CHAPTER

3

Introduction to the Generalized Linear Model: The Simplest Model for Count Data

OUTLINE

Bayesian Population Analysis using WinBUGS
DOI: 10.1016/B978-0-12-387020-9.00003-1

3.1 INTRODUCTION

The generalized linear model (GLM) extends the concept of a linear model from normal response models, such as analysis of variance and regression, to many other response distributions, including the Poisson and the binomial. The GLM is a crucial piece in our lego collection of modeling parts, and perhaps *the* crucial piece. Described synthetically in 1972 by Nelder and Wedderburn (also see McCullagh and Nelder, 1989; Dobson and Barnett, 2008), the GLM unifies a huge number of superficially different analytical techniques, such as regression, analysis of variance, log-linear models, and logistic regression, among many others. We believe that a solid practical understanding of the GLM is essential for the work of every serious ecologist. To understand the GLM, you must know how to build a design matrix, what a link function is and how to choose a statistical distribution for an observed response. WinBUGS, with its lucid and elegant BUGS language, is perhaps the best software available to make one really understand the GLM. Useful introductions to the GLM with WinBUGS include Gelman and Hill (2007), Ntzoufras (2009), and Kéry (2010). In this chapter, we introduce two common GLMs for count data: Poisson and binomial GLMs. The essential difference between the two is that in the Poisson GLM, counts are unbounded in principle, whereas in a binomial GLM, counts are bounded by the so-called binomial totals. The GLM is the quintessential statistical model, so we first review the concept of a statistical model as a response being composed of signal plus noise.

3.2 STATISTICAL MODELS: RESPONSE = SIGNAL + NOISE

The basic form of a statistical model is such that we imagine a response as being composed of two components: signal and noise. The main difference between a statistical model and a purely mathematical one is that the statistical model contains a description of the random variability in an observed response, the noise. Alternative names of the signal–noise distinction are random and fixed part of a model, stochastic and systematic part, and mean and variance, or dispersion, structure of a model.

3.2.1 The Noise Component

The defining feature of a statistical model is that it accommodates the randomness and unpredictability that is the hallmark of all empirical data, even those in physics, chemistry, or molecular biology labs. To describe this noise, we use statistical distributions. Our quantitative descriptors of the random part of a model, the noise component, are the parameters

of these distributions and of course their identity. Some of the most frequently used distributions in population ecology are Poisson, binomial, normal, multinomial, and exponential. For an ecological modeler, it is important to get a feel for which statistical distribution is suitable as a description of the randomness in the data for a particular sampling situation, feature of the data collection protocol or measured trait. Experienced modelers therefore have a bestiary of distributions (Bolker, 2008) at their disposal, and they may well try out more than one for the same response to see which is most appropriate. Here, we do not describe these distributions but rather point to other books where some of the key distributions are described in more detail. Examples include Bolker (2008), Royle and Dorazio (2008), Kéry (2010), or Link and Barker (2010).

Program R has a large catalog of distributions that one can use and study to see how each one looks. Check out `?ddist`, and replace `dist` by any of the following: `pois`, `binom`, `norm`, `multinom`, `exp`, and `unif`. Changing the first letter of the function name from `d` to `p`, `q`, or `r` allows one to get the density, the distribution function, the percentiles, and random numbers from these distributions. For instance, to see how a beta distribution with parameters 2 and 4 looks like, you could execute either of the following R commands. Both generate a random sample of size n from the specified beta distribution and produce a plot. Remember that to know how an R function works you can type a question mark and its name (e.g., `?density`).

```
plot(density(rbeta(n = 10^6, shape1 = 2, shape2 = 4)))
hist(rbeta(10^6, 2, 4), nclass = 100, col = "gray")
```

The GLM in a strict sense has only a single noise component. Often, however, we need several noise components in a statistical model. Such models are introduced in Chapter 4. They include random- or mixed-effects models, hierarchical models, multilevel models, state-space models, or latent-component models. In a sense, they are all fairly similar, although there is a tremendous variety of them. In this book, we usually speak of random-effects or hierarchical models when there is more than a single noise component, except where there is a strong historic precedent favoring one term over the other, such as the state-space models in Chapter 5. In hierarchical models, the definition of a statistical model as being composed of a systematic and a random part must be made separately for each level of the model and what is the random part at one level becomes a component of the systematic part in the next level of the hierarchy.

3.2.2 The Signal Component

The signal component of the model contains the predictable parts of a response or the mean structure of a model. One of the most widely used descriptions of the mean structure is by a linear model, although nonlinear models can be adopted as well (Seber and Wild, 2003). The linear model is

one particular way to describe how we imagine that explanatory variables, such as indicators for group memberships or measured covariates, affect an observed response. A linear model is linear in the parameters and does not need to represent a straight line when plotted; most often it does not. This means that the parameters affect the mean response in an additive way, such as α and β in $y = \alpha^* x_1 + \beta^* x_2$, but not as in $y = (x_1^{\alpha})/(\beta + x_2)$, where x_1 and x_2 are variables. Most models that ecologists use in their daily work can be represented as linear models: for example, the *t*-test, simple and multiple linear regressions, ANOVA, ANCOVA, and mixed models.

