \nat observed forest cover", frame.plot = F)
curve(exp(beta0 + beta2*x), -1, 1, col = "red", main = "Relationship lambda-
forest \ncover at average elevation", xlab = "Scaled forest cover", frame.plot =
F)
plot(forest, lambda, xlab = "Scaled forest cover", main = "Relationship lambda-
forest cover \nat observed elevation", frame.plot = F)
}

# Model for observations
alpha0 <- qlogis(mean.detection)        # mean detection on link scale
p <- plogis(alpha0 + alpha1*elev + alpha2*wind + alpha3*elev*wind)
C <- matrix(NA, nrow = M, ncol = J)     # Prepare matrix for counts
for (i in 1:J){                         # Generate counts by survey
   C[,i] <- rbinom(n = M, size = N, prob = p[,i])
}
summaxC <- sum(apply(C,1,max))          # Sum of max counts (all sites)
psi.obs <- mean(apply(C,1,max)>0)       # Observed occupancy in sample

# More plots
if(show.plot){
par(mfrow = c(2, 2))
curve(plogis(alpha0 + alpha1*x), -1, 1, col = "red", main = "Relationship p-
elevation \nat average wind speed", xlab = "Scaled elevation", frame.plot = F)
matplot(elev, p, xlab = "Scaled elevation", main = "Relationship p-elevation\n
at observed wind speed", pch = "*", frame.plot = F)
curve(plogis(alpha0 + alpha2*x), -1, 1, col = "red", main = "Relationship p-wind
speed \n at average elevation", xlab = "Scaled wind speed", frame.plot = F)
matplot(wind, p, xlab = "Scaled wind speed", main = "Relationship p-wind speed
\nat observed elevation", pch = "*", frame.plot = F)

matplot(elev, C, xlab = "Scaled elevation", main = "Relationship counts and
elevation", pch = "*", frame.plot = F)
matplot(forest, C, xlab = "Scaled forest cover", main = "Relationship counts and
forest cover", pch = "*", frame.plot = F)
matplot(wind, C, xlab = "Scaled wind speed", main = "Relationship counts and
wind speed", pch = "*", frame.plot = F)
desc <- paste('Counts at', M, 'sites during', J, 'surveys')
hist(C, main = desc, breaks = 50, col = "grey")
}

# Output
return(list(M = M, J = J, mean.lambda = mean.lambda, beta0 = beta0, beta1 =
beta1, beta2 = beta2, beta3 = beta3, mean.detection = mean.detection, alpha0 =
alpha0, alpha1 = alpha1, alpha2 = alpha2, alpha3 = alpha3, elev = elev, forest =
forest, wind = wind, lambda = lambda, N = N, p = p, C = C, Ntotal = Ntotal,
psi.true = psi.true, summaxC = summaxC, psi.obs = psi.obs))
}
```

Once we have defined the function by executing the code of its definition in R, we can call it repeatedly and send its results to the display or, more commonly, assign them to an R object, so that we can use the generated data set in an analysis.

```
data.fn()                   # Execute function with default arguments
data.fn(show.plot = FALSE) # same, without plots
data.fn(M = 267, J = 3, mean.lambda = 2, beta1 = -2, beta2 = 2, beta3 = 1,
mean.detection = 0.3, alpha1 = 1, alpha2 = -3, alpha3 = 0) # Explicit defaults
data <- data.fn()           # Assign results to an object called 'data'
```

4.4 Summary and outlook

The two main purposes served by this chapter were: to introduce data simulation and to reinforce, using R code, what we perceive as the two typical processes underlying *all* count data in ecology: an ecological process and an observation, or measurement, process. By data simulation we understand the generation of a data set as the random outcome from a defined stochastic process. We described it by statistical distributions with effects of covariates built into the processes in the manner of generalized linear models (GLM). We have given an extended version of this code, where each step was explained and commented, and an abbreviated version where the key lines are packaged into a function with arguments that can be chosen upon calling the function. Using functions is often how we conduct data simulation, but we wanted to provide the extended version first to better illustrate the great importance of data simulation for applied statistics. We have pointed out some of the important benefits of data simulation.

- When we analyse a data set using a model similar to the one used for data simulation, we can compare the resulting inferences with the known truth. This can help avoid coding errors in the analysis or diagnose problems with the MCMC algorithms. We will see this throughout the book.
- We have just seen how we can observe sampling error by repeatedly executing the data simulation code: unless using a seed, every single data set will differ and so will things calculated from the observed data, such as mean counts or the significance of a parameter estimate. Sampling error can be observed by plotting or summarizing the distribution of those variables of interest across simulated datasets.

- As we will see in 6.6 for N-mixture models and in 10.7 for site-occupancy models, we can check whether some estimator (for instance, the MLE or the posterior mean) is unbiased and gauge its precision for various sample sizes or magnitudes of measurement errors. In a similar vein, we have just seen that in the presence of imperfect detection the sum of the maximum counts over a number of sites is not an unbiased estimator of the total abundance at these sites with three surveys.
- We can do power analyses by setting a parameter of interest at a particular value and creating and analysing replicate data sets. Power is given by the proportion of times that the parameter estimate is significant at a chosen significance level. Repeating this for different sample sizes such as number of sites or surveys can give information relevant for the design of a study.
- We can check the identifiability of a parameter.
- We can check estimator robustness and gauge the effects of assumption violations. For instance, in this chapter, we could generate data including an interaction between wind and elevation on detection and then analyse the resulting data without the interaction to find out how robust estimates of the main effects are to omission of an existing interaction. In chapter 6 we will see how the N-mixture model is affected by the presence of unmodelled effects and in chapter 16, how unmodelled detection heterogeneity affects the parameter estimates in a dynamic occupancy model.
- And finally, we repeat that if you *really* understand the data simulation under a specific model, then you understand that model. Hence, if you truly understand the simulation in this chapter, then you also understand the basic N-mixture model (Royle 2004), which is the subject of chapter 6.

Just as we value the confirmation of field studies with lab experiments, we also believe that the benefits of simulating data are so great that almost any major data analysis project should be complemented with some simulation studies. Nevertheless, some may argue against use of simulated data sets: First, often ecologists appear to immediately become less interested when they know a data set is simulated rather than "real" and second, simulated data may be “too simple” compared with real data sets. Of course, we are not saying you should *only* analyse simulated data sets; we see them as providing you with practice to be better able to manage the challenges of your real data sets. One way in which simulated data in this book are simpler than real data sets is that we usually do not create missing values. Rather, we generate balanced data sets, for instance, in this chapter we assumed the same number of replicate counts at every site. Of course, when you think that a certain complication such as missing data is needed in your analysis, you should simply build it into your data-generating procedure. For instance, when you worry that uneven numbers of counts among sites may affect the quality of your estimates, you could simply turn some of your data into missing values to mimic a pattern in which there are uneven numbers of counts per site, or even sites without counts. Then you can analyse these data sets to see whether there is an effect of a certain number or pattern of missing values on the estimates.

Simulating a data set has permitted us to illustrate the salient features of ecological count data: they are generated by an ecological process and an observation process. We like to call the latter a *measurement error process* because that's exactly what it is: it introduces error into our measurement *C* of abundance *N*. Although there are two possible kinds of measurement errors for counts (false-positives and false-negatives), we have only included the more common false-negative error for now, where the error rate is given by the complement of detection probability. Almost all models introduced in later chapters accommodate this type of measurement error, but some (see chapter 19) will also accommodate false-positives. Incorporating false-positives is an active area of research in capture-recapture modeling.

Throughout this book, we will see many more examples of both topics, data simulation and the interplay of an ecological and a measurement error process underlying observed data on distribution and abundance. Specifically, in chapters 7–10, 16 and 21, you will see considerably more complex R functions to generate an even wider variety of data sets than using the function in this chapter. Data simulation is a vast field and we have only shown a very simple example: you may want to use much more complicated ways of data simulation, either to analyse the resulting data sets or to study emerging properties of a system. One important field for the latter are so-called individual- or agent-based models (e.g., Grimm, 1999; Railsback & Grimm 2012).

In chapter 5, we use our function to generate a data set of counts of great tits and analyse it to illustrate the fitting of several GLMs and simple random-effects models using WinBUGS, OpenBUGS and JAGS, as well as using standard or restricted maximum likelihood procedures in R. Then, in chapter 6, we will analyse exactly this type of data using the hierarchical N-mixture model and we will encounter a much more complex version of a data simulation function that we will use in chapters 6 and 7. If you understood the data simulation procedure in this chapter, then you fully understand the N-mixture model, which is a cornerstone of hierarchical models in this book.

4.5 Exercises

1. Sampling error: Simulate count data with p=1 and a single temporal replicate (and default function arguments otherwise). Fit the data-generating model using R function `glm` and observe how variable the estimates are (perhaps run 1000 simulation replicates). This is sampling error and is quantified by the standard error of the estimates.
2. Small-sample bias: Repeat exercise 1, but with a very small sample size, e.g., 5 or 10 sites, and inspect bias and precision of the maximum likelihood regression estimators.
3. Power analysis: With default arguments apart from p=1 and a single survey per site, what is the power to detect the effects of the effects in abundance using a Poisson GLM fit with `glm` ? Repeat with 20 sites and with 200 sites.
4. Effects of ignoring the observation process: Simulate data sets using the default arguments (but only one survey) and analyse them with a Poisson GLM with the data-generating structure in abundance. What is the effect of ignoring the measurement error (i.e., the observation process) ?
5. "Reliability" of count data (1): Run the data simulation function with a constant detection probability of 0.5 (i.e., set the coefficients for elevation and wind to zero). Does a constant measurement error of 1-(p = 0.5) standardise the counts ? Will we always count the same number of tits at a particular site ? Why/why not ? What can be done to make the counts more reliable in the sense of "less variable" ?
6. "Reliability" of count data (2): Continuing; are replicated counts that are less variable more or less reliable as an index to the local population size N ? Plot the SD of counts obtained with p = 0, 0.1, ..., 0.9, 1.
7. When is the observed maximum count an unbiased estimator of abundance ? By trial and error, check how many surveys are required in the simulation at the end of section 4.3 to make summaxC an unbiased estimator of Ntotal. Be prepared to wait for a while.
8. Relationship of detection/nondetection data to counts and of cloglog-Bernoulli model to Poisson regression: In 3.3.6, the Bernoulli model with complementary log-log link did not produce adequate estimates from the mere detection/nondetection data about the relationship between the actual mite counts and the covariates. We have claimed that this is probably due to the tiny sample size and serious departure from a Poisson. See whether the cloglog-Bernoulli model provides better estimates of an abundance relationship underlying detection/nondetection data

for a large sample size, when the count data are in fact Poisson. Use the data simulation function in this chapter for 267 sites with a single replicate and perfect detection.

```
# Generate data set and fit Poisson and Bernoulli models
data <- data.fn(M = 267, J = 1, mean.lambda = 2, beta1 = -2, beta2 = 2, beta3 =
1, mean.detection = 1, show.plot = FALSE)
summary(fmPois <- glm(C ~ elev*forest, family = poisson, data = data))
presence <- ifelse(data$C > 0, 1, 0)
summary(fmBern <- glm(presence ~ elev*forest, family = binomial(link =
"cloglog"), data = data))

# Compare Poisson and Bernoulli estimates with truth
print(cbind(Truth = c(data$beta0, data$beta1, data$beta2, data$beta3),
summary(fmPois)$coef[,1:2], summary(fmBern)$coef[,1:2]),3)
```

We see that on average, we get quite reasonable estimates from the Bernoulli model. You could also do a simulation to compare bias and precision between the Poisson and the Bernoulli model.

9. Sensitivity of trends based on counts vs. trends based on detection/nondetection (presence/absence) data (see also Pollock 2006): Simulate a population that declines at a chosen rate over a selected number of years. You can use parts of the simulation function and ignore the observation process (i.e., assume p=1). Estimate the power to detect a population decline of your choice and for a selected number of years when you use the counts (you can use 1000 simulation reps). Then, do the same for detection/non-detection data, by collapsing the counts to 0 and 1 and fitting a logistic regression. Fit both models for each simulated data set to enhance comparabilty (this is a more challenging exercise).
10. Generation of overdispersion: The variance of a Poisson random variable is equal to its mean. Use the data simulation function with the effects of all covariates set to zero and mean detection to 1 to verify this. Then, add in first one, then two and finally all three abundance covariates (keeping detection perfect all the time). Observe how the variance/mean ratio of the counts changes. What does this mean ? Are these data no longer Poisson-distributed ?
11. Data simulation for hierarchical detection/nondetection data: Modify the function such that you generate detection/non-detection (or "presence/absence") instead of abundance data, according to a site-occupancy model (see chapter 10) with logit(psi) = covariate effects, and logit(detection) = covariate effects (see also section 10–5).

5. Fitting models using the Bayesian modeling software BUGS and JAGS

5.1 Introduction

In this chapter, we introduce BUGS software (WinBUGS, OpenBUGS, JAGS) by fitting some very basic models that typically serve as components for more complex hierarchical models. We briefly outline the salient features of the BUGS language, point you to the use of WinBUGS and OpenBUGS as standalone software, but then run WinBUGS and JAGS exclusively from R to fit a series of basic linear and generalised linear models (GLMs), including two examples with random effects, one with a Poisson data distribution (Poisson GLMM) and the other with a binomial data distribution (binomial GLMM). This emphasizes the conceptual clarity enforced by the BUGS model definition language about what GLMs and random effects are. The two main groups of topics covered in this chapter are:

- Bayesian MCMC engines and BUGS software: WinBUGS, OpenBUGS and JAGS and Bayesian MCMC-based inference: model specification, posterior inference, predictions, goodness of fit, missing values, all shown through R; the great numerical resemblance between Bayesian and maximum likelihood estimates when vague priors are used in a Bayesian analysis.
- Applied statistics: Linear models, GLMs and traditional random-effects, or mixed, models, i.e., the same topics as in chapter 3, but now illustrated with BUGS, and the things you can do with them, e.g., estimation, testing, predictions, fit assessment using parametric bootstrap and posterior predictive distributions.

Clearly, the aim of this chapter is *not* to provide a detailed and comprehensive introduction to BUGS for GLMs and traditional mixed models: there are entire books dedicated to those topics (e.g., Gelman & Hill, 2007; McCarthy 2007; Ntzoufras 2009; Kéry 2010). Even less do we strive for an overview of the vast field of linear models and GLMs, about which hundreds of books must have been written. Rather, we want to give a hands-on introduction to the use of BUGS software to fit those statistical models that are probably most widely used by ecologists, and illustrate in practice the key topics dealt with in chapters 2 and 3, such as how parameter estimation, testing, prediction and model criticism are accomplished in the most widely used Bayesian software BUGS. We do expect you to have at least some applied knowledge of linear models, GLMs and random effects, as we covered them in chapter 3.

5.2 Introduction to BUGS software: WinBUGS, OpenBUGS, and JAGS

WinBUGS (Lunn et al. 2000, 2009, 2013), OpenBUGS (Thomas et al. 2006; Lunn et al. 2009, 2013) and JAGS (Plummer 2003) are generic Bayesian modeling software packages that allow you to describe almost any arbitrarily complex statistical model using the BUGS model definition language and to fit the model using MCMC techniques. The BUGS language is an ingeniously simple language, very similar to R, that lets you specify remarkably complex statistical models in a very concise and easy-to-understand way. Arguably, BUGS is currently the only software that enables ecologists without a formal training in statistics and computation to fit a very large range of fully custom statistical models with confidence. (Unfortunately, the powerful new STAN software [www.mc-stan.org; also see Gelman *et al.* (2014) and Korner *et al.* (2015)] does *not* allow discrete random effects, such as we always have when modeling distribution and abundance using site-occupancy and N-mixture models. You can fit such models in STAN, but only when you explicitly specify the integrated likelihood. Arguably, doing this yourself is beyond the level of statistical understanding of most BUGS users.). Note that when we write "BUGS", we typically mean WinBUGS or OpenBUGS, but sometimes we mean JAGS as well. We hope that the meaning of "BUGS" will become clear from the context. Functionality is very similar among the three BUGS sisters, with one major exception: WinBUGS and OpenBUGS have a module called geoBUGS for fitting spatial models to deal with spatial or temporal autocorrelation; see chapters 20 and 21. BUGS and JAGS software is freely available and does four things for you:

1. Lets you describe almost any kind of statistical model in the simple and powerful **BUGS language**.
2. Translates your BUGS language description of a statistical model into an **MCMC algorithm**.
3. **Runs the algorithm** for as long as you wish (or have time to wait) and thereby accumulate the samples of the desired joint posterior distribution of all unknown quantities in your analysis.
4. Allows some processing of results, such as graphical or tabular posterior summaries and convergence assessment (not for JAGS).

WinBUGS and OpenBUGS exist as Windows standalone applications, and OpenBUGS is also available for Unix/Linux and Mac operating systems. They are available through each program's website (WinBUGS: www.mrc-bsu.cam.ac.uk/software/bugs/; OpenBUGS: www.openbugs.net/w/FrontPage). They come with a comprehensive hypertext manual and several volumes of worked and richly commented example analyses. Both manuals and example volumes are extremely rich in information, but it is fair to say that this wealth of information is not easy to navigate, most of all, because there is no index available. Development of WinBUGS ceased about 10 years ago and the developmental branch of the BUGS project has moved over to OpenBUGS. However, at the time of writing, OpenBUGS does not seem to have evolved very much beyond the capabilities of the original WinBUGS, in terms of the capabilities of the programming language or the speed of the MCMC algorithms. JAGS is developed by Martyn Plummer at the International Agency for Research on Cancer in Lyon/France. It does not have a Windows user interface, and its manual and other relevant things can be downloaded from sourceforge.net/projects/mcmc-jags/files/. For all three, there is a tremendous amount of documentation freely available on the web (try googling "BUGS and X", where X is your favourite model); also, there are user groups with a discussion list.

In this book, we use WinBUGS and JAGS, but note (and show in one example) that OpenBUGS can be run from R in exactly the same way as WinBUGS. We don't show how to run BUGS as a standalone program, because most people use it from R and because use of the standalone is explained in many books, including Woodworth (2004), McCarthy (2007), Ntzoufras (2009), Kéry (2010) and Woodward (2012). To run BUGS from R, you may use a number of packages including R2WinBUGS or R2OpenBUGS (Sturtz et al. 2005), BRugs (Thomas et al. 2006), R2jags (Su and Yu 2012), rjags (Plummer), dclone (Solymos 2010), jagsUI (Kellner 2015) or runjags (Denham 2013). We use R2WinBUGS for WinBUGS and jagsUI for JAGS, though we illustrate use of rjags once in section 5.4. The new package jagsUI has similar capabilities as R2jags, its output is very similar to that of R2WinBUGS, and it avoids some problems in R2jags (as of mid-2014, the latter does not adequately separate the MCMC adaptation and burnin phases). Package runjags has extensive facilities for parallel-processing with JAGS; something that can be attractive given the sometimes extremely long run times of MCMC analyses. Packages R2jags, jagsUI and dclone also have some facilities for running JAGS and/or WinBUGS in parallel.

Even though the three BUGS engines are totally separate programs and JAGS is an entirely different enterprise, they use mere dialects of the same BUGS language. So what is the BUGS language ? In a nutshell:

- The BUGS language serves to specify statistical models
- It resembles R very much, but is not identical

- There are "nodes" in statistical models: broadly, these are model quantities such as parameters, latent variables, predictions, missing or observed data
- There are exactly two kinds of relations among nodes in a model: deterministic (represented by the arrow operator "<-", *not* the equal sign) or stochastic (represented by the tilde "~")

    All models that can be fitted in BUGS may be represented mathematically as so-called directed acyclic graphs (DAGs; Gilks et al. 1994; Lunn et al. 2000, 2013)

- There is a moderate array of mathematical and statistical functions, see the manuals.
- There is a range of inbuilt statistical distributions, see the manuals; new distributions may be defined using the "zeros trick" or the "ones trick", provided you know how to write the likelihood explicitly (Lunn et al. 2013); see 5.8 below.
- The BUGS language is not really vectorized: we need loops to define the model for every element in vectors or multi-dimensional arrays. Indexing then becomes important: we use one or multiple indices to address elements of vectors or multidimensional arrays. (Note though that some vector or matrix operations are available, such as `inprod`.)
- BUGS is a declarative language, hence the order of statements does not matter, with the only main exception being what you put inside or outside of a loop.

This already concludes our very brief first introduction to BUGS software and the BUGS language. We hope that it suffices as a simple prelude to seeing BUGS "in action" for a series of basic and important statistical models in the remainder of this chapter (and in the rest of the book). We believe that learning by doing (or by watching others do), is the most powerful and quick way of learning a statistical programming language. Thus, in the remainder of this chapter we will fit a range of linear, generalised linear and random-effects (= hierarchical) models that are typical in the applied work of ecologists and indeed most empirical scientists. We illustrate, using BUGS, each type of model described in chapter 3 using a data set of simulated counts of great tits (from chapter 4), those being:

(1) Multiple linear regression with normal errors (normal GLM)

(2) Analysis of covariance (ANCOVA) model with normal errors (normal GLM)

(3) ANCOVA model with Poisson errors (Poisson GLM)

(4) ANCOVA model with binomial errors (logistic regression or binomial GLM)

(5) Poisson mixed model or Poisson generalized linear mixed model (Poisson GLMM)

(6) Binomial mixed model or binomial generalized linear mixed model (binomial GLMM)

In this book, the normal distribution takes on a much less prominent role than in many statistics books, because we deal with models for counts. Counts are discrete and positive-valued, and hence, the normal is arguably an inadequate choice for describing such data, unlike for more conventional kinds of ecological data, such as measurements on a continuous scale. Nevertheless, we will start by briefly illustrating the fitting of a model with normal response for the sake of illustration and because most ecologists are familiar with a normal linear model. We do not necessarily endorse the application of every model in this chapter to the particular type of simulated data. However, we like the idea of using a single data set to illustrate all models. Moreover, one can frequently see in the ecological literature all the kinds of analyses shown in this chapter, e.g., the normal distribution is still quite often adopted for count data (see e.g., state-space models, Buckland et al. 2004, 2007, chapter 5 in Kéry & Schaub 2012).

The building blocks of much of what we do in hierarchical modeling of abundance, occurrence and species richness are Poisson and binomial GLMs. We describe in the BUGS language some prototypical components of hierarchical models, normal, Poisson and binomial GLMs, as well as the specification of random effects. Almost all hierarchical models in the remainder of this book are composed of these building blocks. We also illustrate the fitting of the same models using maximum likelihood, to show how numerically similar Bayesian and maximum likelihood estimates are for reasonable sample sizes and when vague priors are adopted in a Bayesian analysis. (Technically, we use least-squares or iteratively reweighted least-squares for linear models and GLMs, but the resulting estimates are equivalent to MLEs.)

5.3 Linear model with normal response (normal GLM): multiple linear regression

The normal distribution does not take on a prominent role in many analyses of counts, since counts are neither continuous nor can they be negative (though the normal becomes a better approximation when the counts become large). In a data set created using the function in chapter 4, we model the mean count of great tits as coming from a normal distribution. This model is a special case of a GLM with normal distribution and identity link, where we directly model the mean tit count as a linear function of elevation and forest cover.

$$Cmean_i \sim Normal(\mu_i, \sigma^2)$$

$$\mu_i = \alpha_0 + \alpha_1 * elev_i + \alpha_2 * forest_i + \alpha_3 * elev_i * forest_i$$