All these models can be described in several ways, for example, with a plot, in words or in maths. To be able to code a model in the BUGS language, we need to know how to write it mathematically. Therefore, we need to learn how to describe each model by way of matrices and vectors and, in particular, how to build the so-called design matrix of a model. We briefly illustrate these topics with an ANCOVA linear model for a toy data set of nine data points. We assume that `y` is a response, `A` is a factor with three levels (i.e., a categorical covariate), and `X` is a continuous covariate. Here, they are ready to be defined in R:

```
# Define and plot data
y <- c(25, 14, 68, 79, 64, 139, 49, 119, 111)
A <- factor(c(1, 1, 1, 2, 2, 2, 3, 3, 3))
X <- c(1, 14, 22, 2, 9, 20, 2, 13, 22)
plot(X, y, col = c(rep("red", 3), rep("blue", 3), rep("green", 3)),
   xlim = c(-1, 25), ylim = c(0, 140))
```

In R, we can fit an ANCOVA with parallel slopes by issuing the following command:

```
summary(fm <- lm(y ~ A-1 + X))
[...]
Coefficients:
   Estimate Std. Error t value Pr(>|t|)
A1    1.315    19.260    0.068    0.9482
A2   65.218    17.965    3.630    0.0151 *
A3   58.648    19.260    3.045    0.0286 *
X     2.785     1.031    2.701    0.0427 *
[...]
Residual standard error: 25.05 on 5 degrees of freedom
Multiple R-squared: 0.951,     Adjusted R-squared: 0.9117
F-statistic: 24.24 on 4 and 5 DF, p-value: 0.001798
```

This formula language for defining the linear model underlying the ANCOVA or any other linear model is ingenious because it is quick and error-free *if you know how to specify your model*. The downside is that you must know what this statement actually causes R to do. So, let us look under the hood and see what R does when we tell it `y ~ A-1 + X`. When fitting this model, we are in effect fitting the following linear models:

$$y_i = \alpha_{j(i)} + \beta^* X_i + \varepsilon_i \quad \text{and} \quad \varepsilon_i \sim \text{Normal}(0, \sigma^2)$$

Here, y_i is the response of unit (data point, individual, row) i, and X_i is the value of the continuous explanatory variable `X` for unit i. Factor `A` codes for the group membership of each unit (for the meaning of the −1, see below). We have three groups; therefore, the index j is 1, 2, or 3, and we may write $j = A_i$. This means that units with a value $A_i = 1$ have $j = 1$ and α_1 and so forth. There are two parameters in the model, which describe the mean structure of the model, α and β, of which the first is a vector and the second is a scalar. The vector α has three elements, corresponding to the effects of the three levels of factor `A`. The part $\alpha_{j(i)} + \beta^* X_i$ represents the expected response for unit i, that is, the value expected to be observed in the absence of any noise in the system. Thus, this is the signal part of the response. The noise part consists of that part of the response, which we cannot explain by our linear combination of the explanatory variables: it is represented by the residuals ε_i. Since we know a little bit about these noise terms, we claim that they come from a normal distribution with mean equal to zero and common variance σ^2. In all, the model has five parameters: α_1, α_2, α_3, β, and σ^2. A little confusingly, we may also say that the model has three parameters: the vector α and the scalars β and σ^2.

Other algebraic ways of writing this model clarify perhaps even more the structure of a statistical model as being made up of a signal or systematic part and a noise or random part. The following shows clearly that the response is normally distributed around the values of the linear predictor, $\alpha_{j(i)} + \beta^* X_i$. Now, residuals ε_i are defined only implicitly.

$$y_i \sim \text{Normal}(\alpha_{j(i)} + \beta^* X_i, \sigma^2).$$

A further possibility is to express μ_i as the expected response of unit i in the absence of any random noise. It is the same as the value of the linear predictor:

$$y_i \sim \text{Normal}(\mu_i, \sigma^2), \text{ with } \mu_i = \alpha_{j(i)} + \beta^* X_i$$

Being able to write a linear model in algebra greatly simplifies the coding of a model in WinBUGS. Most models, when written in the BUGS language, in fact very much resemble their algebraic description.

Yet another way to write this model for our data set is by way of matrices and vectors:

$$\begin{pmatrix} 25 \\ 14 \\ 68 \\ 79 \\ 64 \\ 139 \\ 49 \\ 119 \\ 111 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 14 \\ 1 & 0 & 0 & 22 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 0 & 9 \\ 0 & 1 & 0 & 20 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 1 & 13 \\ 0 & 0 & 1 & 22 \end{pmatrix} \times \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \beta \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \varepsilon_4 \\ \varepsilon_5 \\ \varepsilon_6 \\ \varepsilon_7 \\ \varepsilon_8 \\ \varepsilon_9 \end{pmatrix}, \text{ with } \varepsilon_i \sim \text{Normal}(0, \sigma^2)$$

From left to right, we have response vector, design matrix, parameter vector, and residual vector. The design matrix (also called model matrix or X matrix) multiplied with the parameter vector produces another vector, which is the expected value of the response for each unit. This expected value is also called the linear predictor and is the same as μ_i above. For example, the value of the linear predictor for the first data point is given by $1*\alpha_1 + 0*\alpha_2 + 0*\alpha_3 + 1*\beta$.

It is useful to practice so that you can jump back and forth between different ways of describing the same model: the R formula language, in algebra, and using matrices and vectors. We need this clear understanding of the linear model if we want to succeed in our modeling with WinBUGS. This may be awkward at first, but a tremendous benefit is that this finally forces on us what we may not have achieved before: a clear understanding of linear models.

The interpretation of the elements of the vector of the mean parameters (α_1, α_2, α_3, β) depends on the way the design matrix is structured, and there are different ways for doing this. These variations are called parameterizations of a model; they are equivalent ways of writing what is effectively the same model. Sometimes one parameterization may be more convenient and sometimes another. WinBUGS can be very sensitive to the choice of parameterization of a model; sometimes one will not work at all, for example, the chains would not mix, but another will result in beautiful mixing. Hence, it is important that we know how to specify different parameterizations of the same model.

One clever trick when trying to understand the design matrix of a model is the use of the R function `model.matrix()`, especially for those of you who have some experience in fitting linear models with a model-definition language such as that in R. We have had cases in which we did not quite understand the structure of a complicated linear model that we wanted to fit in WinBUGS. To understand the design matrix of the model and, therefore, to be able to specify the model in WinBUGS, we went into R and used `model.matrix()` to find out how the design matrix of the model we wanted to specify in WinBUGS looked like.

As an example, two possible parameterizations of our ANCOVA model are those which we specify by the following R commands:

```
# Effects or treatment contrast parameterization
model.matrix(~ A + X)
  (Intercept) A2 A3  X
1           1  0  0  1
2           1  0  0 14
3           1  0  0 22
4           1  1  0  2
5           1  1  0  9
6           1  1  0 20
7           1  0  1  2
```

```
8            1  0  1  13
9            1  0  1  22
[ ... ]

# Means parameterization
model.matrix(~ A-1 + X)
  A1 A2 A3   X
1  1  0  0   1
2  1  0  0  14
3  1  0  0  22
4  0  1  0   2
5  0  1  0   9
6  0  1  0  20
7  0  0  1   2
8  0  0  1  13
9  0  0  1  22
[ ... ]
```

The former is the R default called the effects or treatment contrast parameterization. It specifies the linear model in terms of a baseline response, which here is that for the first level of factor `A`, plus effects of the other levels of `A` relative to the first level and effects of each unit change in the covariate `X`. Hence, the intercept is the mean response of units with level 1 of factor `A` at $X = 0$, and the parameters corresponding to the design matrix columns A_2 and A_3 quantify the *difference* in the mean response in levels 2 and 3, relative to that of level 1, for a given value of X. The parameter representing column X in the design matrix is the common slope of the regression of y on X, regardless of which group a unit belongs to. In contrast, in the means parameterization, the first three parameters directly represent the mean response for each level of factor A at $X = 0$, while the meaning of the parameter represented by the column X is the same as before.

The overly simple setting chosen here allows one to see the main features of these topics very clearly, and in real-world modeling, where one typically has many explanatory variables, one must strive for an understanding of the linear model by breaking down the problem into its smallest understandable parts. In reality, there may be main effects and interaction effects and perhaps aliasing between columns of the design matrix, meaning some parameters cannot be estimated independently. This complicates the construction of the design matrix even more (although most problems are really ones of bookkeeping). We will not go further into these topics, but refer you to any of a huge number of books that explain the linear model, such as Kéry (2010) or chapter 6 in E. Cooch and G. White's free *Gentle Introduction to Program MARK* (www.phidot.org/software/mark/docs/book).

This concludes our very brief description of how the signal component of a linear statistical model is built using the design matrix, which, when

matrix multiplied with the parameter vector, yields the value of the linear predictor or the expected or "typical" response, μ_i, which is also what you get in R by typing `model.matrix(~A+X) %*% fm$coef`.

3.2.3 Bringing the Noise and the Signal Components Together: The Link Function

In the linear models mentioned so far in this chapter, we assumed a normal distribution for the random part of the response. Thus, we directly wrote the response y_i as a simple additive combination of the signal μ_i and the noise ε_i. For responses that cannot be modeled with a normal distribution, this direct combination is no longer possible. For instance, directly adding noise from a Poisson or a binomial distribution to the value of a linear predictor would typically result in inadmissible values for the response, for example, fractional or negative counts.

The big advantage of generalized linear models (GLMs) is that we can apply a linear model to the response indirectly: namely, to a transformation of the mean response. The function that transforms the expected response is called the *link function*. The reason for that name should now be clear: the link function allows us to link the noise component and the signal components in a model. In this way, the useful concepts of linear models can be carried over to a vastly larger class of models.