```
# Generate data with data.fn from chapter 4
set.seed(24)
data <- data.fn()
str(data)
attach(data)

# Summarize data by taking mean at each site and plot
Cmean <- apply(C, 1, mean)
par(mfrow = c(1,3))
hist(Cmean, 50)               # Very skewed
plot(elev, Cmean)
plot(forest, Cmean)
```

To fit a model in `WinBUGS` or JAGS run from R and using package `R2WinBUGS` (Sturtz et al. 2005) and `jagsUI` (Kellner 2015; for package `rjags`, see 5.4), we must first prepare all the "ingredients" of the analysis. Then, we use the functions `bugs` or `jags` to ship them over to BUGS and thereby instruct the BUGS engines on how to run the analysis. After the desired number of MCMC draws have been produced by BUGS, a long list of samples from the joint posterior distribution is imported back into R and summarized in various convenient ways. This list of MCMC samples is also contained in the `coda.txt` files produced by `WinBUGS`. We strive for a consistent layout of all these steps leading to an analysis in BUGS. We do this here for the first time and hence give much more detail than in later examples.
We need to prepare the following objects: data set, model, a function for initial values for the Markov chains, a list of parameters that we want to estimate and the MCMC settings (how many chains, for how long etc.). The first thing we prepare is a list containing the data to be analysed. Importantly, for `WinBUGS` we must only include data that are used in the model definition below, while for JAGS, the data package may also include data not used in the model.

```
# Package the data needed in a bundle
win.data <- list(Cmean = Cmean, M = length(Cmean), elev = elev, forest = forest)
str(win.data)                       # Check what’s in win.data
```

Next, we use the `cat` function to write into the R working directory a named text file containing the model description in the BUGS language. The name of this text file is up to you (though ideally should be shorter than the one we use here…).

```
# Write text file with model description in BUGS language
cat(file = "multiple_linear_regression_model.txt",
"   # --- Code in BUGS language starts with this quotation mark ---
model {

# Priors
alpha0 ~ dnorm(0, 1.0E-06)            # Prior for intercept
alpha1 ~ dnorm(0, 1.0E-06)            # Prior for slope of elev
alpha2 ~ dnorm(0, 1.0E-06)            # Prior for slope of forest
alpha3 ~ dnorm(0, 1.0E-06)            # Prior for slope of interaction
tau <- pow(sd, -2)                    # Precision tau = 1/(sd^2)
sd ~ dunif(0, 1000)                   # Prior for dispersion on sd scale

# Likelihood
for (i in 1:M){
   Cmean[i] ~ dnorm(mu[i], tau)       # dispersion tau is precision (1/variance)
   mu[i] <- alpha0 + alpha1*elev[i] + alpha2*forest[i] +
alpha3*elev[i]*forest[i]
}

# Derived quantities
for (i in 1:M){
   resi[i] <- Cmean[i] - mu[i]
}
}"#  --- Code in BUGS language ends on this line ---
)
```

This is the first place we meet a full model written in the BUGS language, hence, a couple of comments are in order. First, you must be absolutely clear about which code is R and which is BUGS. The way we write our code, everything between the quotes as indicated by the comments is in the BUGS language and everything outside is R. The purpose of the R code is simply to write the BUGS model code to a text file. As far as R is concerned, the BUGS model file is simply a large character vector. Shortly we will ask `WinBUGS` or `JAGS` to try to understand and do something useful with the outputted character vector.

In the BUGS model, you see the following quantities (nodes): `alpha0`, `alpha1`, `alpha2`, `alpha3`, `tau`, `sd`, `mu`, and `Cmean`, `elev` and `forest`. The relationship between the response `Cmean` (the observed mean count of great tits) and the mean and dispersion of the normal random variable, `mu` and `tau`, is a stochastic one, therefore they are connected with a tilde (~), i.e., `Cmean` is drawn from a normal distribution with mean `mu` and dispersion `tau`. In contrast, the expected mean count (`mu`) is connected with covariates `elev` and `forest` in a deterministic manner, therefore we use the arrow operator (`<-`), which is equivalent to an equal sign in the BUGS language. We also see two functions: the power and the normal distribution function (a list of the functions available in BUGS and `JAGS` are found in the manuals). Finally, we use a loop to describe the statistical model for the relationship between the covariates and the mean counts of great tits for each of the M sites in turn, with each data point indexed by `i`. We could have defined the priors after the loop over M (which is the likelihood), but not inside of that loop. Finally, in BUGS the normal distribution is defined in terms of the precision, which is the reciprocal of the variance. Hence, a prior with very small precision such as 1 over 1 million has a large standard deviation of 1000 and is pretty flat over a very large range of possible values of a parameter or where the likelihood has support (see also 5.5.2).

We try to keep the two main parts of the Bayesian analysis separate, namely the likelihood, which describes the relationship between the observed data and the unknown parameters, and the priors, which describe our knowledge about these parameters using a probability distribution. (Note this distinction becomes blurred in hierarchical models, as you will see later.) Priors must be specified for the primary unknown quantities in a Bayesian analysis, here, the four regression parameters, `alpha0` through `alpha3`, and the variance parameter, `tau` or `sd`. We typically use suitably wide uniform distributions or "flat" normal distributions with a small precision (corresponding to a large variance) to specify our wish to let only (or mainly) the data influence the parameter estimates. To be vague, a prior needs to be approximately flat only over the range of values where the likelihood has support, i.e., where the latter has values that are effectively non-zero. Whether or not a prior is vague for a given data set and model must be ascertained by trial and error for every new data set or model anew in a kind of informal sensitivity analysis: you make a guess about what is a vague prior, fit the model and then make the priors wider and refit the model and if the posterior distributions don't change, then you have specified vague priors.

Next, we define a function that generates random initial values with some dispersion for at least some of the parameters. We don't need to give initials for each parameter, since BUGS randomly initialises parameters for which no initial values are given based on the prior defined in the model. At the beginning, you may find the distinction between priors and initial values confusing, however, the two are fundamentally different. Priors are part of the model when you analyse it using Bayesian methods and they always affect the estimates (the posterior distributions), even if often only very slightly. In contrast, initial values are *not* part of the model; they simply represent starting points for the Markov chains. After chain convergence, initial values no longer have any effect on the properties of the posterior samples, and they, along with all chain values until convergence, are discarded as an MCMC "burn-in".

Even though initial values do not affect the posterior distribution, they can be of great practical importance especially for complex models. If we start the chains at too wildly improbable values for a parameter, the Markov chains may never find their way to where the likelihood does have some support, i.e., we may never get convergence. Or the MCMC algorithm may not start when unsuitable initial values are chosen; see chapters 14–15 for particularly acute examples of this. On the other hand, to gauge convergence we want the initial values of multiple chains to be "overdispersed", i.e., to start at reasonably different places. If the chains move to the same parameter space, we have much greater confidence in chain convergence. Therefore, we must strike a balance in the generation of initial values between not enough and too much overdispersion.

Finally, initial values must not contradict the priors, the model or the data. For instance, initialising a variance at a negative value would make BUGS crash. Similarly, in an occupancy model (see chapter 10), if we initialise at zero (corresponding to an "absence") the latent presence/absence state z for a site where a species was detected, JAGS will crash (though not `WinBUGS`). This is a contradiction between model and data in the traditional model, which does not allow false-positives. Picking appropriate initial values is easy for simple models but may become increasingly hard for more complex models (sometimes very much so). This is particularly so for JAGS, which does not forgive *any* contradictions. For complex models, there may thus be considerable trial and error involved in picking initial values.

```
# Initial values (have to give for at least some estimands)
inits <- function() list(alpha0 = rnorm(1,0,10), alpha1 = rnorm(1,0,10), alpha2
= rnorm(1,0,10), alpha3 = rnorm(1,0,10))
```

The next object is a list with the parameters we want to estimate. While all unknown quantities described in the model are internally updated (i.e., estimated), the MCMC samples are only saved in the output for those in this list.

```
# Parameters monitored (i.e., for which estimates are saved)
params <- c("alpha0", "alpha1", "alpha2", "alpha3", "sd", "resi")
```

The final thing is to decide on the number of chains (nc), their length or number of iterations (ni), the length of the initial burnin period (nb) and whether we want to discard any intermediate values ("thin") to produce a smaller, but more information-dense sample from the posterior. For instance, setting `nt = 10` we keep only every tenth value, which may be useful for saving disk space in very parameter-rich models. (However, there is no need to thin, since we always toss out information; McEachern & Berliner, 1994, Link & Eaton, 2012). The MCMC settings are also chosen by trial and error: when developing an analysis, you will often start with extremely short chains to ensure that everything is programmed fine. Then, you may run the chains longer to assess the necessary burnin length, and finally run a production run that includes the required burnin length and a sufficiently large post-burnin sample:

```
# MCMC settings
ni <- 6000    ;   nt <- 1   ;   nb <- 1000   ;  nc <- 3
```

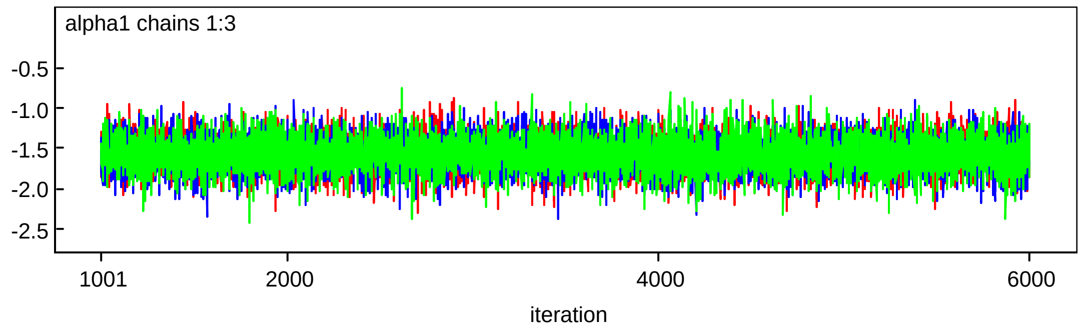
To observe convergence you could choose the following MCMC settings.

```
# ni <- 10    ;   nt <- 1   ;   nb <- 0   ;  nc <- 8 # not run
```

Now we have created all the objects that we need to use the `bugs` or `jags` functions to run an analysis. After loading the R2WinBUGS package, we run the analysis in `WinBUGS` first (check `?bugs` if you need information about the last three function arguments).

```
# Call WinBUGS from R (approximate run time (ART) <1 min)
library(R2WinBUGS)
bugs.dir <- "C:/WinBUGS14/"          # Place where your WinBUGS installed
out1B <- bugs(win.data, inits, params, "multiple_linear_regression_model.txt",
n.chains = nc, n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE,
bugs.directory = bugs.dir, working.directory = getwd())
```

The first thing you do should always be to visually inspect the trace plots to gauge convergence in `WinBUGS` (see Fig. 5-1 for a sample). Then, we exit `WinBUGS` manually and look at the results produced by R2WinBUGS in an overview. NOTE (this is important): with `debug = TRUE`, you have to manually exit `WinBUGS` before the BUGS results are imported into R. Before you do this, the R window is frozen.



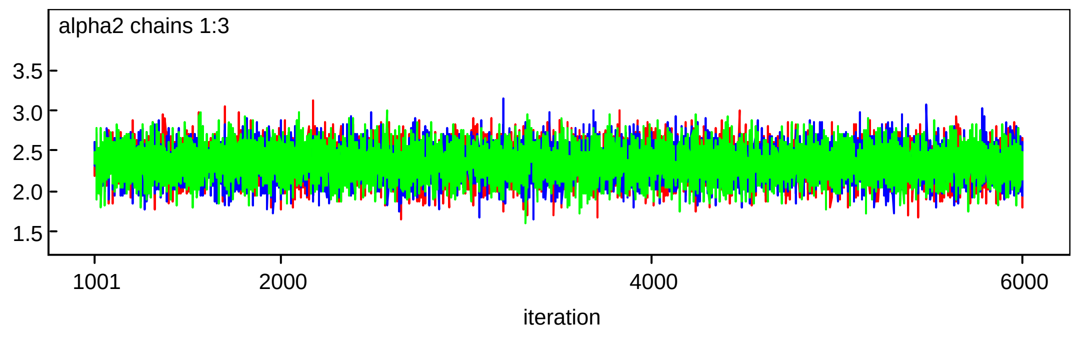


Fig. 5-1: Trace plots produced by WinBUGS (when debug = TRUE) serve to visually check for convergence, here shown for two slope parameters in the multiple linear regression model. Oscillations around a horizontal level indicate convergence.

If convergence has not been reached then we either repeat with longer chains and greater burnin, or manually do an additional burnin in R by tossing out additional draws at the start of the chains. You will only do the latter when a model takes very long to run, otherwise it is much easier to simply rerun the model.

The R object created is huge and contains many elements, of which the most frequently used are perhaps "sims.array", "summary", "mean" and "sd". Here are two overviews of the object.

```
# Overview of the object created by bugs
names(out1B)
> names(out1B.1)
 [1] "n.chains"        "n.iter"          "n.burnin"
 [4] "n.thin"          "n.keep"          "n.sims"
 [7] "sims.array"      "sims.list"       "sims.matrix"
[10] "summary"         "mean"            "sd"
[13] "median"          "root.short"      "long.short"
[16] "dimension.short" "indexes.short"   "last.values"
[19] "isDIC"           "DICbyR"          "pD"
[22] "DIC"             "model.file"      "program"

str(out1B, 1)
List of 24
 $ n.chains       : num 3
 $ n.iter         : num 6000
 $ n.burnin       : num 1000
 $ n.thin         : num 1
 $ n.keep         : num 5000
 $ n.sims         : num 15000
 $ sims.array     : num [1:5000, 1:3, 1:273] 2.5 2.26 2.19 2.4 2.26 ...
  ..- attr(*, "dimnames")=List of 3
 $ sims.list      :List of 7
 $ sims.matrix    : num [1:15000, 1:273] 2.24 2.36 2.59 2.35 2.27 ...
  ..- attr(*, "dimnames")=List of 2
 $ summary        : num [1:273, 1:9] 2.36 -2.38 3.4 -1.98 2.11 ...
  ..- attr(*, "dimnames")=List of 2
 $ mean           :List of 7
 $ sd             :List of 7
 $ median         :List of 7
 $ root.short     : chr [1:7] "alpha0" "alpha1" "alpha2" "alpha3" ...
 $ long.short     :List of 7
 $ dimension.short: num [1:7] 0 0 0 0 0 1 0
 $ indexes.short  :List of 7
 $ last.values    :List of 3
 $ isDIC          : logi TRUE
 $ DICbyR         : logi FALSE
 $ pD              : num 4.98
 $ DIC             : num 1159
 $ model.file      : chr "multiple_linear_regression_model.txt"
 $ program         : chr "WinBUGS"
 - attr(*, "class")= chr "bugs"
```

Before we look any further into the results, we fit the same model in OpenBUGS using the bugs function in the R2OpenBUGS package and finally in JAGS using the function jags in the package jagsUI (Kellner 2015). For both we can use exactly the same ingredients that we just prepared for the analysis in WinBUGS.

```
# Call OpenBUGS from R (ART <1 min)
library(R2OpenBUGS)
out1OB <- bugs(data=win.data, inits=inits, parameters.to.save = params,
model.file = "multiple_linear_regression_model.txt", n.chains = nc, n.thin = nt,
n.iter = ni, n.burnin = nb, debug = TRUE, working.directory = getwd())
detach("package:R2OpenBUGS", unload=T)  # Otherwise R2WinBUGS is 'masked'

# Call JAGS from R (ART <1 min)
library(jagsUI)
?jags                 # Look at main function
out1J <- jags(win.data, inits, params, "multiple_linear_regression_model.txt",
n.chains = nc, n.thin = nt, n.iter = ni, n.burnin = nb)
```

We can easily run JAGS on multiple cores by setting the argument parallel = TRUE:

```
out1J <- jags(win.data, inits, params, parallel = TRUE,
"multiple_linear_regression_model.txt", n.chains = nc, n.thin = nt, n.iter = ni,
n.burnin = nb)
```

For models with long run times (several hours), this will speed up calculations by up to a factor of 3 when three cores are used (though usually the gain in speed is much less). Shorter calculations may actually take longer, as the time to set up the parallel calculations outweighs the time saved by using multiple cores. The disadvantage of running JAGS in parallel is that the famous JAGS progress bar is no longer visible, thus diminishing our sense of accomplishment as we stare at our computer monitor.

For JAGS, we use the function traceplot to gauge convergence, either for a subset or for all parameters monitored.

```
par(mfrow = c(3,2))
traceplot(out1J, param = c('alpha1', 'alpha2', 'resi[c(1,3, 5:6)]')) # Subset
# traceplot(out1J)                     # All params
```

The object created by jagsUI is very similar to the one created when we run R2WinBUGS.

```
# Overview of object created by jags()
names(out1J)
 [1] "sims.list"   "means"       "sd"          "q2.5"        "q25"
 [6] "q50"         "q75"         "q97.5"       "overlap0"    "f"
[11] "Rhat"        "n.eff"       "pD"          "DIC"         "summary"
[16] "samples"     "modfile"     "model"       "parameters"  "mcmc.info"
[21] "run.date"    "random.seed" "parallel"    "bugs.format"
```

Hence, using the R packages R2WinBUGS, R2OpenBUGS and jagsUI we can readily switch between WinBUGS, OpenBUGS and JAGS for model fitting and most of the time, no change at all is required. Next, we summarize the posterior distributions. Up to Monte Carlo error, estimates from BUGS and JAGS at convergence should be identical.

```
# Summarize posteriors from WinBUGS run
print(out1B, 2)
Inference for Bugs model at "multiple_linear_regression_model.txt", fit using
WinBUGS,
 3 chains, each with 6000 iterations (first 1000 discarded)
 n.sims = 15000 iterations saved
             mean   sd    2.5%     25%     50%     75%   97.5% Rhat n.eff
alpha0       1.66 0.12    1.44    1.58    1.66    1.74    1.89    1 15000
alpha1      -1.58 0.21   -1.98   -1.71   -1.58   -1.44   -1.17    1 15000
alpha2       2.35 0.20    1.95    2.21    2.35    2.48    2.73    1 15000
alpha3      -0.85 0.35   -1.55   -1.09   -0.86   -0.62   -0.15    1  5500
sd           1.86 0.08    1.71    1.80    1.86    1.91    2.03    1 15000
resi[1]     -0.48 0.17   -0.81   -0.59   -0.48   -0.37   -0.15    1 13000
resi[2]     -0.45 0.19   -0.82   -0.58   -0.45   -0.32   -0.08    1 11000
resi[3]      1.05 0.27    0.51    0.86    1.05    1.23    1.59    1 11000
[ output truncated ]
resi[265]   -0.61 0.13   -0.88   -0.70   -0.61   -0.52   -0.34    1 15000
resi[266]   -1.82 0.12   -2.06   -1.90   -1.82   -1.74   -1.58    1 15000
resi[267]    1.28 0.45    0.39    0.97    1.27    1.58    2.16    1 13000
deviance  1087.15 3.23 1083.00 1085.00 1086.00 1089.00 1095.00    1 15000

For each parameter, n.eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor (at convergence, Rhat=1).

DIC info (using the rule, pD = Dbar-Dhat)
pD = 5.0 and DIC = 1092.2
DIC is an estimate of expected predictive error (lower deviance is better).

# Summarize posteriors from JAGS run
print(out1J, 2)
JAGS output for model 'multiple_linear_regression_model.txt', generated by
jagsUI.
Estimates based on 3 chains of 6000 iterations,
burn-in = 1000 iterations and thin rate = 1,
yielding 15000 total samples from the joint posterior.
MCMC ran in parallel for 0.054 minutes at time 2014-11-11 14:58:27.

             mean   sd    2.5%     50%   97.5% overlap0    f Rhat n.eff
alpha0       1.66 0.11    1.44    1.66    1.88    FALSE 1.00    1 15000
alpha1      -1.58 0.20   -1.97   -1.58   -1.18    FALSE 1.00    1 15000
alpha2       2.34 0.20    1.96    2.34    2.73    FALSE 1.00    1 15000
alpha3      -0.85 0.36   -1.55   -0.85   -0.14    FALSE 0.99    1 10597
sd           1.86 0.08    1.71    1.86    2.03    FALSE 1.00    1  9993
resi[1]     -0.48 0.17   -0.80   -0.48   -0.15    FALSE 1.00    1 15000
resi[2]     -0.45 0.19   -0.82   -0.44   -0.08    FALSE 0.99    1 13961
resi[3]      1.05 0.28    0.50    1.05    1.60    FALSE 1.00    1 15000
[ output truncated ]
resi[265]   -0.61 0.13   -0.87   -0.61   -0.35    FALSE 1.00    1 15000
resi[266]   -1.82 0.12   -2.06   -1.82   -1.58    FALSE 1.00    1 15000
resi[267]    1.28 0.45    0.40    1.27    2.15    FALSE 1.00    1 15000
deviance  1087.09 3.18 1082.89 1086.45 1094.82    FALSE 1.00    1 15000

Successful convergence based on Rhat values (all < 1.1).
Rhat is the potential scale reduction factor (at convergence, Rhat=1).
For each parameter, n.eff is a crude measure of effective sample size.

overlap0 indicates if 0 falls within the 95% credible interval for the
parameter.
f represents the proportion of a parameter's posterior distribution with the
same
sign as the mean; i.e., our confidence that the parameter is positive or
negative.
```

```
DIC info: (pD = var(deviance)/2)
pD = 5.1 and DIC = 1092.15
DIC is an estimate of expected predictive error (lower is better).
```

The first thing to check in the work flow of our analysis is the penultimate column. It shows the value of the Gelman-Rubin convergence diagnostic `Rhat`, which is 1 at convergence (Gelman and Rubin 1992); values below 1.1 or 1.2 are usually taken as indicating convergence, but not necessarily for producing posterior summaries with sufficiently low Monte Carlo error (for which, say, 1.01 or 1.001 might be better). The chains for different parameters converge at different rates and it is entirely possible to obtain an `Rhat` value that indicates convergence for one parameter, but not for another one. Similarly, `n.eff` typically varies strongly between different parameters and often even between replicate runs of an analysis. It is the estimated size of an equivalent, independent sample that contains the same amount of information as does your dependent sample. Chains with stronger autocorrelation are associated with higher values of `Rhat` and lower values of `n.eff`. You can visually check the autocorrelation for the first five parameters and up to lag `n` by typing this (though in this simple model, there is almost no autocorrelation):

```
n <- 10                  # maximum lag
par(mfrow = c(2, 3), mar = c(5,4,2,2), cex.main = 1)
for(k in 1:5){
   matplot(0:n, autocorr.diag(as.mcmc(out1B$sims.array[,,k]), lags = 0:n),
      type = "l", lty = 1, xlab = "lag (n)", ylab = "autocorrelation",
      main = colnames(out1B$sims.matrix)[k], lwd = 2)
   abline(h = 0)
}
```

Note also that the different BUGS engines are liable to perform very differently in terms of the MCMC efficiency (as measured by `n.eff`, or more accurately, `n.eff` per time unit). Thus, you should expect that sometimes one of the BUGS engines might not mix well, or maybe not even work for a certain class of model. For this reason, we think it is always a good idea to try more than one BUGS engine on a model, at least in the exploratory stage of an analysis.

To continue our discussion of the model output; the first two columns in the posterior summary contain the posterior means and standard deviation, which can be used for a Bayesian point estimate and an analogue to the standard error of the estimate in a frequentist analysis. Columns 3–7 show the percentiles of the posterior samples, of which the 2.5% and 97.5% form a customary 95% Bayesian confidence interval (often called a credible interval and abbreviated CRI). At the bottom of the tables we obtain the effective number of parameters (pD) and the value of the deviance information criterion (DIC; Spiegelhalter et al. 2003), which for non-hierarchical models can be used for model selection in the same way as the Akaike's information criterion (Burnham and Anderson, 2002). Unfortunately, for hierarchical models with strongly non-normal posterior distributions of random effects and for those with discrete random effects, such as occupancy and N-mixture models and hence, for essentially all models in this book, the DIC is not appropriate for model selection.

We can easily get various graphical overviews for both analyses (Fig. 5-2).

```
plot(out1B)              # For WinBUGS analysis from R2WinBUGS
plot(out1J)              # For JAGS analysis from jagsUI

par(mfrow = c(1, 2), mar = c(5,4,2,2), cex.main = 1)
whiskerplot(out1J, param = c('alpha0', 'alpha1', 'alpha2', 'alpha3', 'sd',
'resi[c(1,3, 5:7)]'))    # For JAGS analysis from jagsUI
library(denstrip)      # Similar, but more beautiful, with package denstrip
```

```
plot(out1J$alpha0, xlim=c(-4, 4), ylim=c(1, 5), xlab="", ylab="", type="n", axes
= F, main = "Density strip plots")
axis(1)
axis(2, at = 1:5, labels = c('alpha0','alpha1','alpha2','alpha3','sd'), las = 1)
abline(v = c(-4,-2,2,4), col = "grey")  ;  abline(v = 0)
for(k in 1:5){
   denstrip(unlist(out1J$sims.list[k]), at = k, ticks = out1J$summary[k,
c(3,5,7)])
}
```

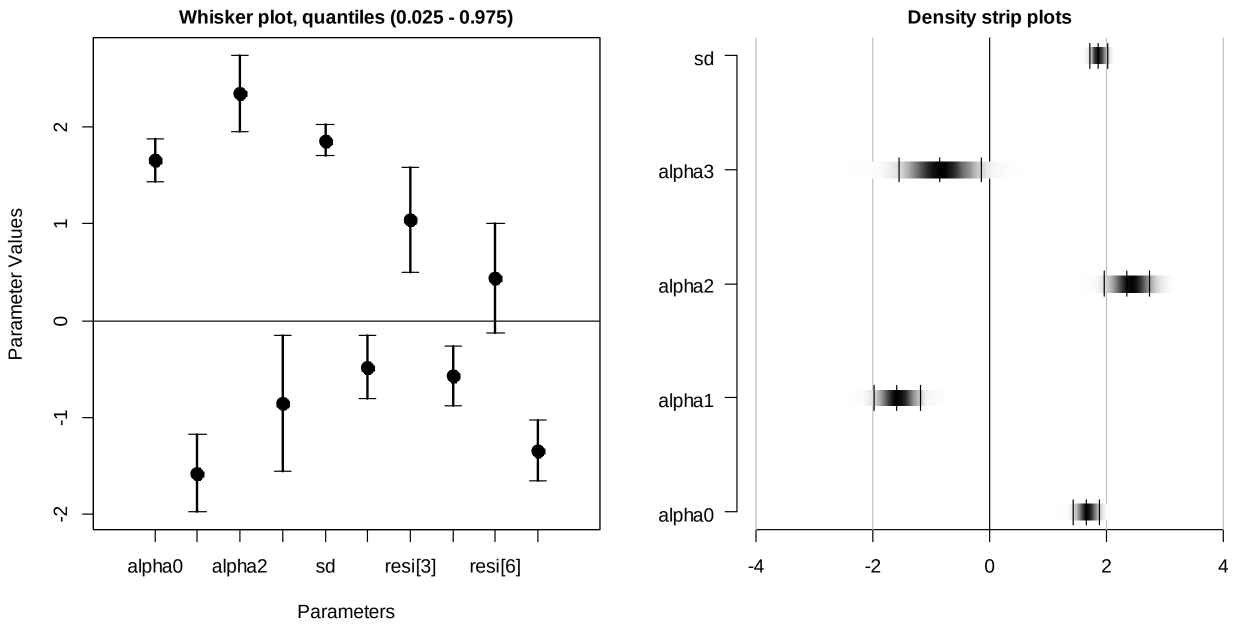


Fig. 5-2: Two useful ways of graphically summarising the posterior samples using `whiskerplot` (left) and `denstrip` (right).

For comparison, let's fit the same model using least-squares, which for normal models yields estimates that are identical to those obtained using the more general maximum likelihood method.

```
(fm <- summary(lm(Cmean ~ elev*forest)))

Call:
lm(formula = Cmean ~ elev * forest)
[ ... ]
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   1.6603     0.1137  14.607  < 2e-16 ***
elev         -1.5765     0.2029  -7.771 1.76e-13 ***
forest        2.3440     0.1977  11.857  < 2e-16 ***
elev:forest  -0.8507     0.3540  -2.403    0.017 *
[ ... ]
Residual standard error: 1.849 on 263 degrees of freedom
Multiple R-squared:  0.447,     Adjusted R-squared:  0.4407
F-statistic: 70.86 on 3 and 263 DF,  p-value: < 2.2e-16
```

We observe numerically very similar estimates (posterior means and sd from `WinBUGS` and from `JAGS` and MLEs with SEs from `lm` in R; note no SE is given for the dispersion parameter using `lm` in R).

```
print(cbind(out1B$summary[1:5, 1:2], out1J$summary[1:5, 1:2],
rbind(fm$coef[,1:2], c(fm$sigma, NA))), 4)
```

```
          mean      sd   mean      sd Estimate Std. Error
alpha0  1.6605 0.11523  1.660 0.11334   1.6603     0.1137
alpha1 -1.5765 0.20665 -1.575 0.20164  -1.5765     0.2029
alpha2  2.3450 0.19790  2.343 0.19852   2.3440     0.1977
alpha3 -0.8532 0.35456 -0.850 0.35716  -0.8507     0.3540
sd      1.8586 0.08223  1.858 0.08153   1.8495         NA
```

We see that when we use vague priors in a Bayesian analysis, we will typically obtain parameter estimates with great numerical resemblance to the MLEs. Moreover, the posterior standard deviations will be very similar numerically to the standard errors in the ML analysis. Thus, the fundamental thing is the model, which is the same whether analysed in a non-Bayesian or in a Bayesian way. Whatever you may want to do with this model in a non-Bayesian analysis you can also do in a Bayesian analysis, e.g., residual checks. In a frequentist analysis in R, you can do the following to do residual checks (and you will find out that there are too many large residuals for the assumed normal distribution):

```
plot(lm(Cmean ~ elev*forest))
```

In a Bayesian analysis, every unknown in a model is estimated and therefore has a posterior distribution. Residuals depend on the unknown parameters and hence are unknown quantities themselves and have an entire posterior distribution. We can plot the residuals to check for normality visually, by seeing whether there is any evidence for lack of symmetry. We can also plot the residuals against their order in the data set and versus the predicted values for a visual check of variance homogeneity. These are three frequent residual diagnostic plots (Fig. 5-3). For the predictions, we could simply have saved mu and then used its posterior mean, but since we saved the MCMC draws from its "ingredients" (the regression coefficients), we can compute the posterior mean of mu outside of BUGS in R.

```
mu <- out1B$mean$alpha0 + out1B$mean$alpha1 * elev + out1B$mean$alpha2 * forest
+ out1B$mean$alpha3 * elev * forest       # Compute the posterior mean of mu

par(mfrow = c(2, 2), mar = c(5,4,2,2), cex.main = 1)
plot(1:M, out1B$summary[6:272, 1], xlab = "Order of values", ylab = "Residual",
frame.plot = F, ylim = c(-10, 15))
abline(h = 0, col = "red", lwd = 2)
segments(1:267, out1B$summary[6:272, 3], 1:267, out1B$summary[6:272, 7], col =
"grey")
text(10, 14, "A", cex = 1.5)
hist(out1B$summary[6:272, 1], xlab = "Residual", main = "", breaks = 50, col =
"grey", xlim = c(-10, 15))
abline(v = 0, col = "red", lwd = 2)
text(-9, 48, "B", cex = 1.5)
qq <- qnorm(seq(0,0.9999,,data$M), mean = 0, sd = out1B$summary[5, 1])
plot(sort(qq), sort(out1B$summary[6:272, 1]), xlab = "Theoretical quantile",
ylab = "Residual", frame.plot = F, ylim = c(-10, 15)) # could also use qqnorm()
abline(0, 1, col = "red", lwd = 2)
text(-4.5, 14, "C", cex = 1.5)
plot(mu, out1B$summary[6:272, 1], xlab = "Predicted values", ylab = "Residual",
frame.plot = F, ylim = c(-10, 15))
abline(h = 0, col = "red", lwd = 2)
segments(mu, out1B$summary[6:272, 3], mu, out1B$summary[6:272, 7], col = "grey")
text(-1, 14, "D", cex = 1.5)
```

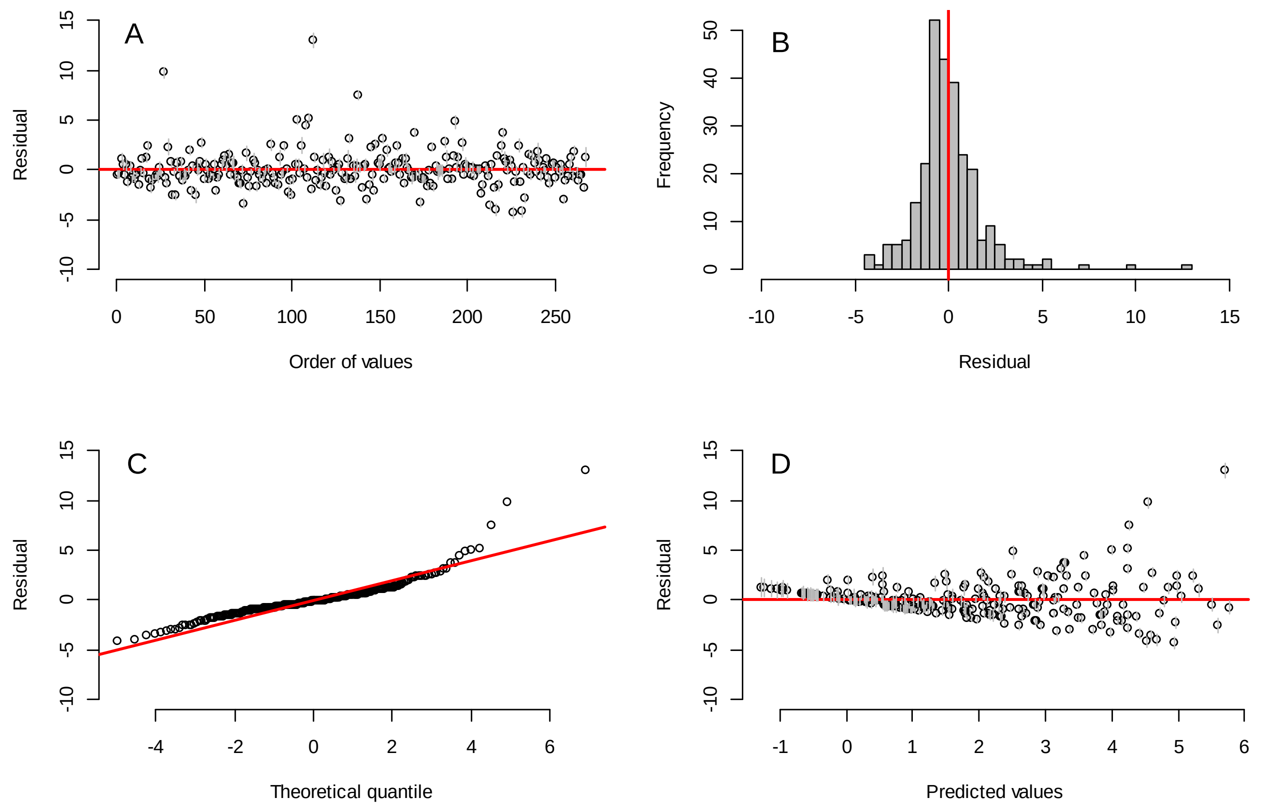


Fig. 5-3: Residual diagnostic plots from the Bayesian analysis of the model: (A) residuals against their order, (B) histogram of residuals, (C) Q-Q plot and (D) residuals against the fitted values. Red line shows zero, except in C, where it is the 1:1 line.

Clearly, the homoscedasticity assumption underlying the normal model is violated somewhat, since the residuals do not form a patternless cloud around a value of 0 and the Q-Q plot shows deviations from a straight line. A more appropriate analysis might use log(counts) or better, a Poisson GLM for the counts directly. However, since we show this normal model for the mean great tit count for illustration only, we are not overly concerned with this potential structural problem. The important message we want to relay is simply that any model diagnostic that you can do for a frequentist analysis, you can also do with a Bayesian analysis of the same model.

Hence, we ignore the lack of model fit and continue our illustration of a Bayesian analysis. Two things that we often want to do is (1) make a statement about how certain we are that a parameter has a particular value and (2) make predictions, i.e., compute (with uncertainty assessment) what response we would expect for one value or for an entire range of values of the covariates. We illustrate this here and again compare with a maximum likelihood (ML) analysis, where we can do a significance test for each of the regression coefficients.

```
fm
[ ... ]
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   1.6603     0.1137  14.607  < 2e-16 ***
elev         -1.5765     0.2029  -7.771 1.76e-13 ***
forest        2.3440     0.1977  11.857  < 2e-16 ***
elev:forest  -0.8507     0.3540  -2.403    0.017 *
```

We see that all three (and the intercept trivially so) are highly significant. We can also compute a confidence interval (CI), which has a somewhat contorted meaning in a non-Bayesian analysis: if we randomly sampled the same population of great tits a great many times, fitted our model and computed a 95% CI each time, then 95% of these intervals would contain the true parameter value. Thus, the frequentist CI is an assessment of the reliability of a method, not a direct statement of uncertainty about an unknown quantity. In frequentist statistics, no probabilistic statements can be made about parameters, because parameters are not random variables.

```
confint(lm(Cmean ~ elev*forest))
                2.5 %    97.5 %
(Intercept)  1.436495  1.884119
elev        -1.975904 -1.177001
forest       1.954756  2.733260
elev:forest -1.547748 -0.153619
```

In a Bayesian analysis we use probability to express our degree of knowledge about uncertain quantities, such as the parameters in our regression model. This is the posterior distribution. Let's plot this and include the central range of the distribution containing 95% of its mass (Fig. 5-4): this is the simplest way of computing a 95% Bayesian confidence interval (often called credible interval, CRI). Unlike a frequentist confidence interval, this is a direct probability statement about an unknown quantity: under the model we can be 95% certain that the true parameter lies within this interval.