The classical way to describe a GLM is by three components: a random part (noise), a systematic part (signal), and a link function. More generally, for response y_i, we can write the following:

1. Random part of the response (the noise)—a statistical distribution f with mean response μ_i:

$$y_i \sim f(\mu_i)$$

2. A link function g, which is applied to the mean response μ_i:

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

3. Systematic part of the response (the signal)—a linear predictor (η_i), for example, for a simple linear regression–type of GLM:

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

We can describe a GLM succinctly in only two lines:

$$y_i \sim f(\mu_i)$$

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

This is exactly the way in which GLMs are specified in the BUGS language, and this is the reason why BUGS is so great if you want to really

understand GLMs. The GLM concept gives you considerable creative freedom to combine the three components of a GLM, but there are typically pairs of response distributions and link functions that go together particularly well. These link functions are called canonical link functions and 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 responses ($\eta_i = \log(\mu_i)/\log(1 - \mu_i)$). Together, these three standard GLMs make up a vast number of statistical methods used in population ecology and elsewhere; for an overview, see Kéry (2010).

The broad scope of the GLM is one reason for the great importance of the GLM for you. The other one, which we will see many times later in the book, is that the GLM represents the main building block for many more complicated models, especially hierarchical models. Many of the most exciting ecological models for inference about populations can be viewed simply as a sequence of coupled GLMs (Royle and Dorazio, 2008).

3.3 POISSON GLM IN R AND WinBUGS FOR MODELING TIME SERIES OF COUNTS

The Poisson distribution is defined for positive integers at 0, 1, 2, … and hence is a suitable model for counts under the assumption of independence and spatial or other randomness. It describes the "residual variation" (noise), after any kinds of systematic effects (signal) in the Poisson mean have been taken account of, for example, in the form of covariate effects.

The Poisson distribution has a single parameter, the intensity or rate parameter λ representing the expected count. The variance of a Poisson random variable is not a free parameter, rather it is identical to the mean (expected) count. In practice, this strong assumption about the mean–variance relationship is frequently violated and the variance of counts is larger than their mean. There are various ways to take account of this. The conceptually easiest is perhaps the introduction of data-level random effects to take account of such overdispersion (Section 4.2).

For an example of a Poisson GLM, let us assume we model counts C_i from a number of years i. Here is the algebraic description of the Poisson GLM for count C_i in year i with a single covariate X:

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

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

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

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

3. Systematic part of the response (linear predictor η_i):

$$\eta_i = \alpha + \beta^* X_i$$

Here, λ_i is the expected count (the mean response) in year i on the arithmetic scale, η_i is the expected count in year i on the link scale (i.e., the linear predictor), X_i is the value of covariate X in year i, and α and β are the two parameters of the log-linear regression of these counts on the covariate. We will next look at the generation and analysis of Poisson GLM data for a simulated and also for a real data set.

3.3.1 Generation and Analysis of Simulated Data

We define a function that generates Poisson counts of peregrine falcons (Fig. 3.1) for one population over n years. The parameter values are inspired by actual data from the French Jura mountains (Monneret, 2006), see Section 3.3.2. In this example, the linear predictor will be a cubic polynomial function of time, $\eta_i = \alpha + \beta_1{}^* X_i + \beta_2{}^* X_i^2 + \beta_3{}^* X_i^3$. In this section, we will, first, show how a GLM is analyzed in the frequentist framework in R and in the Bayesian framework using WinBUGS and illustrate how similar numerically the resulting estimates are. Second, as this is the first time we run WinBUGS, we explain each step of the analysis in more detail than later in this book.

FIGURE 3.1 Peregrine falcon (*Falco peregrinus*), Switzerland (Photograph by B. Renevey).

```
data.fn <- function(n = 40, alpha = 3.5576, beta1 = -0.0912,
    beta2 = 0.0091, beta3 = -0.00014){
    # n: Number of years
    # alpha, beta1, beta2, beta3: coefficients of a
    #    cubic polynomial of count on year

    # Generate values of time covariate
    year <- 1:n

    # Signal: Build up systematic part of the GLM
    log.expected.count <- alpha + beta1 * year + beta2 * year^2 + beta3 *
       year^3
    expected.count <- exp(log.expected.count)

    # Noise: generate random part of the GLM: Poisson noise around
    expected counts
    C <- rpois(n = n, lambda = expected.count)

    # Plot simulated data
    plot(year, C, type = "b", lwd = 2, col = "black", main = "", las = 1,
       ylab = "Population size", xlab = "Year", cex.lab = 1.2,
       cex.axis = 1.2)
    lines(year, expected.count, type = "l", lwd = 3, col = "red")

    return(list(n = n, alpha = alpha, beta1 = beta1, beta2 = beta2,
       beta3 = beta3, year = year, expected.count = expected.count, C = C))
    }
```

We obtain one realization of the stochastic process, that is, population counts over 40 years, and plot the population trajectory over time (Fig. 3.2a)

```
data <- data.fn()
```