```
par(mfrow = c(2, 2), mar = c(5,4,2,2), cex.main = 1)
hist(out1B$sims.list$alpha1, main = "", breaks = 100, col = "grey", freq=F)
abline(v = quantile(out1B$sims.list$alpha1, prob = c(0.025, 0.975)), col =
"red", lwd = 2)
text(-2.4, 1.8, "A", cex = 1.5)
hist(out1B$sims.list$alpha2, main = "", breaks = 100, col = "grey", freq=F)
abline(v = quantile(out1B$sims.list$alpha2, prob = c(0.025, 0.975)), col =
"red", lwd = 2)
text(1.7, 2, "B", cex = 1.5)
hist(out1B$sims.list$alpha3, main = "", breaks = 100, col = "grey", freq=F)
abline(v = quantile(out1B$sims.list$alpha3, prob = c(0.025, 0.975)), col =
"red", lwd = 2)
text(-2.2, 1.2, "C", cex = 1.5)
hist(out1B$sims.list$sd, main = "", breaks = 100, col = "grey", freq=F)
abline(v = quantile(out1B$sims.list$sd, prob = c(0.025, 0.975)), col = "red",
lwd = 2)
text(1.6, 4.9, "D", cex = 1.5)
```

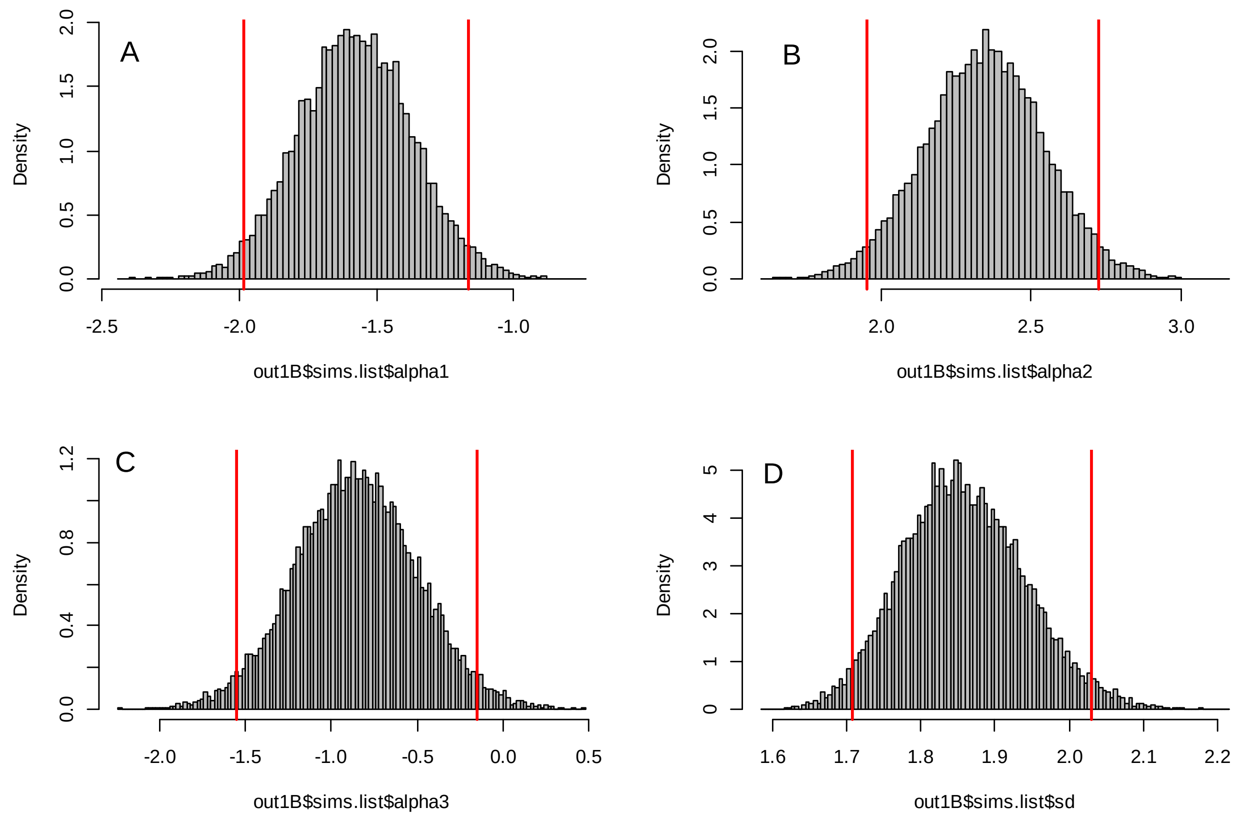


Fig. 5-4: Marginal posterior distributions with percentile-based 95% credible intervals (CRI, red) for four parameters in the normal model for mean tit counts. (A) Effect of elevation, (B) effect of forest cover, (C) interaction effect between elevation and forest cover, (D) residual standard deviation.

Note that there are many different ways of constructing a 95% Bayesian credible interval (Link & Barker, 2010), but the percentile method here is the easiest. One particular CRI is the highest-posterior density interval (HPDI), which is the shortest of such intervals, and can be computed using the `HPDinterval` function in the coda package.

```
HPDinterval(as.mcmc(out1B$sims.list$sd), prob = 0.95)  # HPDI
quantile(out1B$sims.list$sd, prob = c(0.025, 0.975))   # Percentile-based CRI
```

Although the fundamental meaning is quite different, CRIs in a Bayesian analysis can be used to do something analogous to a significance test in frequentist statistics: we see that 0 is a fairly unlikely value for all four parameters shown and that 0 is not included in the 95% CRI of any of them. Hence, we can say that we are quite certain that these parameters are different from zero. We would perhaps not want to say that the parameters are *significant*, because we feel that this term is associated too strongly with the frequentist technique of a significance test. Nevertheless, we often see Bayesians call this a significance test without even using quotes. This information is given in the posterior summary from `jags` in columns 6 and 7.

Although the posterior distribution is a degree-of-belief probability distribution and does not have the same meaning as a probability ("long run frequency") in a frequentist analysis, Bayesian estimates are often very well calibrated in a frequentist sense (Le Cam 1953), i.e., their frequentist characteristics are often quite good. That means, when replicated a great many times, a 95% Bayesian credible interval will often contain the true value 95% of the time. Accordingly, frequentist CI's and Bayesian CRI's are typically very similar numerically.

```
cbind(confint(lm(Cmean ~ elev*forest))[2:4,], out1B$summary[2:4, c(3,7)])
                2.5 %    97.5 %      2.5%     97.5%
elev        -1.975904 -1.177001 -1.981000 -1.167000
forest       1.954756  2.733260  1.954000  2.728025
elev:forest -1.547748 -0.153619 -1.551025 -0.150095
```

A particularly attractive feature when communicating the results of a Bayesian analysis is the ability to make direct probability statements about the magnitude of the parameters. For instance, from the posterior samples of `alpha1` we can easily compute the probability that the slope of `elev` is more extreme than -1.6 or that it lies between -1.8 and -1.6. Since the posterior is a probability distribution function and integrates to one, the geometrical interpretation of these probabilities is simply the area under the posterior defined by these limits.

```
mean(out1B$sims.list$alpha1 < -1.6)
[1] 0.4566
mean(out1B$sims.list$alpha1 < -1.6 & out1B$sims.list$alpha1 > -1.8)
[1] 0.3178667
```

Technically, what we obtain from the MCMC analysis is an estimate of a joint posterior distribution, i.e., the joint distribution of all estimated quantities. For n estimated quantities, you can imagine the joint posterior as a cloud in n-dimensional space. Let's look at this in an example in two dimensions only, for `alpha1` and `alpha2`. We can do "probability games" as before in two dimensions as well, e.g., we could test the hypothesis that the effect of elev lies between -1.9 and -1.6 and that of forest between 2.5 and 2.8. This is the proportion of samples that lie in the inner red square in Fig. 5-5.

```
plot(out1B$sims.list$alpha1, out1B$sims.list$alpha2)
abline(h = c(2.5, 2.8), col = "red", lwd = 2)
abline(v = c(-1.9, -1.6), col = "red", lwd = 2)
```

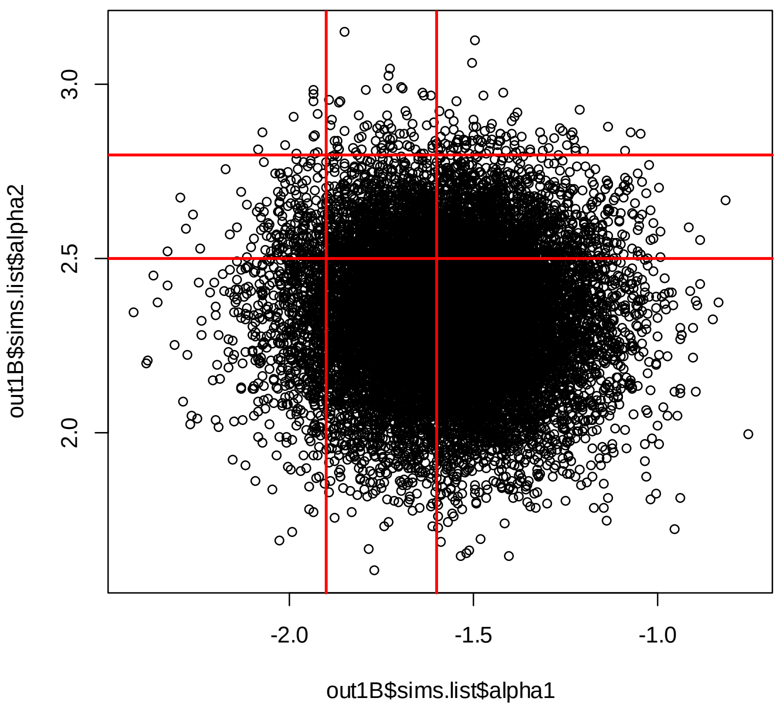


Fig. 5-5: Joint posterior distribution of `alpha1` and `alpha2` with geometrical representation (red square) of the probability that the effect of elev lies between -1.9 and -1.6 and that of forest between 2.5 and 2.8.

```
mean(out1B$sims.list$alpha1 < -1.6 & out1B$sims.list$alpha1 > -1.9 &
out1B$sims.list$alpha2 > 2.5 & out1B$sims.list$alpha2 < 2.8)
0.08286667
```

Hence, that probability is about 8%. Thus, there is a lot of cool stuff that we can do with our posterior samples after running an MCMC algorithm that aren't (easily) possible using a frequentist analysis. At many places in this book will we see examples of where we use posterior samples to compute derived quantities, e.g., functions of one or more parameters, by simply applying the function for each set of MCMC draws of the parameters. This can be done inside of the BUGS program, or outside, in R, if all the "ingredients" of the derived quantities have been sampled and saved.

Indeed, one of the neatest things in a Bayesian MCMC-based analysis is the ease with which such derived quantities can be obtained, along with a full assessment of their uncertainty. In non-Bayesian analyses, we have to use approximations like the delta rule (see 2.3.2) or else use bootstrapping (see 2.3.4), but in an MCMC-based analysis we can simply compute the function of interest for every step of the MCMC algorithm and then base the inference on the resulting posterior. To illustrate further, assume that we had some crazy theory that postulated the main effect of forest was more extreme than the main effect of elevation on the abundance of great tits. From the least-squares fit of the model, we can estimate the ratio between the two at `2.34/1.58 = 1.48`; this is not difficult. But what about the uncertainty in this estimate ? And: is the observed ratio of the absolute effects "significantly" different from 1 ?

Both estimates have an associated estimation uncertainty represented by their standard error (SE). In a simple case, it would not be too hard to work out the SE of the ratio or to bootstrap it, but in a Bayesian analysis the uncertainty of the ratio is readily obtained for models or derived quantities of any complexity. Fig. 5-6 gives a picture of the posterior distribution of the absolute ratio between the effects of forest cover and elevation along with a 95% CRI (red). Since 1 is outside of the 95% CRI, we can thus say that the data are in agreement with the crazy theory.

```
crazy.ratio <- out1B$sims.list$alpha2 / abs(out1B$sims.list$alpha1)
hist(crazy.ratio, main = "", breaks = 100, col = "grey", freq = F)
abline(v = quantile(crazy.ratio, prob = c(0.025, 0.975)), col = "red", lwd = 3)
```

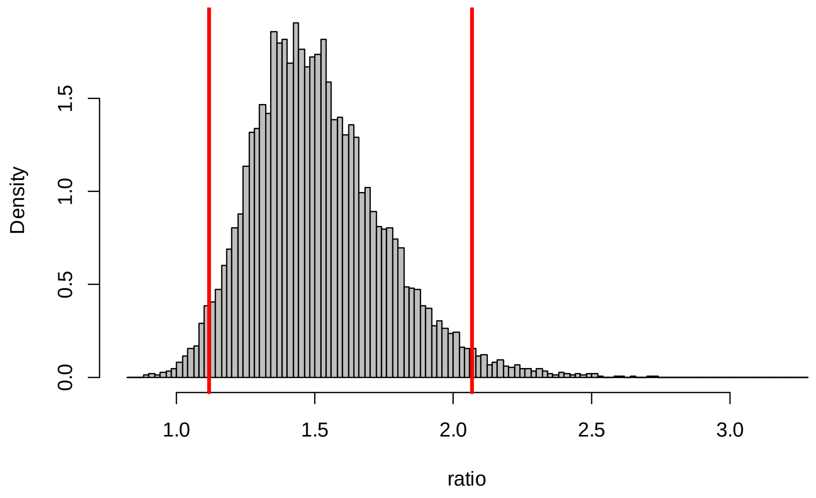


Fig. 5-6: Posterior distribution of a crazy derived parameter, the absolute ratio of the effects of forest cover and elevation on the abundance of great tits, with 95% CRI shown with the red lines.

```
mean(abs(out1B$sims.list$alpha2 / out1B$sims.list$alpha1) > 1)
[1] 0.9964667
```

Thus, we can be nearly 100% certain that the effect of forest cover is more extreme than that of elevation. Hence, very rich inferences are possible and indeed readily obtainable based on the MCMC sample of posterior distributions.

The final thing we illustrate here is the forming of predictions. Let's use our Bayesian parameter estimates to predict a response surface of the mean observed tit abundance as a function of elevation and forest cover and as a series of regression lines on elevation for selected values of forest cover (Fig. 5-7 A and B):

```
# Compute expected abundance for a grid of elevation and forest cover
elev.pred <- seq(-1, 1,,100)                       # Values of elevation
forest.pred <- seq(-1,1,,100)                      # Values of forest cover
pred.matrix <- array(NA, dim = c(100, 100)) # Prediction matrix
for(i in 1:100){
   for(j in 1:100){
      pred.matrix[i, j] <- out1J$mean$alpha0 + out1J$mean$alpha1 * elev.pred[i]
+ out1J$mean$alpha2 * forest.pred[j] + out1J$mean$alpha3 * elev.pred[i] *
forest.pred[j]
   }
}

par(mfrow = c(1, 3), mar = c(5,5,3,2), cex.main = 1.6, cex.axis = 1.5, cex.lab =
1.5)
mapPalette <- colorRampPalette(c("grey", "yellow", "orange", "red"))
image(x=elev.pred, y= forest.pred, z=pred.matrix, col = mapPalette(100), xlab =
"Elevation", ylab = "Forest cover")
contour(x=elev.pred, y=forest.pred, z=pred.matrix, add = TRUE, lwd = 1, cex =
1.5)
title(main = "A")
matpoints(elev, forest, pch="+", cex=1.5)
abline(h = c(-1, -0.5, 0, 0.5, 1))

# Predictions for elev. at specific values of forest cover (-1,-0.5,0,0.5,1)
pred1 <- out1J$mean$alpha0 + out1J$mean$alpha1 * elev.pred + out1J$mean$alpha2 *
(-1) + out1J$mean$alpha3 * elev.pred * (-1)
pred2 <- out1J$mean$alpha0 + out1J$mean$alpha1 * elev.pred + out1J$mean$alpha2 *
(-0.5) + out1J$mean$alpha3 * elev.pred * (-0.5)
pred3 <- out1J$mean$alpha0 + out1J$mean$alpha1 * elev.pred + out1J$mean$alpha2 *
0 + out1J$mean$alpha3 * elev.pred * 0
# pred3b <- out1J$mean$alpha0 + out1J$mean$alpha1 * elev.pred   # same
pred4 <- out1J$mean$alpha0 + out1J$mean$alpha1 * elev.pred + out1J$mean$alpha2 *
0.5 + out1J$mean$alpha3 * elev.pred * 0.5
pred5 <- out1J$mean$alpha0 + out1J$mean$alpha1 * elev.pred + out1J$mean$alpha2 *
1 + out1J$mean$alpha3 * elev.pred * 1
matplot(seq(-1, 1,,100), cbind(pred1, pred2, pred3, pred4, pred5), type = "l",
lty= 1, col = "blue", ylab = "Prediction of mean count", xlab = "Elevation",
ylim = c(-1.5, 7), lwd = 2)
title(main = "B")
```

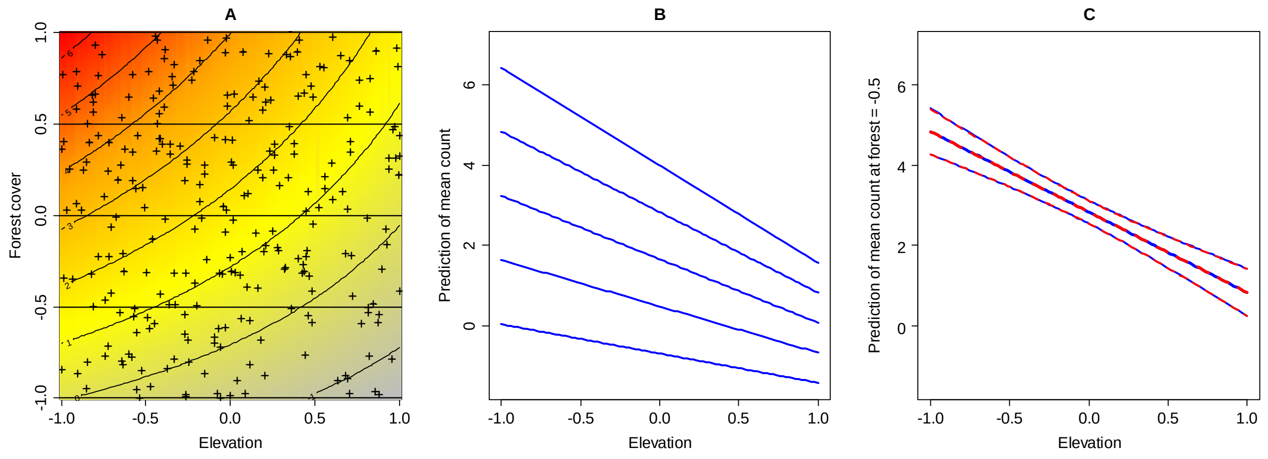


Fig. 5-7: (A and B) Two ways of plotting predictions of the expected tit counts as a function of two continuous covariates: as a continuous response surface (A) and as a bundle of regression lines for a series of discrete values (here, for -1, -0.5, 0, 0.5 and 1) for the forest covariate (B). (C): prediction with uncertainty (blue: Bayesian analysis, red: maximum likelihood, with 95% credible or confidence intervals, respectively). The curvature in the left and the non-parallelism in the right indicates the presence of an interaction between elevation and forest cover. Note that the normal model for counts may predict impossible values, such as negative tit counts.

These plots do not show the estimation uncertainty. This would be unwieldy in this type of plot, but it can easily be shown for simpler examples. We do this next, both for the Bayesian and for the frequentist analysis, and plot the estimated relationship between the expected mean count of great tits and elevation at a forest cover of 0.5 (Fig. 5–7C). For this, we take all or a subset of the MCMC samples of the constituents of the predictions, i.e., the parameters alpha0 through alpha3, and compute predictions using the values of the prediction covariate. We do this in a loop, but first prepare a matrix to contain the posterior samples for the predictions.

```
pred.mat <- array(dim = c(length(elev.pred), length(out1J$sims.list$alpha0)))
for(j in 1:length(out1J$sims.list$alpha0)){
   pred.mat[,j] <- out1J$sims.list$alpha0[j] + out1J$sims.list$alpha1[j] *
elev.pred + out1J$sims.list$alpha2[j] * 0.5 + out1J$sims.list$alpha3[j] *
elev.pred * 0.5
}
```

To plot the 95% CRI of the Bayesian prediction we can use the pointwise 95% CRI:

```
CL <- apply(pred.mat, 1, function(x){quantile(x, prob = c(0.025, 0.975))})
plot(seq(-1, 1,,100), pred4, type = "l", lty= 1, col = "blue", ylab =
"Prediction of mean count at forest = -0.5", xlab = "Elevation", las =1, ylim =
c(-1.5, 7), lwd = 3)
matlines(seq(-1, 1,,100), t(CL), lty = 1, col = "blue", lwd = 2)
title(main = "C")
```

We add the frequentist prediction along with the 95% confidence limits.

```
pred <- predict(lm(Cmean ~ elev*forest), newdata = data.frame(elev = seq(-1,
1,,100), forest = 0.5), se.fit = TRUE, interval = "confidence")
lines(seq(-1, 1,,100), pred$fit[,1], lty= 2, col = "red", lwd = 3)
matlines(seq(-1, 1,,100), pred$fit[,2:3], lty = 2, col = "red", lwd = 2)
```

In real-life data analysis, what many find confusing is how predictions are presented for a covariate that has been transformed (e.g., scaled) before analysis. Then, we first have to form the predictions for the transformed covariates and then backtransform again for graphing. We show this in later parts of the book, e.g., in section 5.9.

5.6 Linear model with normal response (normal GLM): analysis of covariance (ANCOVA)

Returning to the illustration of common linear models, we next use BUGS to fit a linear model that underlies a technique called analysis of covariance (ANCOVA). Specifically, within a generalised linear model with normal response we fit to the mean tit counts the linear model underlying a fixed-effects ANCOVA with interaction effects. For this, we somewhat artificially first construct a factor that classifies the continuous covariate `forest cover` into four levels or groups, with level 1 for values between -1 and -0.5, level 2 corresponding to -0.49 and 0 etc.; see Fig. 5–10 (A) for the raw relationship between mean tit count and levels of the forest factor. Factors in BUGS must be labeled with integer numbers and not, for instance, with letters or words, and the numbering must start at 1 and end at the number of levels, i.e., have no jumps (e.g., 1, 2, 4, 5 would cause a crash). We fit the following model in the effects and in the means parameterisation (see chapter 3), where *j* indexes the four levels of the forest factor:

$$Cmean_i \sim Normal(\mu_i, \sigma^2)$$
$$\mu_i = \alpha_{0,j} + \alpha_{1,j} * elev_i$$

```
# Generate factor and plot raw data in boxplot as function of factor A
facFor <- as.numeric(forest < -0.5)             # Factor level 1
facFor[forest < 0 & forest > -0.5] <- 2         # Factor level 2
facFor[forest < 0.5 & forest > 0] <- 3          # Factor level 3
facFor[forest > 0.5] <- 4                       # Factor level 4
table(facFor)                                   # every site assigned a level OK

par(mfrow = c(1, 2), mar = c(5,5,3,2), cex.lab = 1.5, cex.axis = 1.5)
plot(Cmean ~ factor(facFor), col = c("red", "blue", "green", "grey"), xlab =
"Forest cover class", ylab = "Mean count of great tits", frame.plot = F, ylim =
c(0,20))
text(0.8, 20, "A", cex=1.6)
```

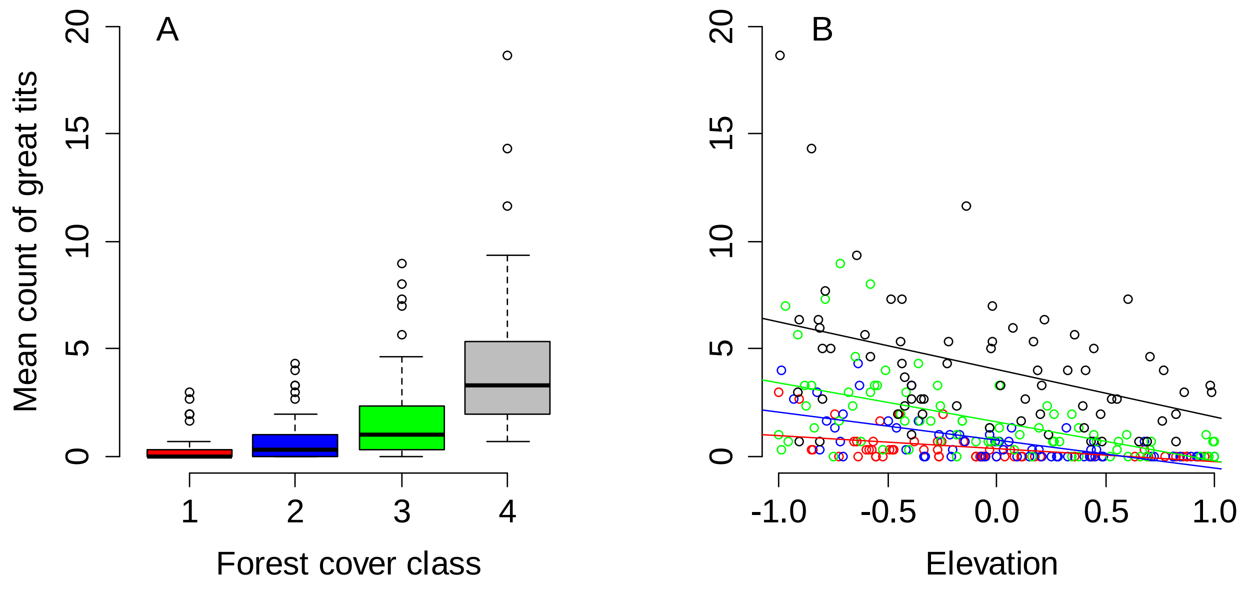


Fig. 5–10: (A) Relationship between the mean count of great tits and the levels of the forest factor (`facFor`). Raw data shown for each level of the forest factor. (B) Raw data and predicted

relationship with elevation under the ANCOVA model with a least-square fit. Colors denote the four levels of the forest factor.

```
# Bundle data
win.data <- list(Cmean = Cmean, M = length(Cmean), elev = elev, facFor = facFor)
```

We can define the model in the effects or in the means parameterisation and we show both. In either case, we define vector-valued parameters using the handy nested indexing in the BUGS language. We will fit the model in WinBUGS, JAGS and, using function `lm` in R, using maximum likelihood.

```
# Specify model in BUGS language in effects parameterisation
cat(file = "ANCOVA1.txt","
model {

# Priors
alpha ~ dnorm(0, 1.0E-06)            # Prior for intercept = effect of level 1
of forest factor
beta2 ~ dnorm(0, 1.0E-06)            # Prior for slope = effect of elevation for
level 1 of forest factor
beta1[1] <- 0                        # Set to zero effect of first level of
facFor
beta3[1] <- 0                        # Set to zero effect of first level of
facFor of elevation
for(k in 2:4){
   beta1[k] ~ dnorm(0, 1.0E-06)       # Prior for effects of factor facFor
   beta3[k] ~ dnorm(0, 1.0E-06)       # Prior for effects of factor facFor
}
tau <- pow(sd, -2)
sd ~ dunif(0, 1000)                  # Prior for dispersion on sd scale

# Likelihood
for (i in 1:M){
   Cmean[i] ~ dnorm(mu[i], tau)          # precision tau = 1 / variance
   mu[i] <- alpha + beta1[facFor[i]] + beta2 * elev[i] + beta3[facFor[i]] *
elev[i]
}
}
")
```

We must not give any initial values for fixed quantities (here, beta1[1] and beta3[1]); note that in place of the initial for the first element of the parameter vectors beta1 and beta3, we have an 'NA'.

```
# Initial values
inits <- function() list(alpha = rnorm(1,,10), beta1 = c(NA, rnorm(3,,10)),
beta2 = rnorm(1,,10), beta3 = c(NA, rnorm(3,,10)))

# Parameters monitored
params <- c("alpha", "beta1", "beta2", "beta3", "sd")

# MCMC settings
ni <- 6000   ;   nt <- 1   ;   nb <- 1000   ;  nc <- 3

# Call WinBUGS or JAGS from R (ART <1 min)
out3 <- bugs(win.data, inits, params, "ANCOVA1.txt", n.chains = nc, n.thin = nt,
n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory = bugs.dir,
working.directory = getwd())
```

```
out3J <- jags(win.data, inits, params, "ANCOVA1.txt", n.chains = nc, n.thin =
nt, n.iter = ni, n.burnin = nb)
traceplot(out3J)

# Fit model using least-squares (produces MLEs)
(fm <- summary(lm(Cmean ~ as.factor(facFor)*elev)))

Coefficients:
                        Estimate Std. Error t value Pr(>|t|)
(Intercept)               0.3353     0.2301   1.457  0.14633
as.factor(facFor)2        0.4244     0.3231   1.313  0.19028
as.factor(facFor)3        1.2690     0.3083   4.115  5.2e-05 ***
as.factor(facFor)4        3.7205     0.3162  11.766  < 2e-16 ***
elev                     -0.6013     0.4203  -1.431  0.15377
as.factor(facFor)2:elev  -0.6866     0.5999  -1.145  0.25345
as.factor(facFor)3:elev  -1.2116     0.5427  -2.232  0.02644 *
as.factor(facFor)4:elev  -1.6164     0.5708  -2.832  0.00499 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.783 on 259 degrees of freedom
Multiple R-squared:  0.4941,    Adjusted R-squared:  0.4804
F-statistic: 36.13 on 7 and 259 DF,  p-value: < 2.2e-16

# Summarize posteriors
print(out3, 3)
            mean    sd     2.5%      25%      50%      75%    97.5%  Rhat n.eff
alpha      0.337 0.231   -0.114    0.183    0.337    0.491    0.799 1.001 15000
beta1[2]   0.422 0.324   -0.211    0.205    0.422    0.641    1.051 1.001 15000
beta1[3]   1.268 0.310    0.664    1.062    1.267    1.473    1.887 1.001 15000
beta1[4]   3.721 0.318    3.093    3.509    3.721    3.931    4.350 1.001 15000
beta2     -0.602 0.421   -1.442   -0.885   -0.600   -0.319    0.222 1.001 15000
beta3[2]  -0.687 0.605   -1.859   -1.101   -0.692   -0.277    0.503 1.001 15000
beta3[3]  -1.215 0.544   -2.290   -1.581   -1.218   -0.847   -0.158 1.001 15000
beta3[4]  -1.611 0.578   -2.744   -1.999   -1.610   -1.223   -0.456 1.001  6100
sd         1.791 0.078    1.648    1.737    1.788    1.842    1.953 1.001 15000
deviance 1067.483 4.339 1061.000 1064.000 1067.000 1070.000 1078.000 1.001 15000

DIC info (using the rule, pD = Dbar-Dhat)
pD = 9.0 and DIC = 1076.5
DIC is an estimate of expected predictive error (lower deviance is better).
```

We see the usual close numerical agreement between the Bayesian estimates and those using maximum likelihood with function `lm` in R. Next, we fit the model using the means parameterisation, where we fit directly the effect of each level of factor `facFor` (note the changed parameter naming). We don't need any change in the data bundle. In addition, we also illustrate how we can estimate custom contrasts as derived quantities, i.e., differences or other functions of parameters. We estimate all pairwise differences between the group means beta[1:4]. Of course, we could also easily compute these derived quanties in R using posterior samples of the vector beta produced by BUGS.

```
# Specify model in BUGS language
cat(file = "ANCOVA2.txt","
model {

# Priors
for(k in 1:4){
   alpha[k] ~ dnorm(0, 1.0E-06)       # Priors for intercepts
   beta[k] ~ dnorm(0, 1.0E-06)        # Priors for slopes
}
tau <- pow(sd, -2)
sd ~ dunif(0, 1000)                  # Prior for dispersion on sd scale
# Likelihood
for (i in 1:M){
   Cmean[i] ~ dnorm(mu[i], tau)          # precision tau = 1 / variance
   mu[i] <- alpha[facFor[i]] + beta[facFor[i]] * elev[i]
}

# Derived quantities: comparison of slopes (now you can forget the delta rule !)
for(k in 1:4){
   diff.vs1[k] <- beta[k] - beta[1]    # Differences relative to beta[1]
   diff.vs2[k] <- beta[k] - beta[2]    # ... relative to beta[2]
   diff.vs3[k] <- beta[k] - beta[3]    # ... relative to beta[3]
   diff.vs4[k] <- beta[k] - beta[4]    # ... relative to beta[4]
}
}
")

# Initial values
inits <- function() list(alpha = rnorm(4,,10), beta = rnorm(4,,10))

# Parameters monitored
params <- c("alpha", "beta", "sd", "diff.vs1", "diff.vs2", "diff.vs3",
"diff.vs4")

# MCMC settings
ni <- 6000   ;   nt <- 1   ;   nb <- 1000   ;  nc <- 3

# Call WinBUGS or JAGS from R (ART <1 min) and summarize posteriors
out4 <- bugs(win.data, inits, params, "ANCOVA2.txt", n.chains = nc, n.thin = nt,
n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory = bugs.dir,
working.directory = getwd())

system.time(out4J <- jags(win.data, inits, params, "ANCOVA2.txt", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb))
traceplot(out4J)

print(out4, 2)
Inference for Bugs model at "ANCOVA2.txt", fit using WinBUGS,
Current: 3 chains, each with 6000 iterations (first 1000 discarded)
Cumulative: n.sims = 15000 iterations saved
               mean   sd    2.5%     25%     50%     75%   97.5% Rhat n.eff
alpha[1]       0.33 0.23   -0.13    0.18    0.33    0.49    0.79    1 15000
alpha[2]       0.76 0.23    0.31    0.61    0.76    0.91    1.21    1 15000
alpha[3]       1.60 0.21    1.19    1.46    1.60    1.74    2.00    1 15000
alpha[4]       4.06 0.22    3.64    3.91    4.05    4.20    4.49    1 15000
beta[1]       -0.60 0.42   -1.41   -0.88   -0.60   -0.31    0.23    1 15000
beta[2]       -1.29 0.43   -2.14   -1.58   -1.29   -0.99   -0.43    1 15000
beta[3]       -1.82 0.34   -2.49   -2.05   -1.82   -1.59   -1.13    1  6700
beta[4]       -2.22 0.39   -2.97   -2.48   -2.22   -1.96   -1.45    1 15000
sd             1.79 0.08    1.65    1.74    1.79    1.84    1.96    1  9900
diff.vs1[1]    0.00 0.00    0.00    0.00    0.00    0.00    0.00    1     1
diff.vs1[2]   -0.69 0.60   -1.88   -1.10   -0.69   -0.28    0.48    1 15000
diff.vs1[3]   -1.22 0.55   -2.30   -1.58   -1.22   -0.86   -0.15    1 14000
diff.vs1[4]   -1.62 0.57   -2.73   -2.01   -1.63   -1.23   -0.49    1 15000
diff.vs2[1]    0.69 0.60   -0.48    0.28    0.69    1.10    1.88    1 15000
diff.vs2[2]    0.00 0.00    0.00    0.00    0.00    0.00    0.00    1     1
diff.vs2[3]   -0.53 0.56   -1.63   -0.90   -0.53   -0.16    0.57    1 15000
diff.vs2[4]   -0.93 0.58   -2.07   -1.32   -0.93   -0.54    0.21    1 15000
diff.vs3[1]    1.22 0.55    0.15    0.86    1.22    1.58    2.30    1 14000
diff.vs3[2]    0.53 0.56   -0.57    0.16    0.53    0.90    1.63    1 15000
diff.vs3[3]    0.00 0.00    0.00    0.00    0.00    0.00    0.00    1     1
diff.vs3[4]   -0.40 0.52   -1.42   -0.75   -0.40   -0.05    0.62    1  9700
diff.vs4[1]    1.62 0.57    0.49    1.23    1.63    2.01    2.73    1 15000
diff.vs4[2]    0.93 0.58   -0.21    0.54    0.93    1.32    2.07    1 15000
diff.vs4[3]    0.40 0.52   -0.62    0.05    0.40    0.75    1.42    1  9700
diff.vs4[4]    0.00 0.00    0.00    0.00    0.00    0.00    0.00    1     1
```

```
# Fit model using least-squares (produces MLEs)
(fm <- summary(lm(Cmean ~ as.factor(facFor)*elev-1-elev)))

Coefficients:
                        Estimate Std. Error t value Pr(>|t|)
as.factor(facFor)1        0.3353     0.2301   1.457 0.146328
as.factor(facFor)2        0.7596     0.2269   3.348 0.000935 ***
as.factor(facFor)3        1.6042     0.2052   7.816 1.37e-13 ***
as.factor(facFor)4        4.0558     0.2169  18.700  < 2e-16 ***
as.factor(facFor)1:elev  -0.6013     0.4203  -1.431 0.153772
as.factor(facFor)2:elev  -1.2880     0.4280  -3.009 0.002880 **
as.factor(facFor)3:elev  -1.8129     0.3433  -5.281 2.73e-07 ***
as.factor(facFor)4:elev  -2.2177     0.3862  -5.743 2.60e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.783 on 259 degrees of freedom
Multiple R-squared:  0.6689,    Adjusted R-squared:  0.6587
F-statistic: 65.42 on 8 and 259 DF,  p-value: < 2.2e-16
```

We will often see the linear model figure of an ANOVA (analysis of variance) or an ANCOVA using nested indexing in the BUGS language. Let's plot the predicted response as a function of the explanatory variables facFor and elevation (Fig. 5-10B). We use the parameter estimates from the least-squares fit, but clearly could also use the Bayesian posterior means.

```
plot(elev[facFor==1], Cmean[facFor==1], col = "red", ylim = c(0, 20), xlab =
"Elevation", ylab = "", frame.plot = F)
points(elev[facFor==2], Cmean[facFor==2], col = "blue")
points(elev[facFor==3], Cmean[facFor==3], col = "green")
points(elev[facFor==4], Cmean[facFor==4], col = "black")
abline(fm$coef[1,1], fm$coef[5,1], col = "red")
abline(fm$coef[2,1], fm$coef[6,1], col = "blue")
abline(fm$coef[3,1], fm$coef[7,1], col = "green")
abline(fm$coef[4,1], fm$coef[8,1], col = "black")
text(-0.8, 20, "B", cex=1.6)
```

To further illustrate how simple it is to test custom hypotheses in an MCMC-based analysis, let's compute the probability that the difference in the slopes between forest factor level 3 and the others is greater than 1. We plot the histograms of these contrasts (Fig. 5–11) and then compute the proportion of area under the curve that lies to the right of 1.

```
attach.bugs(out4)     # Allows to directly address the sims.list
str(diff.vs3)
par(mfrow = c(1, 3), mar = c(5,5,3,2), cex.lab = 1.5, cex.axis = 1.5)
hist(diff.vs3[,1], col = "grey", breaks = 100, main = "", freq=F, ylim = c(0,
0.8))
abline(v = 1, lwd = 3, col = "red")
text(-1.2, 0.8, "A", cex = 2)
hist(diff.vs3[,2], col = "grey", breaks = 100, main = "", freq=F, ylim = c(0,
0.8))
abline(v = 1, lwd = 3, col = "red")
text(-1.4, 0.8, "B", cex = 2)
hist(diff.vs3[,4], col = "grey", breaks = 100, main = "", freq=F, ylim = c(0,
0.8))
abline(v = 1, lwd = 3, col = "red")
text(-2.2, 0.8, "C", cex = 2)
```

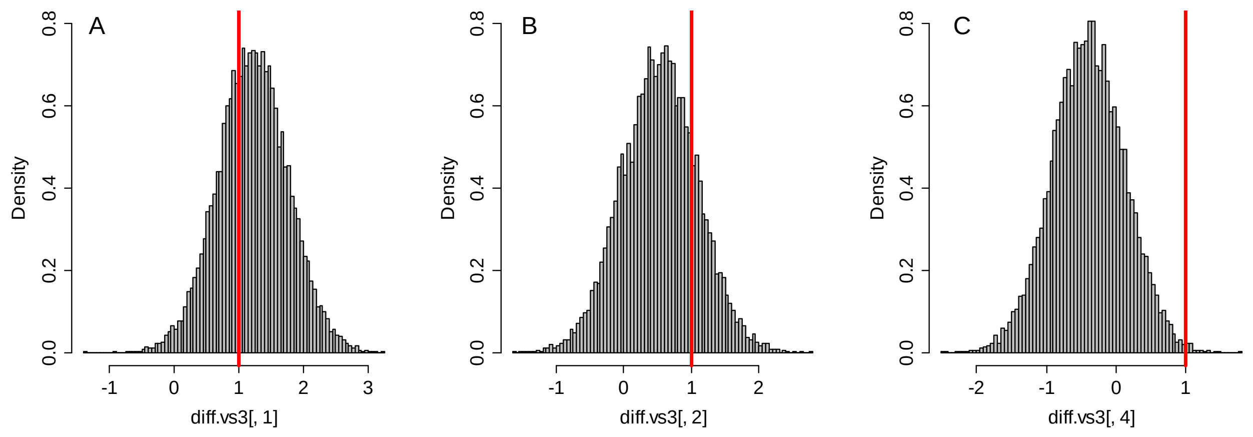


Fig. 5–11: Posterior distributions of the difference between the slope of mean tit counts on elevation in forest factor level 3 versus level 1 (A), 2 (B) and 4 (C). The probability that this difference is greater than 1 is represented by the area under the curve to the right of the red lines.

```
# Prob. difference greater than 1
mean(diff.vs3[,1] > 1)
mean(diff.vs3[,2] > 1)
mean(diff.vs3[,4] > 1)
 [1] 0.6554667
 [1] 0.1981333
 [1] 0.003733333
```

Hence, there is a 66% probability that the difference between the slopes in group 1 and 3 of the forest factor is greater than 1, and corresponding probabilities of 20% and of essentially 0% for the analogous slope differences between group 3 and groups 2 and 4, respectively.

5.8 Fitting a model with non-standard likelihood using the zeros or the ones tricks

Using the standard distribution functions in BUGS, you can fit a vast number of models. Moreover, by combining two or more of them, e.g., in a hierarchical model, you can extend the range of models considerably still. However, sometimes you may encounter a distribution that you cannot specify, or you may want to fit the integrated likelihood (see chapter 2) of an HM directly. When you know how to write the likelihood of your model, you can fit it in BUGS using what is known as the "zeros trick" or the "ones trick" (Lunn *et al.* 2013, p. 204–206).

For the zeros trick, imagine that you want to fit a model to a data set where observation *i* contributes a likelihood term L[i]. If we invent a dummy data set of all-zeros and assume a Poisson(ϕ) distribution for it, then every observation has a contribution to the likelihood equal to exp(-ϕ). If we then specify ϕ to be equal to the negative log-likelihood of observation *i* under our original model, we obtain the correct likelihood. In practice we will have to add an arbitrary constant *C* to ensure non-negativity of the Poisson mean. For the ones trick, we start with dummy data consisting of solely ones and specify a Bernoulli distribution with success parameter p_i, which is defined to be proportional to the desired likelihood term L[i] under the original model. Again, an arbitrary scaling constant *C* is usually required to ensure that $p_i \leq 1$. Here we illustrate both approaches for the trivial example of the normal response multiple linear regression from 5.3. We compare both solutions with the solution obtained using the standard distribution function for the normal.

Remember the likelihood of an individual normal observation (see eq. 2.2),

$$L(\mu, \sigma^2 \mid \mathbf{y}) = \sqrt{\frac{1}{2\pi\sigma^2}} \exp(-\frac{(y_i - \mu)^2}{2\sigma^2}),$$

and, hence, the negative log-likelihood is

$$NLL(\mu, \sigma^2 \mid \mathbf{y}) = -\log(\sqrt{\frac{1}{2\pi\sigma^2}}) + \frac{(y_i - \mu)^2}{2\sigma^2}.$$

Below we will use both equations in the likelihood specification, the former in the ones trick and the latter in the zeros trick.