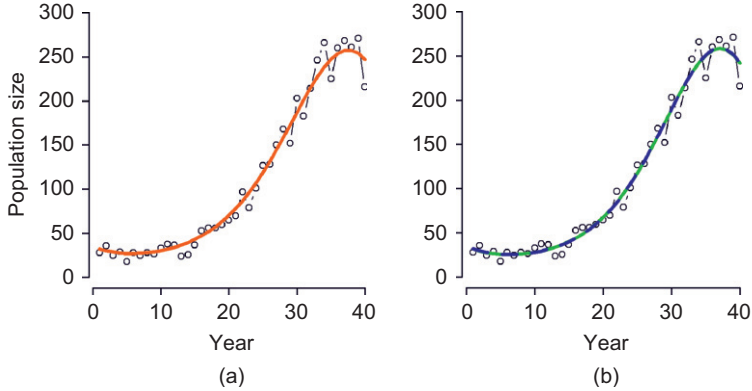

FIGURE 3.2 Simulated population size of peregrines in the French Jura over 40 years. (a) Expected population size (red) and the observed data, the realized population size (black). (b) Observed data (black) and estimated population trajectories from a frequentist (green) and a Bayesian analysis (blue) of a Poisson regression with cubic polynomial effects of year. The R code to produce this figure slightly differs from the one shown in this book.

Next, we analyze this data set in R and in WinBUGS. Fitting a GLM in the frequentist mode of analysis, using the method of maximum likelihood, is trivially easy in statistical software like R that have canned functions such as `glm()`. Here is the analysis using R; one or two lines of code suffice. Up to Poisson sampling error, this will recover parameter estimates that resemble the values of the input.

```
fm <- glm(C ~ year + I(year^2) + I(year^3), family = poisson, data = data)
summary(fm)

Call:
glm(formula = C ~ year + I(year^2) + I(year^3), family = poisson,
    data = data)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.9036  -0.5815   0.2250   0.8888   1.4972

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)  3.570e+00  1.159e-01  30.797  < 2e-16 ***
year        -1.099e-01  2.023e-02  -5.431 5.61e-08 ***
I(year^2)    1.026e-02  1.005e-03  10.204  < 2e-16 ***
I(year^3)   -1.578e-04  1.467e-05 -10.756  < 2e-16 ***
—
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 2771.076  on 39  degrees of freedom
Residual deviance:   42.729  on 36  degrees of freedom
AIC: 297.77

Number of Fisher Scoring iterations: 4
```

We note in passing that in order to arrive at this solution, R had to turn the handle on a blackbox called "Fisher Scoring" four times.

In contrast, the code for fitting the same model in WinBUGS is less succinct: about 20 lines compared with 1–2 for the same analysis in R. And this is only for the description of the model in the BUGS language; more lines of code are required to actually conduct the analysis of that model with WinBUGS from R.

```
# Specify model in BUGS language
sink("GLM_Poisson.txt")
cat("
model {

# Priors
alpha ~ dunif(-20, 20)
beta1 ~ dunif(-10, 10)
```

```
beta2 ~ dunif(-10, 10)
beta3 ~ dunif(-10, 10)

# Likelihood: Note key components of a GLM on one line each
for (i in 1:n){
    C[i] ~ dpois(lambda[i])           # 1. Distribution for random part
    log(lambda[i]) <- log.lambda[i]   # 2. Link function
    log.lambda[i] <- alpha + beta1 * year[i] + beta2 * pow(year[i],2) +
       beta3 * pow(year[i],3)         # 3. Linear predictor
    } # i
}
",fill = TRUE)
sink()
```

In this book, we present each WinBUGS analysis in a common layout, by first writing a text file with the model definition in the BUGS language, followed by all the other ingredients that the function `bugs()` in the R2WinBUGS package (Sturtz et al., 2005) requires to instruct WinBUGS from R. Remember that before executing the following code, you must define an R object called `bugs.dir` that contains the address of WinBUGS. For a Swiss-German locale, this might be

```
bugs.dir <- "c:/Programme/WinBUGS14/"
```

Next, we bundle into an R list the data needed for the analysis by WinBUGS.

```
# Bundle data
win.data <- list(C = data$C, n = length(data$C), year = data$year)
```

The next step is to define initial values for the estimated quantities. WinBUGS can generate initial values by drawing them from their priors, so it is not necessary to give inits for all estimands. Nevertheless, when running WinBUGS by calling it from R, we need to define inits for at least one quantity. For complex models, it is often vital to choose good initial values because otherwise WinBUGS may crash. It is useful to define a function to define inits. This function is then executed once for each Markov chain run in the analysis.

```
# Initial values
inits <- function() list(alpha = runif(1, -2, 2), beta1 = runif(1, -3, 3))
```

Next, we write a list with the quantities we want to estimate ("monitor" as WinBUGS calls it), that is, for which we want WinBUGS to save the draws from the joint posterior distribution. This includes derived quantities such as `lambda` here.