```
# Package the data needed in a bundle
win.data <- list(Cmean1 = Cmean, Cmean2 = Cmean, zeros = rep(0, M), ones =
rep(1, M), M = length(Cmean), elev = elev, forest = forest) # note 2 copies of
response

# Write text file with model description in BUGS language
cat(file = "multiple_linear_regression_model.txt",
"model {

# Priors
for(k in 1:3){ # Loop over three ways to specify likelihood
   alpha0[k] ~ dnorm(0, 1.0E-06)           # Prior for intercept
   alpha1[k] ~ dnorm(0, 1.0E-06)           # Prior for slope of elev
   alpha2[k] ~ dnorm(0, 1.0E-06)           # Prior for slope of forest
   alpha3[k] ~ dnorm(0, 1.0E-06)           # Prior for slope of interaction
   sd[k] ~ dunif(0, 1000)                  # Prior for dispersion on sd scale
}
var1 <- pow(sd[1], 2)                      # Variance in zeros trick
var2 <- pow(sd[2], 2)                      # Variance in ones trick
tau <- pow(sd[3], -2)                      # Precision tau = 1/(sd^2)

C1 <- 10000 # zeros trick: make large enough to ensure lam >= 0
C2 <- 10000 # ones trick: make large enough to ensure p <= 1
pi <- 3.1415926

# Three variants of specification of the likelihood
for (i in 1:M){
# 'Zeros trick' for normal likelihood
   zeros[i] ~ dpois(phi[i])  # likelihood contribution is exp(-phi)
#   negLL[i] <- log(sd[1]) + 0.5 * pow((Cmean1[i] - mu1[i]) / sd[1],2 )
   negLL[i] <- -log(sqrt(1/(2*pi*var1))) + pow(Cmean1[i]-mu1[i],2)/(2*var1)
   phi[i] <- negLL[i] + C1
   mu1[i] <- alpha0[1] + alpha1[1]*elev[i] + alpha2[1]*forest[i] +
alpha3[1]*elev[i]*forest[i]

# 'Ones trick' for normal likelihood
   ones[i] ~ dbern(p[i])  # likelihood contribution is p directly
   L[i] <- sqrt(1/(2*pi*var2)) * exp(-pow(Cmean1[i]-mu2[i],2)/(2*var2))
   p[i] <- L[i] / C2
   mu2[i] <- alpha0[2] + alpha1[2]*elev[i] + alpha2[2]*forest[i] +
alpha3[2]*elev[i]*forest[i]

# Standard distribution function for the normal
   Cmean2[i] ~ dnorm(mu3[i], tau)
   mu3[i] <- alpha0[3] + alpha1[3]*elev[i] + alpha2[3]*forest[i] +
alpha3[3]*elev[i]*forest[i]
}
}"
)

# Initial values
inits <- function() list(alpha0 = rnorm(3, 0, 10), alpha1 = rnorm(3,0,10),
alpha2 = rnorm(3,0,10), alpha3 = rnorm(3,0,10))

# Parameters monitored (i.e., for which estimates are saved)
params <- c("alpha0", "alpha1", "alpha2", "alpha3", "sd")

# MCMC settings
ni <- 1200   ;   nt <- 1   ;   nb <- 200   ;  nc <- 3    # For JAGS
```

Much longer chains are required for BUGS to converge, so we fit the model in JAGS.

```
# Call JAGS
library(jagsUI)
outX <- jags(win.data, inits, params, "multiple_linear_regression_model.txt",
n.chains = nc, n.thin = nt, n.iter = ni, n.burnin = nb)
print(outX)

                 mean    sd        2.5%         50%       97.5% overlap0     f  Rhat n.eff
alpha0[1]       1.662 0.115       1.441       1.663       1.884    FALSE 1.000 1.002  1417
alpha0[2]       1.660 0.114       1.433       1.660       1.874    FALSE 1.000 1.000  3000
alpha0[3]       1.657 0.115       1.436       1.660       1.878    FALSE 1.000 1.000  3000
alpha1[1]      -1.567 0.202      -1.957      -1.570      -1.176    FALSE 1.000 1.001  3000
alpha1[2]      -1.577 0.206      -1.980      -1.570      -1.187    FALSE 1.000 1.001  2180
alpha1[3]      -1.585 0.209      -1.989      -1.587      -1.170    FALSE 1.000 1.000  3000
alpha2[1]       2.346 0.191       1.966       2.347       2.719    FALSE 1.000 1.000  3000
alpha2[2]       2.349 0.202       1.963       2.345       2.753    FALSE 1.000 1.000  3000
alpha2[3]       2.346 0.199       1.957       2.342       2.720    FALSE 1.000 1.001  1435
alpha3[1]      -0.848 0.362      -1.574      -0.849      -0.153    FALSE 0.990 1.001  1226
alpha3[2]      -0.842 0.355      -1.524      -0.846      -0.128    FALSE 0.991 1.000  3000
alpha3[3]      -0.837 0.359      -1.548      -0.845      -0.119    FALSE 0.991 1.000  3000
sd[1]           1.861 0.083       1.706       1.858       2.033    FALSE 1.000 1.000  3000
sd[2]           1.861 0.083       1.702       1.858       2.032    FALSE 1.000 1.002  1333
sd[3]           1.860 0.084       1.704       1.856       2.030    FALSE 1.000 1.000  3000
deviance  5348179.889 5.641 5348171.008 5348179.076 5348192.794    FALSE 1.000 1.000  3000
```

Up to MC error, we get identical answers for all three specifications of the normal likelihood. Finally, let's amuse ourselves by doing the analogous thing with ML, i.e., maximise the likelihood for the explicit description of the normal negative log-likelihood. We again find estimates that are numerically virtually identical to the posterior means.

```
# Define negative log-likelihood function
neglogLike <- function(param) {
   alpha0 = param[1]
   alpha1 = param[2]
   alpha2 = param[3]
   alpha3 = param[4]
   sigma = exp(param[5])    # Estimate sigma on log-scale
   mu = alpha0 + alpha1*elev + alpha2*forest + alpha3*elev*forest
#   -sum(dnorm(Cmean, mean=mu, sd=sigma, log=TRUE))  # cheap quick way
sum(-log(sqrt(1/(2*3.1415926*sigma^2))) + (Cmean-mu)^2/(2*sigma^2))
}

# Find parameter values that minimize function value
(fit <- optim(par = rep(0, 5), fn = neglogLike, method = "BFGS"))
 [1]  1.6603052 -1.5764599  2.3440036 -0.8506793  0.6073662

exp(fit$par[5])            # Backtransform to get sigma
[1] 1.83559
```

The zeros and ones tricks allow you to fit very general models, and you will sometimes see people fit hierarchical models using these methods (e.g., Garrard *et al.* 2008; 2013; Chelgren et al. 2011b).

5.9 Poisson generalized linear model (Poisson GLM)

We continue with a non-normal GLM and adopt a Poisson distribution with ANCOVA linear model for the counts of great tits. Since we have not one count per site, but three, we will follow a common approach and simply analyse the maximum count at each site, knowing that this must be the best non-model-based approximation, in the sense of being closest, to the true abundance of great tits at a site. We do *not* encourage this practice in general, but show this analysis here mainly to illustrate a Poisson GLM. Let *j* index the four levels of the forest factor:

$$Cmax_i \sim Poisson(\lambda_i)$$
$$\log(\lambda_i) = \alpha_{0,j} + \alpha_{1,j} * elev_i$$

where $Cmax_i$ is the maximum count for unit *i* and λ_i is the expected maximum count. We also compute Pearson residuals, $(Cmax_i - \lambda_i)/\sqrt{\lambda_i}$, which have the form of a raw residual divided by the standard deviation of unit *i*. To emphasize the relatedness among different GLMs, we fit the same ANCOVA linear model as in the previous section, namely main and interaction effects of the forest factor and elevation. To avoid numerical problems when the expected value becomes equal to zero, resulting in a division by zero in the Pearson residual, we add a small number e to the denominator.

```
# Summarize data by taking max at each site
Cmax <- apply(C, 1, max)
table(Cmax)
Cmax
0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 22 30
77 55 30 12 17 23 11 10  6  6  2  2  2  3  4  1  2  1  2  1

# Bundle data
win.data <- list(Cmax = Cmax, M = length(Cmax), elev = elev, facFor = facFor, e
= 0.0001)

# Specify model in BUGS language
cat(file = "Poisson_GLM.txt","
model {

# Priors
for(k in 1:4){
   alpha[k] ~ dnorm(0, 1.0E-06)       # Prior for intercepts
   beta[k] ~ dnorm(0, 1.0E-06)        # Prior for slopes
}

# Likelihood
for (i in 1:M){
   Cmax[i] ~ dpois(lambda[i])         # note no variance parameter
   log(lambda[i]) <- alpha[facFor[i]] + beta[facFor[i]] * elev[i]
   resi[i] <- (Cmax[i]-lambda[i]) / (sqrt(lambda[i])+e)   # Pearson resi
}
}
")
```

This non-normal GLM is specified in the first two code lines inside the loop under the heading of the likelihood. The first defines the data distribution and the second line specifies a log link and the linear predictor, with the latter corresponding simply to an ANCOVA linear model as before.

```
# Initial values
inits <- function() list(alpha = rnorm(4,,3), beta = rnorm(4,,3))

# Parameters monitored
params <- c("alpha", "beta", "lambda", "resi")

# MCMC settings
ni <- 6000   ;   nt <- 1   ;   nb <- 1000   ;  nc <- 3

# Call WinBUGS or JAGS from R and summarize posteriors
out5 <- bugs(win.data, inits, params, "Poisson_GLM.txt", n.chains = nc, n.thin =
nt, n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory = bugs.dir,
working.directory = getwd())

system.time(out5J <- jags(win.data, inits, params, "Poisson_GLM.txt", n.chains =
nc, n.thin = nt, n.iter = ni, n.burnin = nb))
par(mfrow = c(4,2))    ;    traceplot(out5J, c("alpha[1:4]", "beta[1:4]"))
print(out5J, 3)
                mean    sd    2.5%      50%    97.5% overlap0     f  Rhat n.eff
alpha[1]      -1.028 0.248  -1.526   -1.019   -0.569    FALSE 1.000 1.030    57
alpha[2]      -0.170 0.151  -0.484   -0.166    0.115     TRUE 0.874 1.012 10409
alpha[3]       0.754 0.091   0.569    0.755    0.927    FALSE 1.000 1.001 12407
alpha[4]       1.909 0.049   1.810    1.910    2.002    FALSE 1.000 1.001  2094
beta[1]       -2.294 0.398  -3.068   -2.287   -1.529    FALSE 1.000 1.023    97
beta[2]       -1.926 0.245  -2.418   -1.922   -1.461    FALSE 1.000 1.011  2797
beta[3]       -1.315 0.139  -1.595   -1.313   -1.051    FALSE 1.000 1.002  1818
beta[4]       -0.715 0.082  -0.875   -0.716   -0.555    FALSE 1.000 1.001  5423
```

We produce three residual plots as we did for the normal GLM (Fig. 5–12). They do not look quite perfect; there are more large than small residuals. Since we fit this model for illustration only, we ignore this moderate lack of fit here.

```
par(mfrow = c(1, 3), mar = c(5,5,3,2), cex = 1.3, cex.lab = 1.5, cex.axis = 1.5)
hist(out5$summary[276:542, 1], xlab = "Pearson residuals", col = "grey", breaks
= 50, main = "", freq = F, xlim = c(-5, 5), ylim = c(0, 0.57))
abline(v = 0, col = "red", lwd = 2)
text(-4.7, 0.54, "A", cex = 1.5)

plot(1:267, out5$summary[276:542, 1], main = "", xlab = "Order of data", ylab =
"Pearson residual", frame.plot = F)
abline(h = 0, col = "red", lwd = 2)
text(8, 4, "B", cex = 1.5)

plot(out5$summary[9:275, 1],out5$summary[276:542, 1], main = "", xlab =
"Predicted values", ylab = "Pearson residual", frame.plot = F, xlim = c(-1, 14))
abline(h = 0, col = "red", lwd = 2)
text(-0.5, 4, "C", cex = 1.5)
```

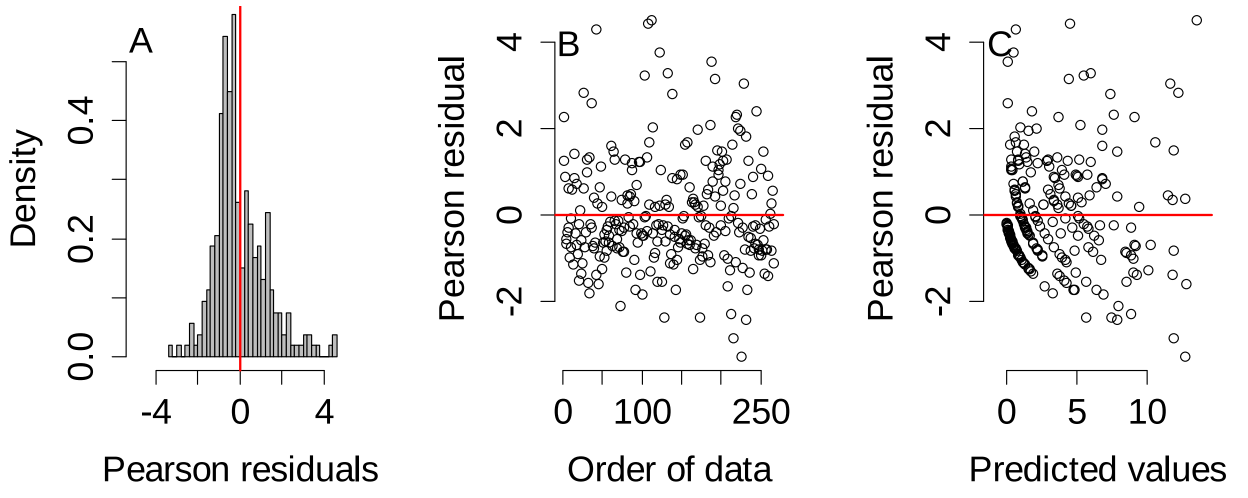


Fig. 5–12: Three diagnostic plots for the Pearson residuals in a Poisson GLM: (A) Histogram of the frequency distribution of the residuals, (B) residuals *vs.* the order of the data and (C) residuals *vs.* predicted values. We could also plot the residuals against the covariates included in the analysis or those left out from the model.

Next, we fit the model using ML and find the usual comforting numerical similarity with the Bayesian estimates. Note how we specify the same means parameterisation of the linear model that we chose in the BUGS fit.

```
summary(glm(Cmax ~ factor(facFor)*elev-1-elev, family = poisson))

Coefficients:
                     Estimate Std. Error z value Pr(>|z|)    
factor(facFor)1      -0.99784    0.26366  -3.785 0.000154 ***
factor(facFor)2      -0.13993    0.15705  -0.891 0.372948    
factor(facFor)3       0.76246    0.08999   8.472  < 2e-16 ***
factor(facFor)4       1.90985    0.04914  38.867  < 2e-16 ***
factor(facFor)1:elev -2.27293    0.41834  -5.433 5.54e-08 ***
factor(facFor)2:elev -1.90978    0.25114  -7.605 2.86e-14 ***
factor(facFor)3:elev -1.31210    0.13730  -9.556  < 2e-16 ***
factor(facFor)4:elev -0.69995    0.08343  -8.389  < 2e-16 ***
```

Derived quantities (= functions of parameters) may be computed inside of BUGS or outside in R, using results from a BUGS model fit. Sometimes it can be advantageous to do such calculations in R after fitting the model in BUGS. We illustrate this using the expected maximum count of great tits, i.e., `lambda`. The Poisson expectation is a deterministic function of the regression parameters `alpha[1:4]` and `beta[1:4]`, for which we have 15,000 posterior samples each. We can use these to get samples of the posterior distribution of `lambda` for each unit in the data set. We create a data structure wherein we store the posterior samples for the newly calculated `lambda`, let's call it `lambda2`. We do this by applying the linear regression with each pair of MCMC samples from the two parameters (`facFor` and `elev`) and back-transforming using the inverse of the log link function. As an aside, note how we can use nested indexing in R exactly as in BUGS.

```
lambda2 <- array(dim = c(15000, 267))
for(j in 1:267){                                 # Loop over sites
   lambda2[,j] <- exp(out5$sims.list$alpha[,facFor[j]] +
out5$sims.list$beta[,facFor[j]] * elev[j]) # linear regression/backtransform
}
plot(out5$sims.list$lambda ~ lambda2, pch = ".")  # Check the two are identical
lm(c(out5$sims.list$lambda) ~ c(lambda2))
```

Finally, let's assume that we want to produce predictions of the expected maximum count *vs.* elevation separately for each forest factor level (but we illustrate this for the first level only). We produce three plots (Fig. 5–13) which illustrate different ways of graphing the point estimate of a prediction along with its uncertainty. Note that we have to sort pairs of elevation and predicted response according to the order of the values of elevation for easy plotting. In the third plot we show the uncertainty around the prediction by plotting a random sample of 50 from their posterior predictive distribution.

```
sorted.ele1 <- sort(elev[facFor == 1])
sorted.y1 <- out5$summary[9:275,][facFor == 1,][order(elev[facFor == 1]),]

# Plot A
par(mfrow = c(1, 3), mar = c(5,5,3,2), cex.lab = 1.5, cex.axis = 1.5)
plot(elev[facFor == 1], jitter(Cmax[facFor ==1]), ylab = "Maximum count", xlab =
"Elevation (scaled)", frame.plot=F), ylim = c(0, 6))
lines(sorted.ele1, sorted.y1[,1], col = "blue", lwd = 2) # Post. mean
lines(sorted.ele1, sorted.y1[,3], col = "grey", lwd = 2) # Lower 95% CL
lines(sorted.ele1, sorted.y1[,7], col = "grey", lwd = 2) # Upper 95% CL
text(-0.8, 6, "A", cex = 2)

# Plot B
plot(sorted.ele1, sorted.y1[,1], type='n', xlab = "Elevation (scaled)", ylab =
"", frame.plot = F, ylim = c(0, 6))
polygon(c(sorted.ele1, rev(sorted.ele1)), c(sorted.y1[,3], rev(sorted.y1[,7])),
col='grey', border=NA)
lines(sorted.ele1, sorted.y1[,1], col = "blue", lwd = 2)
text(-0.8, 6, "B", cex = 2)

# Plot C
elev.pred <- seq(-1,1, length.out = 200)  # Cov. for which to predict lambda
n.pred <- 50                             # Number of prediction profiles
pred.matrix <- array(NA, dim = c(length(elev.pred), n.pred))
for(j in 1:n.pred){
   sel <- sample(1:length(out5$sims.list$alpha[,1]),1) # Choose one post. draw
   pred.matrix[,j] <- exp(out5$sims.list$alpha[sel,1] +
out5$sims.list$beta[sel,1] * elev.pred)
}
plot(sorted.ele1, sorted.y1[,1], type='n', xlab = "Elevation (scaled)", ylab =
"", frame.plot = F, ylim = c(0, 6))
matlines(elev.pred, pred.matrix, col = "grey", lty = 1, lwd = 1)
lines(sorted.ele1, sorted.y1[,1], col = "blue", lwd = 2)
text(-0.8, 6, "C", cex = 2)
```

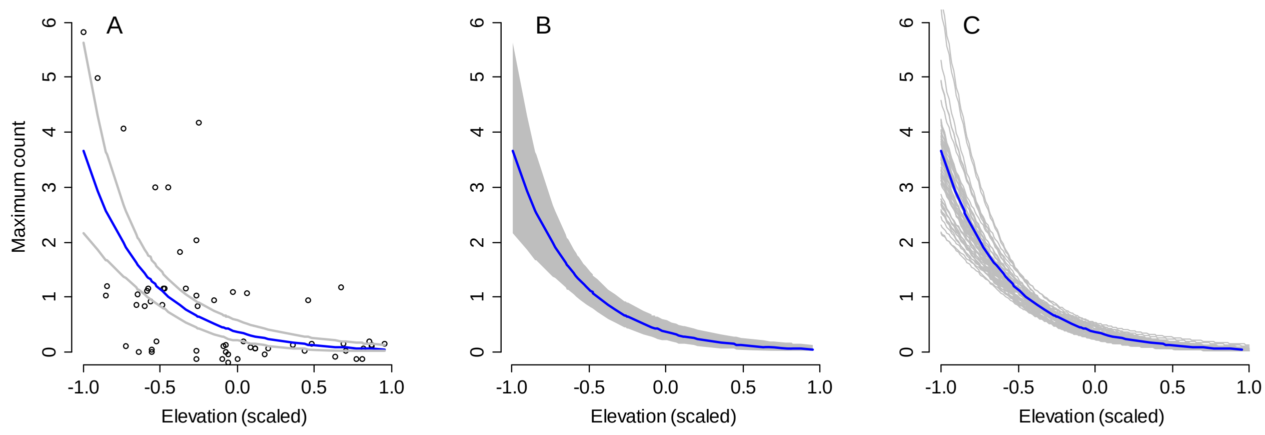


Fig. 5–13: Three graphical summaries of the Poisson ANCOVA model fitted to the maximum count of great tits at each site to scaled elevation and the forest cover factor (shown for level 1 of the forest factor only; analogous plots could be produced for levels 2–4). Posterior means of the predicted regression line of the mean tit count on scaled elevation are shown in blue and the uncertainty around the prediction is depicted in grey (A and B: 95% CRI; C: a random sample of 50 of the posterior samples are shown). Note that the credible interval is for the expected count, not for the observed counts; hence, unsurprisingly, in the left plot most of the observed data lie outside of the interval (moreover, they are slightly jittered).

5.11 Binomial generalised linear model (binomial GLM, logistic regression)

We next use BUGS to fit a logistic regression, also known as a binomial GLM. For this, we first quantize the great tit counts from one survey to obtain zeros and ones, indicating whether a count is zero or greater than zero. At several places in this book we emphasize that this is exactly how "presence/absence" or "detection/nondetection" data often arise in practice, namely as a simple summary of an abundance distribution (or here, of a distribution of counts). Once more we fit the model using Bayesian and ML methods to emphasize their numerical similarity. We actually fit a Bernoulli GLM (i.e., a binomial GLM with trial size 1), but binomial GLMs with trial sizes >1 occur throughout the book, for instance in chapter 6. We fit the following model to $y[,1]$, the detection/nondetection data for the first survey:

$$y_{i,1} \sim Bernoulli(\phi_i)$$
$$\text{logit}(\phi_i) = \alpha_{0,j} + \alpha_{1,j} * elev_i$$

As before, *j* indexes the four levels of the forest factor.