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

Before running the analysis, we set the MCMC characteristics: the number of draws per chain, thinning rate, burnin length, and the number of chains. The burnin should be long enough to discard the initial part of the Markov chains that have not yet converged to the stationary distribution. We typically determine the required burnin in initial trials. Thinning is useful to save computer disk space. We run multiple chains to check for convergence.

```
# MCMC settings
ni <- 2000
nt <- 2
nb <- 1000
nc <- 3
```

Now, we have defined all R objects that we need as arguments for our call to R function `bugs()`. When calling WinBUGS from R, we usually set the argument `debug = TRUE`. WinBUGS then remains open after the requested number of iterations has been produced, and we can visually inspect whether the chains have converged, or in the case of an error, directly read the log file.

```
# Call WinBUGS from R
out <- bugs(data = win.data, inits = inits, parameters.to.save = params,
   model.file = "GLM_Poisson.txt", n.chains = nc, n.thin = nt,
   n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory = bugs.dir,
   working.directory = getwd())
```

Most shockingly, WinBUGS almost immediately crashes, claiming that there was an "undefined real result" trap. Why on Earth should you continue with this book (or even buy the book!) when GLMs can be fitted so very much more easily using functions in software such as R? There are at least three reasons for why you should continue:

1. You can use the same kind of analysis for *much* more complex models, models that you will hardly or not at all be able to fit with R. So, understanding how to fit the simpler models in WinBUGS rather than in R is an essential preparation for that.
2. You conduct a Bayesian analysis instead of a frequentist one. The Bayesian view of statistics has several advantages as argued by many (e.g., McCarthy, 2007; Link and Barker, 2010; also see Chapter 2).
3. The model that you fit is more transparent when using the BUGS language to describe it than when describing it in R. So, there is a huge heuristic benefit of statistical modeling in WinBUGS: that of actually understanding the model you are fitting.

But back to our Bayesian analysis of that really simple model, our Poisson GLM: why was WinBUGS not able to get a solution in this simple case? The answer is that we need to ensure that covariate values are not too far away from zero, that is, that they have neither too large negative nor too large positive values. Note that year^3 goes up to $40^3 = 64{,}000$, and this causes numerical overflow (covariate values in the 10s or even in the 100s should not be a problem). To avoid this problem, we usually center or standardize our covariates. We could use the usual standardization by subtracting the mean and dividing by the standard deviation; this results in transformed covariates with approximately zero mean and unit standard deviation. However, any other transformation usually also works, provided that the range of the transformed covariate does not extend to far on either side of 0. Here, we subtract 20 and divide 10. This is easy and the only cost is when we want to present the results and when we want to compare the input values with our parameter estimates. We will graph the results of the model fitted in WinBUGS with the standardized year covariate. This will convince you that we have actually fitted the equivalent model. So next we repeat the analysis using the standardized covariate values. We can do this simply by adapting the statement in which we package the data and then recycle the rest of the code.

```
# Bundle data
mean.year <- mean(data$year)          # Mean of year covariate
sd.year <- sd(data$year)              # SD of year covariate
win.data <- list(C = data$C, n = length(data$C),
   year = (data$year - mean.year) / sd.year)

# Call WinBUGS from R (BRT < 1 min)
out <- bugs(data = win.data, inits = inits, parameters.to.save = params,
   model.file = "GLM_Poisson.txt", n.chains = nc, n.thin = nt,
   n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory = bugs.dir,
   working.directory = getwd())
```

This works smoothly and produces a nice first bit of output from our first Bayesian analysis; how reassuring! To return the data into R, you have to manually exit WinBUGS and can then obtain a numerical summary of the Bayesian analysis.

```
# Summarize posteriors
print(out, dig = 3)
Inference for Bugs model at "GLM_Poisson.txt", fit using WinBUGS,
3 chains, each with 2000 iterations (first 1000 discarded), n.thin = 2
n.sims = 1500 iterations saved
             mean    sd    2.5%     25%     50%     75%   97.5%  Rhat n.eff
alpha       4.268 0.030   4.208   4.248   4.268   4.288   4.324 1.005   410
beta1       1.308 0.042   1.230   1.277   1.307   1.338   1.391 1.003   590
beta2       0.076 0.024   0.031   0.060   0.076   0.093   0.122 1.013   480
beta3      -0.253 0.022  -0.297  -0.268  -0.253  -0.237  -0.212 1.006   340
```

```
lambda[1]   32.367 3.243  26.579  30.000  32.340  34.492  38.965 1.007   340
lambda[2]   29.815 2.562  25.184  27.997  29.850  31.490  35.057 1.006   390
lambda[3]   27.993 2.068  24.229  26.550  27.995  29.402  32.111 1.005   480
    [ ... ]
lambda[38] 257.081 6.451 245.100 252.600 256.900 261.500 269.800 1.001  1500
lambda[39] 251.555 7.723 237.442 246.000 251.300 256.700 267.252 1.000  1500
lambda[40] 242.160 9.271 225.147 235.700 241.900 248.325 261.400 1.000  1500
deviance   293.695 2.622 290.300 291.800 293.200 295.000 300.700 1.001  1500

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 = 3.9 and DIC = 297.6
DIC is an estimate of expected predictive error (lower deviance is better).
```

We note that there is a plot function for R objects of the bugs class. Typing `plot(out)` will produce a useful graphical summary of the Bayesian analysis, but we do not show this plot here.