```
# Quantize counts from first survey and describe
y1 <- as.numeric(C[,1] > 0)  # Gets 1 if first count greater than zero
table(y1)
y1
  0   1
137 130

mean(N > 0)           # True occupancy
mean(y1)              # Observed occupancy after first survey
```

Hence, true occupancy (proportion of occupied sites) is 77%, while the observed occupancy after the first survey is only 49%. We fit the ANCOVA linear model to the binary detection/nondetection response by specifying a Bernoulli GLM.

```
# Bundle data
win.data <- list(y1 = y1, M = length(y1), elev = elev, facFor = facFor)

# Specify model in BUGS language
cat(file = "Bernoulli_GLM.txt","
model {

# Priors
for(k in 1:4){
   alpha[k] <- logit(mean.psi[k])     # intercepts
   mean.psi[k] ~ dunif(0,1)
   beta[k] ~ dnorm(0, 1.0E-06)        # slopes
}

# Likelihood
for (i in 1:M){
   y1[i] ~ dbern(theta[i])
   logit(theta[i]) <- alpha[facFor[i]] + beta[facFor[i]] * elev[i]
}
}
")

# Initial values
inits <- function() list(mean.psi = runif(4), beta = rnorm(4,,3))   # Priors 2

# Parameters monitored
params <- c("mean.psi", "alpha", "beta", "theta")

# MCMC settings
ni <- 6000   ;   nt <- 1   ;   nb <- 1000   ;  nc <- 3

# Call WinBUGS or JAGS from R (ART <1 min)
out6 <- bugs(win.data, inits, params, "Bernoulli_GLM.txt", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory =
bugs.dir, working.directory = getwd())

out6J <- jags(win.data, inits, params, "Bernoulli_GLM.txt", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb)
par(mfrow = c(4,2))    ;    traceplot(out6J, c("alpha[1:4]", "beta[1:4]"))
```

We can summarize the posterior distributions for the logit-linear regression parameters, though we can no longer directly compare them with the truth, since the data were generated under a different model than the one fitted.

```
print(out6, 2)
              mean   sd   2.5%    25%    50%    75%  97.5% Rhat n.eff
mean.psi[1]   0.17 0.06   0.08   0.13   0.17   0.21   0.29    1  3900
mean.psi[2]   0.25 0.06   0.14   0.21   0.25   0.30   0.39    1  4000
mean.psi[3]   0.54 0.06   0.43   0.50   0.54   0.58   0.66    1 15000
mean.psi[4]   0.82 0.05   0.73   0.80   0.83   0.86   0.91    1 15000
alpha[1]     -1.62 0.41  -2.50  -1.87  -1.59  -1.33  -0.88    1  4100
alpha[2]     -1.11 0.35  -1.82  -1.33  -1.10  -0.87  -0.47    1  4000
alpha[3]      0.17 0.24  -0.29   0.01   0.17   0.33   0.65    1 15000
alpha[4]      1.58 0.33   0.99   1.36   1.57   1.79   2.26    1 15000
beta[1]      -1.98 0.78  -3.62  -2.47  -1.93  -1.43  -0.57    1  2700
beta[2]      -2.56 0.74  -4.12  -3.05  -2.53  -2.05  -1.21    1  8500
beta[3]      -0.95 0.42  -1.77  -1.22  -0.94  -0.66  -0.13    1 11000
```

```
beta[4]       0.49 0.59  -0.63   0.10   0.49   0.88   1.68    1 15000
theta[1]      0.64 0.07   0.50   0.59   0.64   0.68   0.76    1 15000
theta[2]      0.66 0.07   0.52   0.62   0.67   0.71   0.79    1 15000
theta[3]      0.85 0.06   0.73   0.81   0.85   0.89   0.94    1 15000
[ ... ]

# Compare with MLEs
summary(glm(y1 ~ factor(facFor)*elev-1-elev, family = binomial))

[ .... ]
Coefficients:
                     Estimate Std. Error z value Pr(>|z|)
factor(facFor)1       -1.6194     0.4172  -3.881 0.000104 ***
factor(facFor)2       -1.1122     0.3481  -3.195 0.001398 **
factor(facFor)3        0.1774     0.2392   0.742 0.458303
factor(facFor)4        1.5873     0.3307   4.801 1.58e-06 ***
factor(facFor)1:elev  -1.9277     0.7860  -2.453 0.014180 *
factor(facFor)2:elev  -2.4480     0.7260  -3.372 0.000747 ***
factor(facFor)3:elev  -0.9154     0.4125  -2.219 0.026477 *
factor(facFor)4:elev   0.4924     0.5821   0.846 0.397600
[ .... ]
```

We overlay the estimated regression equations onto plots of the observed data (jittered for enhanced readability), see Fig. 5–16. Note the slight contortions to get the ordering of the x and y variables right.

```
# Plot of observed response vs. two covariates
par(mfrow = c(1, 2), mar = c(5,5,3,2), cex.lab = 1.5, cex.axis = 1.5)
F1 <- facFor == 1 ; F2 <- facFor == 2 ; F3 <- facFor == 3 ; F4 <- facFor == 4
plot(jitter(y1,,0.05) ~ facFor, xlab = "Forest factor", ylab = "Observed
occupancy probability", frame.plot = F, ylim = c(0, 1.15))
lines(1:4, out6$summary[1:4,1], lwd = 2)
segments(1:4, out6$summary[1:4,3], 1:4, out6$summary[1:4,7])
text(1.15, 1.1, "A", cex=1.6)

plot(elev[F1], jitter(y1,,0.1)[F1], xlab = "Elevation", ylab = "", col = "red",
frame.plot = F)
points(elev[F2], jitter(y1,,0.05)[F2], col = "blue")
points(elev[F3], jitter(y1,,0.05)[F3], col = "green")
points(elev[F4], jitter(y1,,0.05)[F4], col = "grey")
lines(sort(elev[F1]), out6$mean$theta[F1][order(elev[F1])], col="red", lwd=2)
lines(sort(elev[F2]), out6$mean$theta[F2][order(elev[F2])], col="blue", lwd=2)
lines(sort(elev[F3]), out6$mean$theta[F3][order(elev[F3])], col="green", lwd=2)
lines(sort(elev[F4]), out6$mean$theta[F4][order(elev[F4])], col="grey", lwd=2)
text(-0.9, 1.1, "B", cex=1.6)
```

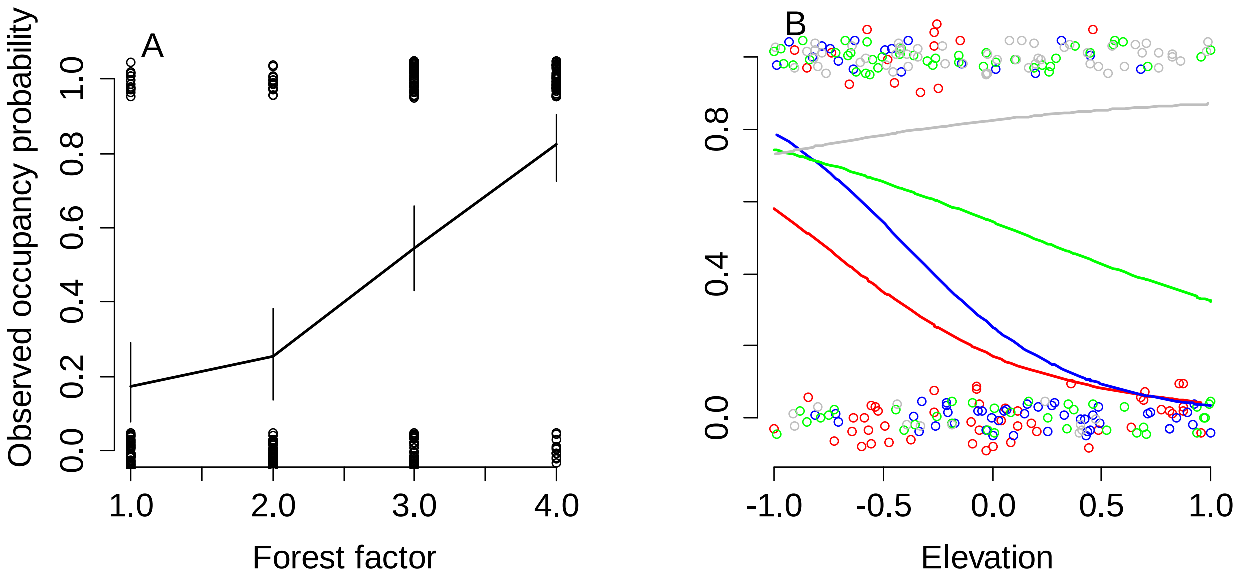


Fig. 5–16: Predictions of the observed occupancy probability of simulated great tit data under the Bernoulli model, circles are observed data (jittered). (A) observed detection/nondetection data vs. forest factor, line shows estimates of theta (with 95% CRI) at average scaled elevation. (B) Observed and estimated relationship with elevation for all four levels of the forest factor (1 – red, 2 – blue, 3 – green, 4 – grey).

5.13 Random-effects Poisson GLM (Poisson GLMM)

In the concluding two examples of the BUGS language, we illustrate the specification of random effects. Random effects are a defining feature of hierarchical models, and the BUGS language makes it particularly transparent what they are: realisations from an un- or partially observed random variable. Or in other words: random effects are parameters or latent variables that are given a distribution with (hyper-)parameters that are estimated from the data.

We illustrate in the context of a hierarchical model that is somewhat related to the N-mixture model (see chapter 6): a Poisson regression, or GLM, with random site effects fitted to all three simulated counts of great tits per site (Dennis et al. 2015, also unpublished manuscripts by W.A. Link and J. Knape et al.). We include a random site effect, perhaps to avoid pseudoreplication and to account for the correlation of replicated counts made at the same site. In contrast to the N-mixture model, which is a Poisson-binomial mixture, this model is a Poisson-normal mixture model. Because we no longer aggregate the repeated measures, we can now fit both site covariates (elevation and forest cover) and sampling covariates (wind speed). We fit the following generalized linear mixed model (GLMM) to the counts C_{ij} of great tits at site *i* during replicate survey *j*:

$$C_{ij} \sim Poisson(\lambda_{ij})$$
$$\log(\lambda_{ij}) = \alpha_{0,i} + \alpha_1 * elev_i + \alpha_2 * forest_i + \alpha_3 * elev_i * forest_i + \alpha_4 * wind_{ij}$$
$$\alpha_{0,i} \sim Normal(\mu_\alpha, \sigma_\alpha^2)$$

It is the last line that defines the regression intercepts as random effects: the `alpha0` parameters are defined to be draws from a normal distribution with mean and variance as hyperparameters that are estimated.

We write the code in a slightly less verbose format now. We will also compute, as a derived quantity in the BUGS code, a zero-centered version of the random site effects `alpha0`, i.e., subtract the value of their hyper-mean, for direct comparison with the frequentist analyses below. Note that much longer chains are required to achieve convergence in random effects models.

```
# Bundle data
win.data <- list(C = C, M = nrow(C), J = ncol(C), elev = elev, forest = forest,
elev.forest = elev * forest, wind = wind)

# Specify model in BUGS language
cat(file = "RE.Poisson.txt","
model {

# Priors
mu.alpha ~ dnorm(0, 0.001)                 # Mean hyperparam
tau.alpha <- pow(sd.alpha, -2)
sd.alpha ~ dunif(0, 10)                    # sd hyperparam
for(k in 1:4){
   alpha[k] ~ dunif(-10, 10)               # Regression params
}

# Likelihood
for (i in 1:M){
   alpha0[i] ~ dnorm(mu.alpha, tau.alpha) # Random effects and hyperparams
   re0[i] <- alpha0[i] - mu.alpha         # zero-centered random effects
   for(j in 1:J){
      C[i,j] ~ dpois(lambda[i,j])
      log(lambda[i,j]) <- alpha0[i] + alpha[1] * elev[i] + alpha[2] * forest[i]
+ alpha[3] * elev.forest[i] + alpha[4] * wind[i,j]
   }
}
}")

# Other model run preparations
inits <- function() list(alpha0 = rnorm(M), alpha = rnorm(4)) # Inits
params <- c("mu.alpha", "sd.alpha", "alpha0", "alpha", "re0") # Params
ni <- 30000 ; nt <- 25 ; nb <- 5000 ; nc <- 3                 # MCMC settings

# Call WinBUGS or JAGS from R (ART 6-7 min) and summarize posteriors
out8 <- bugs(win.data, inits, params, "RE.Poisson.txt", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory =
bugs.dir, working.directory = getwd())

out8 <- jags(win.data, inits, params, "RE.Poisson.txt", n.chains = nc, n.thin =
nt, n.iter = ni, n.burnin = nb)
par(mfrow = c(3,2))  ;  traceplot(out8, c("mu.alpha", "sd.alpha", "alpha[1:3]"))

print(out8, 3)
                 mean    sd    2.5%     50%   97.5% overlap0     f  Rhat n.eff
mu.alpha       -0.839 0.072  -0.980  -0.840  -0.699    FALSE 1.000 1.001  2756
sd.alpha        0.267 0.049   0.169   0.268   0.362    FALSE 1.000 1.003  3000
alpha0[1]      -0.856 0.218  -1.306  -0.847  -0.444    FALSE 1.000 1.001   796
alpha0[2]      -0.910 0.208  -1.330  -0.903  -0.522    FALSE 1.000 1.004   650
alpha0[3]      -1.216 0.215  -1.657  -1.210  -0.809    FALSE 1.000 1.002  3000
[ ... ]
alpha0[265]    -0.845 0.269  -1.409  -0.839  -0.340    FALSE 0.999 1.003   950
alpha0[266]    -0.909 0.273  -1.466  -0.894  -0.387    FALSE 0.999 1.002  1298
alpha0[267]    -0.836 0.276  -1.386  -0.831  -0.301    FALSE 0.999 1.001  3000
alpha[1]       -1.694 0.108  -1.900  -1.695  -1.486    FALSE 1.000 1.002  2944
```

```
alpha[2]       2.128  0.096   1.943   2.130   2.315   FALSE 1.000 1.003  2938
alpha[3]       1.390  0.161   1.076   1.391   1.702   FALSE 1.000 1.000  2721
alpha[4]      -1.711  0.063  -1.837  -1.711  -1.590   FALSE 1.000 1.001  3000
re0[1]        -0.017  0.206  -0.437  -0.014   0.386    TRUE 0.529 1.001   997
re0[2]        -0.071  0.195  -0.475  -0.064   0.298    TRUE 0.637 1.002   821
re0[3]        -0.377  0.195  -0.783  -0.368  -0.013   FALSE 0.981 1.001  3000
[ ... ]
re0[265]      -0.006  0.262  -0.534   0.003   0.491    TRUE 0.495 1.003  1243
re0[266]      -0.069  0.262  -0.598  -0.061   0.445    TRUE 0.596 1.002  1776
re0[267]       0.003  0.266  -0.544   0.003   0.524    TRUE 0.506 1.001  3000
```

If your computer runs out of memory, try running the model without the zero-centered random effects in the params list and compute the posterior samples for "re0" in R. You might also want to thin more to save fewer samples per parameter.

At last, we fit this model non-Bayesianly using maximum likelihood with package lme4 (Bates *et al.* 2014). Function glmer requires the data to be input in a vector format, rather than as an array. Therefore, we first reformat our data set. Note how the frequentist model fit is much faster than using MCMC (though, admittedly, our MCMC settings are a little overkill).

```
Cvec <- as.vector(C)            # Vector of M*J counts
elev.vec <- rep(elev, J)        # Vectorized elevation covariate
forest.vec <- rep(forest, J)    # Vectorized forest covariate
wind.vec <- as.vector(wind)     # Vectorized wind covariate
fac.site <- factor(rep(1:M, J)) # Site indicator (factor)
cbind(Cvec, fac.site, elev.vec, forest.vec, wind.vec) # Look at data

# Fit same model using maximum likelihood (NOTE: glmer uses ML instead of REML)
library(lme4)
summary(fm <- glmer(Cvec ~ elev.vec*forest.vec + wind.vec + (1| fac.site),
family = poisson))              # Fit model
ranef(fm)                       # Print zero-centered random effects

Generalized linear mixed model fit by maximum likelihood (Laplace Approximation)
['glmerMod']
 Family: poisson ( log )
Formula: Cvec ~ elev.vec * forest.vec + wind.vec + (1 | fac.site)
[ ... ]
Random effects:
 Groups   Name        Variance Std.Dev.
 fac.site (Intercept) 0.06461  0.2542
Number of obs: 801, groups:  fac.site, 267

Fixed effects:
                    Estimate Std. Error z value Pr(>|z|)
(Intercept)         -0.83243    0.07252  -11.48   <2e-16 ***
elev.vec            -1.69303    0.10480  -16.16   <2e-16 ***
forest.vec           2.12447    0.09546   22.25   <2e-16 ***
wind.vec            -1.70929    0.06326  -27.02   <2e-16 ***
elev.vec:forest.vec  1.39146    0.15582    8.93   <2e-16 ***
[ ... ]
```

We compare the fixed-effects estimates in a table below, and also the random-effects estimates from the Bayesian and the non-Bayesian analyses in a table and a graph (Fig. 5–17A).

```
# Compare fixed-effects estimates (in spite of the confusing naming in glmer
output), Bayesian post. means and sd left, frequentist MLEs and SEs right
print(cbind(out8$summary[c(1:2, 270:273), 1:2], rbind(summary(fm)$coef[1,1:2],
c(sqrt(summary(fm)$varcor$fac.site), NA), summary(fm)$coef[c(2,3,5,4),1:2])), 3)
           mean     sd Estimate Std. Error
mu.alpha -0.837 0.0758   -0.832     0.0725
sd.alpha  0.264 0.0507    0.254         NA
```

```
alpha[1] -1.693 0.1079   -1.693     0.1048
alpha[2]  2.126 0.0976    2.124     0.0955
alpha[3]  1.389 0.1602    1.391     0.1558
alpha[4] -1.712 0.0650   -1.709     0.0633

# Compare graphically non-Bayesian and Bayesian random effects estimates
Freq.re <- ranef(fm)$fac.site[,1]         # Non-Bayesian estimates (MLEs)
Bayes.re <- out8$summary[274:540,]        # Bayesian estimates

par(mfrow = c(1, 2), mar = c(5,5,3,2), cex.lab = 1.5, cex.axis = 1.5)
plot(Freq.re, Bayes.re[,1], xlab = "Non-Bayesian (glmer)", ylab = "Bayesian
(BUGS)", xlim = c(-0.4, 0.42), ylim = c(-2, 2), frame.plot = F, type = "n")
segments(Freq.re, Bayes.re[,3], Freq.re, Bayes.re[,7], col = "grey", lwd = 0.5)
abline(0, 1, lwd = 2)
points(Freq.re, Bayes.re[,1])
text(-0.38, 2, "A", cex=1.6)
```

Once again, we note what many consider to be a comforting similarity between the non-Bayesian and the Bayesian parameter estimates, even with respect to their uncertainty (SEs and posterior standard deviations, respectively). In the Bayesian analysis, exact (i.e., not asymptotic) uncertainty estimates of the random effects are obtained easily (the grey error bars in Fig. 5–17 A), while for the non-Bayesian estimate, these would be harder to get (presumably, by bootstrapping).

Finally, we plot predictions of the relationship between the expected counts and wind speed for each site separately and at a determined value of the other covariates (elevation and forest cover; Fig. 5–17B). We could use the posterior samples of all parameters to produce uncertainty intervals as well, but this would be graphically unwieldy; hence, we plot only the posterior means. We write the entire linear predictor, including the effects of the three other covariates set at 0 for clarity.

```
wind.pred <- seq(-1, 1, , 1000)     # Covariate values for prediction
pred <- array(NA, dim = c(1000, 267))
for(i in 1:267){
   pred[,i]<- exp(out8$mean$alpha0[i] + out8$mean$alpha[1] * 0 +
out8$mean$alpha[2] * 0 + out8$mean$alpha[3] * 0 + out8$mean$alpha[4] *
wind.pred)    # Predictions for each site
}

matplot(wind.pred, pred, type = "l", lty = 1, col = "grey", xlab = "Wind speed",
ylab = "Expected count", frame.plot = F, ylim = c(0, 4))
lines(wind.pred, exp(out8$mean$mu.alpha + out8$mean$alpha[4] * wind.pred), col =
"black", lwd = 3)
text(-0.9, 4, "B", cex=1.6)
```

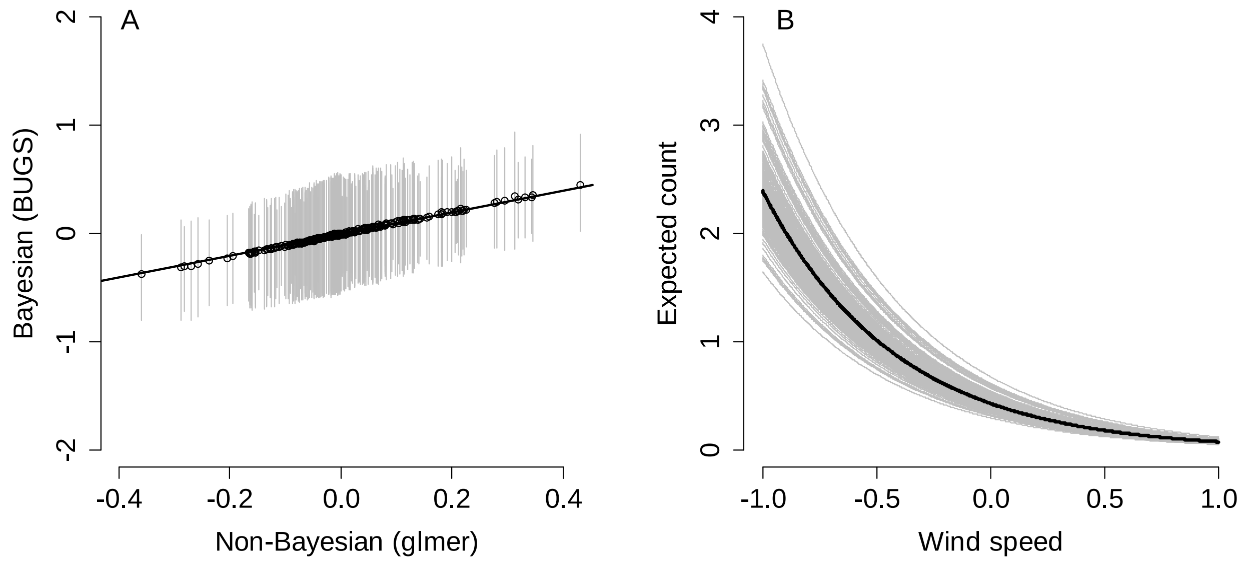


Fig. 5–17: (A) Estimates of random site intercepts from a Bayesian (with 95% CRI in grey) and a non-bayesian analysis of the Poisson GLMM. (B) Bayesian estimates of site-specific relationships between expected count of great tits and wind speed (black line is the population average).

Note that for hierarchical models, it is not straightforward to compute a quantity like the proportion of explained variation (R^2). The reason is that one may define such quantities for every level in the hierarchy of the model. For an attempt at defining R^2-like quantities in hierarchical models, see Gelman & Pardoe (2006).

Often, fitting a model is only the first step of an analysis and much can be learnt by summarizing the results of that fit. This is true especially of a Bayesian analysis, where rich insights are possible based on the full posterior distributions and functions thereof, as we have already seen. The ability to make direct probability statements about parameters and the ease with which derived quantities such as predictions of the response for selected values of covariates can be computed, with a full assessment of the uncertainty, are great and underused benefits of a Bayesian analysis using MCMC.

5.14 Random-effects binomial GLM (binomial GLMM)

We conclude our overview of 'standard' GLMs and GLMMs by fitting a random-effects model with binomial response. We convert the counts for each survey at a site into 'presence/absence' or, more accurately, 'detection/nondetection', observations, i.e., 'squash' the counts into zeros and ones, depending on whether they are equal to zero or greater.

```
# Get detection/nondetection response
y <- C
y[y > 0] <- 1
```

We then fit a kind of 'naive' site-occupancy model, where we describe the detection/nondetection observations by a logistic regression with a continuous site-specific random effect assumed to be drawn from a normal distribution. This means, that we assume that the effect of wind is specific to each site, but that the slopes of wind among all sites cluster around some common mean and with a variance that can be estimated. In contrast to the site-occupancy model (see chapter 10), which is a Bernoulli-Bernoulli mixture, this model is a Bernoulli-normal mixture. We fit the following GLMM to the binary detection-nondetection observations y_{ij} at site *i* during replicate survey *j*:

$$y_{ij} \sim Bernoulli(\phi_{ij})$$
$$\text{logit}(\phi_{ij}) = \alpha_{0,i} + \alpha_1 * elev_i + \alpha_2 * forest_i + \alpha_3 * elev_i * forest_i + \alpha_{4,i} * wind_{ij}$$
$$\alpha_{0,i} \sim Normal(\mu_{\alpha 0}, \sigma^2_{\alpha 0})$$
$$\alpha_{4,i} \sim Normal(\mu_{\alpha 4}, \sigma^2_{\alpha 4})$$

For illustration, we fit a slightly more complicated random-effects model than in the previous section, since we now specify the coefficients of wind to be random, by defining them to be draws from another (normal) prior distribution, with hyperparameters that we estimate. Other than that, we keep the linear model of this Bernoulli GLMM identical to that of the Poisson GLMM in the previous section, to emphasize that the linear model can be chosen entirely separately from the distribution chosen to describe the randomness in the response.

```
# Bundle data
win.data <- list(y = y, M = nrow(y), J = ncol(y), elev = elev, forest = forest,
elev.forest = elev * forest, wind = wind)
str(win.data)
```

In the BUGS code, we have to take the coefficient `alpha4` out of the vector `alpha`, since it is no longer a scalar, but a vector itself.

```
# Specify model in BUGS language
cat(file = "RE.Bernoulli.txt","
model {

# Priors
mu.alpha0 <- logit(mean.theta)              # Random intercepts
mean.theta ~ dunif(0,1)
tau.alpha0 <- pow(sd.alpha0, -2)
sd.alpha0 ~ dunif(0, 10)
mu.alpha4 ~ dnorm(0, 0.001)                 # Random slope on wind
tau.alpha4 <- pow(sd.alpha4, -2)
sd.alpha4 ~ dunif(0, 10)
for(k in 1:3){
   alpha[k] ~ dnorm(0, 0.001)               # Slopes
}

# Likelihood
for (i in 1:M){
   alpha0[i] ~ dnorm(mu.alpha0, tau.alpha0) # Intercept random effects
   re00[i] <- alpha0[i] - mu.alpha0         # same zero-centered
   alpha4[i] ~ dnorm(mu.alpha4, tau.alpha4) # Slope random effects
   re04[i] <- alpha4[i] - mu.alpha4         # same zero-centered
   for(j in 1:J){
      y[i,j] ~ dbern(theta[i,j])
      logit(theta[i,j]) <- alpha0[i] + alpha[1] * elev[i] + alpha[2] * forest[i]
+ alpha[3] * elev.forest[i] + alpha4[i] * wind[i,j]
   }
}
}")

# Other model run preparations
inits <- function() list(alpha0 = rnorm(M), alpha4 = rnorm(M))# Inits
params <- c("mu.alpha0", "sd.alpha0", "alpha0", "alpha", "mu.alpha4",
"sd.alpha4", "alpha4", "re00", "re04")                      # Params
```

```
ni <- 30000 ; nt <- 25 ; nb <- 5000 ; nc <- 3                # MCMC settings

# Call WinBUGS from R .... and crash !
out9 <- bugs(win.data, inits, params, "RE.Bernoulli.txt", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory =
bugs.dir, working.directory = getwd())
```

While WinBUGS gets an "undefined real result" crash, JAGS works fine; this is an example that shows that JAGS is sometimes numerically more robust than WinBUGS (We could get WinBUGS to run by adding into the model some numerical "stabilisation", see trick 15 in Appendix 1 of Kéry & Schaub, 2012, but we won't.)

```
# Call JAGS from R (ART 2.5 min)
out9 <- jags(win.data, inits, params, "RE.Bernoulli.txt", n.chains = nc, n.thin
= nt, n.iter = ni, n.burnin = nb)
par(mfrow = c(2,2))
traceplot(out9, c("mu.alpha0", "sd.alpha0", "alpha[1:3]", "mu.alpha4",
"sd.alpha4"))
print(out9, 3)
```

Again, if you run into memory problems, don't save the posterior samples for the zero-centered random effects but compute them in R instead. Finally, we fit this model non-Bayesianly using the maximum likelihood in the package lme4 after vectorizing the data.

```
yvec <- as.vector(y)            # Vector of M*J counts
elev.vec <- rep(elev, J)        # Vectorized elevation covariate
forest.vec <- rep(forest, J)    # Vectorized forest covariate
wind.vec <- as.vector(wind)     # Vectorized wind covariate
fac.site <- factor(rep(1:M, J)) # Site indicator (factor)
cbind(yvec, fac.site, elev.vec, forest.vec, wind.vec) # Look at data

# Fit same model using maximum likelihood
library(lme4)                   # Load package
summary(frem <- glmer(yvec ~ elev.vec*forest.vec + wind.vec + (wind.vec ||
fac.site), family = binomial))            # Fit model

Random effects:
 Groups     Name        Variance Std.Dev.
 fac.site   (Intercept) 4.255    2.063
 fac.site.1 wind.vec    8.779    2.963
Number of obs: 801, groups:  fac.site, 267

Fixed effects:
                    Estimate Std. Error z value Pr(>|z|)
(Intercept)          -0.6381     0.2504  -2.548 0.010837 *
elev.vec             -3.8703     0.7855  -4.927 8.35e-07 ***
forest.vec            5.7574     1.0614   5.424 5.82e-08 ***
wind.vec             -5.9007     1.1714  -5.037 4.72e-07 ***
elev.vec:forest.vec   3.8181     1.0009   3.815 0.000136 ***

# Compare Bayesian and non-Bayesian estimates
print(out9$summary[c(1:2, 270:274),c(1:3,7:9)], 4)
             mean     sd    2.5%   97.5%  Rhat n.eff
mu.alpha0 -0.6091 0.2414 -1.1101 -0.1457 0.9996  3000
sd.alpha0  2.0662 0.4821  1.2639  3.2134 1.0014  3000
alpha[1]  -3.8639 0.6980 -5.5057 -2.7587 1.0032  1633
alpha[2]   5.7575 0.9339  4.3289  7.9514 1.0057  1322
alpha[3]   3.7679 0.9383  2.1356  5.8646 1.0001  3000
mu.alpha4 -5.8058 1.0006 -8.1132 -4.3193 1.0067  1443
sd.alpha4  2.4177 1.0510  0.5137  4.6309 1.0034  1086
```

We find slightly less agreement between the two analyses now than in all the previous examples. This is perhaps not surprising given that binary data contain much less information about all the parameters of this fairly complex model. Next, we look at the estimates of the random effects.

```
(re <- ranef(frem))                     # Print zero-centered random effects
$fac.site
       (Intercept)         wind.vec
1    0.01538103261 -0.5521873386140
2    0.13475066011 -0.0394227279709
3   -1.07204621369 -1.4696483198851
[...]
265  0.63961927035  2.0279735503384
266 -1.07504683084 -0.1964205985223
267 -0.00131161668  0.0019269593948
```

5.15 General strategy of model building with BUGS

We have introduced statistical modeling with BUGS for some of the simplest possible models: GLMs with just a few covariates and with 0–2 random effects. When you start modeling your real data, there are a *lot* of things that can, and will frequently, go wrong. Of course, many problems can be avoided with a little experience. So can we give a few tips to avoid the many possible pitfalls in BUGS modeling ?

Unfortunately, this is not possible in general. We have given a list of BUGS survival tips elsewhere (see appendix in Kéry & Schaub, 2012). No doubt there will be much additional anecdotal wisdom available *somewhere*, which in some cases may be essential to success. Here we mention what we believe are some of the most important, general tips.

(1) *By far* the single most important BUGS modeling tip is: ***Always start from the simplest possible version of your problem !*** Strip down your model to its bare bones and toss out any non-essential details at the start, such as covariates, time-dependence, or spatial or temporal autocorrelation. Start with a very simple caricature of the model you want to get at; in a sense, *start with a model of your model*. Once you get this simple model to run and you understand it, add details one at a time until you reach the desired model structure. This step-wise approach will help you to recognize and diagnose problems when something goes wrong, e.g., when some parameter estimates change dramatically when going from one model to a slight variant. Similarly, when something goes wrong in your modeling, go back to the last (simpler) model variant that did work and then try to identify what caused the problem in a new step-up approach. Arguably, this advice is essential for any kind of even moderately advanced statistical modeling and we acknowledge Gelman et al. (2004), who stress this in their book. There is also a heuristic advantage of such a step-up modeling strategy that carries over to all science: there can hardly ever be a point in trying to understand a big version of a problem unless you first understand the simplest version of the problem. Thus, this modeling strategy is just a variant of a powerful model for the process of learning in general.

(2) *Start with a template that runs.* Only very rarely will you write code for a new analysis from scratch; it is much too easy to get bogged down in a myriad of errors that are so easy to commit. Rather, it is usually far more efficient to take code for a version of your problem or for a related problem that you developed earlier, that a colleague has given to you or that you found in a book or on the internet, and then adapt that code towards the model you want to fit.

(3) *Play with simulated data first.* If you get the right answers and understand your model, then you will be much better prepared to fit the model to your real data and understand it.

(4) *Don't forget to google cryptic error messages* (B. Schmidt, pers. comm.).

(5) *Write tidy code* (inspired by B. Schmidt).

5.16 Summary and outlook

In this chapter we have covered much ground. In an applied way we have wrapped up much of the contents of chapters 2–4. Most importantly, we have presented a crash course in BUGS, which we use throughout the book to fit hierarchical models. All three BUGS incarnations (`WinBUGS`, `OpenBUGS`, `JAGS`) use virtually the same model defition language, the BUGS language, and hence, from R you can readily send a model to either of them. Indeed, it often pays to try a model in more than one BUGS engine: sometimes an analysis does not work in one, but will work in another, and typically one engine is faster. We have also introduced our typical workflow of a Bayesian analysis using BUGS run from R. We have illustrated many times the richness of posterior inferences in a Bayesian analysis. We fitted all models with maximum likelihood as well, thereby emphasizing that with reasonable sample size, Bayesian estimates with vague priors agree numerically very well with the corresponding MLEs. We have covered several types of linear models, GLMs and two simple random effects, or mixed, models (GLMMs).

To emphasize the value of generating and analysing simulated data sets, we have worked exclusively with a simulated data set, generated using the code from chapter 4, and imagined these were counts of a small Eurasian passerine bird. We have encountered various ways in which such counts can be summarized or aggregated, e.g., to obtain presence/absence or species distribution data from the underlying counts. Finally, we have also covered quite an eclectic set of other topics which we think are important in applied Bayesian modeling, including prediction, goodness of fit and residual diagnostics, the Bayesian treatment of missing values, proportion of variance explained, specification of non-standard likelihoods using the zeros and the ones trick, and of alternative parameterisations of a model using moment matching (Hobbs & Hooten 2015). One potentially important topic that we have not touched upon in this chapter is model selection (see Burnham & Anderson, 2002; Kadane & Lazaar, 2004; O'Hara & Sillanpää, 2009; Tenan et al., 2014; Hobbs & Hooten, 2015), but we introduced it in chapter 2 and illustrate aspects of this throughout the book. We do not claim to cover any of these topics in any exhaustion. If Bayesian analysis, linear models, GLMs, or simple mixed models, and BUGS software is completely new to you, then you should complement your reading of this book with that of other books on these topics. At the very least, you should read the preceding chapters 2–4 a couple of times.

We don't claim that all of the models in this chapter were the best possible analysis for our data set. For instance, the Poisson GLMM in section 5.13 would probably not be our first choice; rather, we might adopt a binomial N-mixture model (see chapter 6) for inference about abundance from such data. Similarly, the binomial GLMM in section 5.14 is not the most natural model for such data. Instead, we would naturally adopt a site-occupancy model (see chapter 11) for inference about species occurrence from such data.

We wanted to introduce BUGS with the simplest models that you are likely to know well and use a lot: linear models, GLMs and simple mixed models. These form the backbone of most of applied statistics in ecology, and yet, there are many aspects that ecologists may find confusing, e.g., the parameterisation of factors, link functions and random effects. The BUGS language can greatly enhance your understanding of these fundamental statistical models, because of the way in which these models are described in the BUGS language. BUGS may be an algorithmic black box, but it is the opposite to a modeling black box: in the BUGS language, the model specified is utterly transparent.

Nowadays, a certain class of mixed models in particular have become the workhorse models of many ecologists, and they are widely fitted using functions such as `(g)lmer` in R or `PROC MIXED` in SAS. You can think of BUGS, with its simple but comprehensive model definition language, as a super-powerful `lmer` function for fitting virtually *any* kind of mixed (=hierarchical) model. The power of BUGS extends way beyond the 'simple' mixed models with normal random effects, which are the only ones you can fit with `(g)lmer`. It includes models with many nested levels of random effects, random effects that are nested or crossed , continuous or discrete-valued and can come from many other distributions than just a normal. Thus, BUGS allows you to easily specify a very large class of hierarchical (=mixed) models.

In almost all of the rest of the book, we will combine GLMs like those in this chapter to build hierarchical models that exactly reflect the way in which we imagine the sequence of processes (e.g., ecological and observational) occur that produce the observed data. Hence, if you understand linear models, GLMs and the concept of random effects, and know how to specify these modeling figures using the BUGS language, then you are very well prepared for building a vast number of HMs in a powerful and creative manner. Moreover, if you learn how to "walk" in BUGS, then you are likely to experience a dramatic modeling freedom that you would never have dreamt of as an ecologist.

In other words, almost all of the rest of the book presents variants of hierarchical models consisting of combinations of two to a couple of submodels, each of which can be described as a simple GLM in a sense. Their specific combination of GLM represents the basic structure of the particular model. Everything else then boils down to clever specification of a linear predictor to address the specific question and the specifics of your data collection protocol. In summary, a very good practical understanding of linear modeling and of GLMs is vital for your existence as a hierarchical modeler.

5.17 Exercises

1. In section 5.3, fit the linear model which you get when typing in R `lm(Cmean ~ (elev + I(elev^2)) * (forest + I(forest^2)))`, i.e., add quadratic effects of elevation and forest cover including all pair-wise interaction effects.
2. In section 5.6, the residuals are seen to be heterogeneous in the different groups (levels of the forest factor); see Fig. 5–10. Why is this ? What can be done about this ? Plot the residuals for each group for the model in that section. Then, fit a normal model with heterogeneous variances for each level of facFor. Alternatively, specify a linear model with an effect of the continuous covariate forest in the variance to see whether in this way the heterorogeneous variance can be accommodated.
3. Fit a Poisson GLM to one of the count vectors (e.g., from the first survey; section 5.6) and check whether the residual plots look better than those for the maximum count.
4. In Section 5.9, the residuals in the Poisson regression of the maximum count have too many large values. Try to add an "extra-residual", assumed to be normally distributed, and convert the Poisson GLM into a Poisson-log-normal GLM, to soak up that variability, and inspect whether you get new residuals that are more nearly symmetrical around 0. Check how the posterior standard deviations of the coefficient estimates change and whether the Bayesian p-value now indicates a fitting model.
5. In the Poisson GLM (section 5.9), fit a quadratic and a cubic term for elevation as well. Are these additional polynomial terms "significant" ?
6. In section 5.11, fit a binomial GLM to the detection frequency of the great tit, i.e., to the number of surveys (out of three) that a great tit was detected at a site.
7. In 5.13, fit a normal-normal mixed model to the logarithm of the replicated counts.
8. In 5.13, turn the Poisson GLMM into a Poisson GLM with fixed site effects.
9. Fit a binomial GLMM in section 5.14 to the detection frequency data (as in exercise 6).
10. In the Bernoulli GLMM in section 5.14, model the site-specific effect of wind speed by some site-specific covariates, such as elevation. That is, fit a model with the last line as this

$\alpha_{4,i} \sim Normal(\mu_{\alpha 4} + \beta_{\alpha 4} * elev_i, \sigma^2_{\alpha 4})$. What proportion of the variance of the slope of wind speed is explained by elevation ?

11. In section 5.14, check whether increasing the number of replicates per site (for instance, to 10 or 20) improves the agreement between the Bayesian and the frequentist estimates.

12. To find out which estimates under the Bernoulli GLMM are more trustworthy with small samples (as those in chapter 5.14), you can analyse simulated occupancy data where you have a known truth to compare with (see exercise 4–11 and section 10–5).