Now, before we even inspect the parameter estimates, we should make sure that their Markov chains have converged. Only then are the random draws produced a valid sample from the desired target distribution, which is the posterior distribution of these parameters. Convergence can never be proven; what *looks* like convergence may indeed sometimes represent chains that have definitely not reached their stationary distribution (see Kéry, 2010, for some nice examples of this). However, in many cases, visual and numerical checks are adequate, and this is what we do in this book: we always inspect the time-series plots in the WinBUGS log file and the values of the Rhat statistics in the table of posterior summaries produced by the function `bugs()`. Rhat is a formal convergence test criterion comparing the among- and the within-chain variance in an ANOVA fashion; values around 1 suggest the absence of a "chain effect" and therefore convergence (Brooks and Gelman, 1998). Often, 1.1 or 1.2 is taken as an Rhat-value that indicates convergence (Gelman and Hill, 2007).

The eye is quite good at picking up a pattern in a graph, and this visual assessment of the likely convergence of chains will complement the numerical assessment by the Rhat values (Fig. 3.3). With experience, you will get a trained eye because you will have seen so many plots of chains that have converged (based on Rhat) and so many others that have not. With `debug = TRUE` as an argument of the function `bugs()`, WinBUGS stays open after execution of the desired number of iterations, and the full WinBUGS functionality can be used for further analyses. For instance, additional iterations could be asked for and numerical summaries produced. However, one may also simply skim over the time-series plots of all monitored parameters to see whether any of them looks like they have not converged (yet). At convergence, the chains should oscillate randomly around a horizontal level and (when three parallel chains are run) should look "grassy" (Link and Barker, 2010). There should not be a sustained trend.

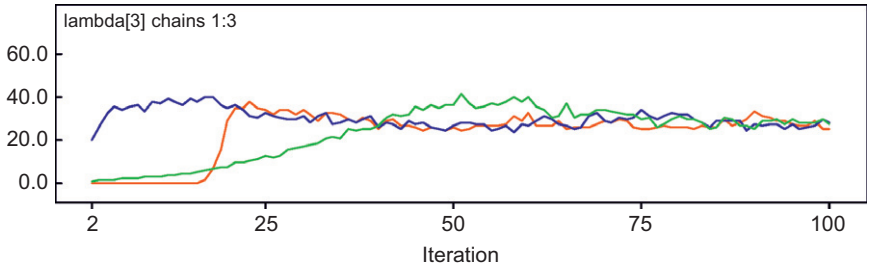

FIGURE 3.3 Example of a time-series plot with three chains for one parameter (the expected count in year 3). The chains have converged after around 75 iterations and show good mixing. Thus we would repeat the analysis and might set the burnin to at least 75, but probably even to several 100 to be on the safe side.

Furthermore, the lines representing the different chains (usually, 2 or 3) should be strongly interspersed: if this is the case, one says that the chains mix well. In our simple model, we can inspect the time-series plots and see that after the burnin of length 200, the chains of all parameters have indeed converged. We arrive at the same conclusion based on the Rhat values.

To see what initial nonconvergence looks like, you may repeat the analysis with the below MCMC settings. Looking at the plots in WinBUGS shows that convergence is generally achieved after some 75 iterations (Fig. 3.3).

```
# New MCMC settings with essentially no burnin
ni <- 100
nt <- 1
nb <- 1

# Call WinBUGS from R (BRT <1 min)
tmp <- bugs(data = win.data, inits = inits, parameters.to.save = params,
   model.file = "GLM_Poisson.txt", n.chains = nc, n.thin = nt,
   n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory = bugs.dir,
   working.directory = getwd())
```

Once we are satisfied with the convergence of the chains, we can see what we learn from the analysis of the model. For instance, we can plot the Poisson means (the `lambda` parameters) for each year; these represent the expected peregrine counts in each year. We plot them in the same figure as the predicted values from the analysis of the same model using the R function `glm()`, and we will see that our inference is virtually identical (Fig. 3.2b).

```
plot(1:40, data$C, type = "b", lwd = 2, col = "black", main = "", las = 1,
   ylab = "Population size", xlab = "Year")
R.predictions <- predict(glm(C ~ year + I(year^2) + I(year^3),
   family = poisson, data = data), type = "response")
lines(1:40, R.predictions, type = "l", lwd = 3, col = "green")
WinBUGS.predictions <- out$mean$lambda
lines(1:40, WinBUGS.predictions, type = "l", lwd = 3, col = "blue", lty = 2)
```

We see that the predicted population sizes are so similar from R and WinBUGS that we can hardly see both lines. This is a very common observation: typically, inference from a frequentist and a Bayesian analysis of a statistical model is numerically almost identical when noninformative priors are used. Let us plot the two predictions side by side to convince us that they are indeed so similar:

```
cbind(R.predictions, WinBUGS.predictions)
  R.predictions WinBUGS.predictions
1      32.14482          32.36720
2      29.66780          29.81523
3      27.89637          27.99280
  [ .... ]
38    256.76388         257.08067
39    251.21847         251.55480
40    241.79082         242.16020
```

R was able to recover parameter estimates that were very similar to those that we input when assembling the data set. Now we see that the predicted population trajectory from the frequentist analysis in R, using untransformed covariate values, and from the Bayesian analysis in WinBUGS, using transformed covariate values, are virtually identical.

There is always a trade-off between simplicity of model fitting and flexibility: R functions such as `glm()` are very handy, but they can only fit a relatively limited array of models. In addition, and perhaps more importantly, the model fitted with a function like `glm()` is much less transparent than when the same model is fit in WinBUGS. We will see this in the random-effects extension to GLMs in Chapter 4; in WinBUGS, it is conceptually trivial to go from the pure Poisson GLM to the Poisson-lognormal mixed-effects GLM or Poisson generalized linear mixed model (GLMM). In contrast, in R we would have to use a different setting of the same function (`glm(,family=quasipoisson)`) or use an altogether different function, such as `lmer()`.

3.3.2 Analysis of Real Data Set

We will repeat this analysis now using actual data by analyzing the trajectory of the peregrine population breeding in the French Jura from 1964 to 2003 (R.-J.Monneret, personal communication). Note that by merely conducting this analysis, we implicitly make the assumption that the survey coverage and detection probability of peregrine pairs in the French Jura have not changed in a sustained way over time, otherwise our perceived trends will be distorted (see, e.g., Nichols et al., 2009; Kéry, 2010; Chapter 5). If we have doubts about this important assumption, then a survey design should be used that allows detection probability to be estimated and therefore corrected for. The metapopulation estimation

methods in Chapters 12 and 13 illustrate ways this could be done in the context of population studies of raptors such as the peregrine. Another feature that our model does not address either is possible temporal autocorrelation. To add this, we could use models developed in Chapter 5.

```
# Read data
peregrine <- read.table("falcons.txt", header = TRUE)
```

We attach the data set to be able to directly write the variable names when addressing them.

```
attach(peregrine)
```

The data set contains the counts of adult pairs (Pairs), reproductive pairs (R.Pairs), and fledged young (Eyasses) for each of 40 years. We plot variable Pairs (Fig. 3.4a).

```
plot(Year, Pairs, type = "b", lwd = 2, main = "", las = 1, ylab = "Pair
   count", xlab = "Year", ylim = c(0, 200), pch = 16)
```

We fit the model in WinBUGS and plot the predictions, vector `lambda`, into the same plot (Fig. 3.4a, blue line). We will use the same R/WinBUGS code as before.

```
# Bundle data
mean.year <- mean(1:length(Year))          # Mean of year covariate
sd.year <- sd(1:length(Year))              # SD of year covariate
win.data <- list(C = Pairs, n = length(Pairs), year = (1: length(Year) -
   mean.year) / sd.year)

# Initial values
inits <- function() list(alpha = runif(1, -2, 2), beta1 = runif(1, -3, 3))
```

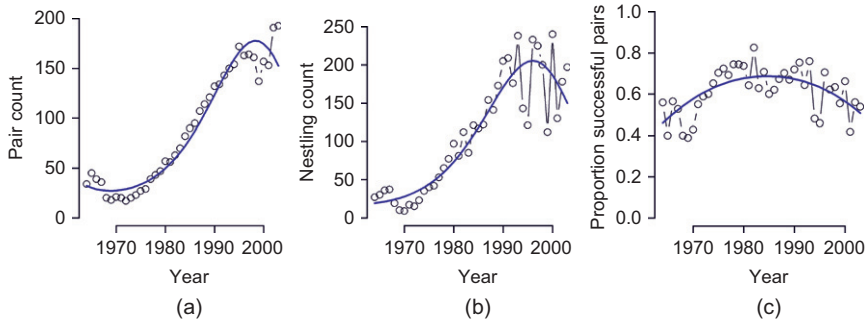

FIGURE 3.4 Analysis of (a) population size (number of territorial pairs), (b) fecundity (total number of fledged young), and (c) proportion of successful pairs in the peregrine population of the French Jura from 1964 to 2003 (data courtesy of R.-J. Monneret). Observed data are in black and Bayesian posterior means from WinBUGS in blue. The R code to produce this figure slightly differs from the one shown in this book.

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

# MCMC settings
ni <- 2500
nt <- 2
nb <- 500
nc <- 3

# Call WinBUGS from R (BRT < 1 min)
out1 <- bugs(data = win.data, inits = inits, parameters.to.save = params,
   model.file = "GLM_Poisson.txt", n.chains = nc, n.thin = nt,
   n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory = bugs.dir,
   working.directory = getwd())

# Summarize posteriors
print(out1, dig = 3)
              mean    sd    2.5%     25%     50%     75%   97.5%  Rhat n.eff
alpha        4.234 0.030   4.176   4.214   4.234   4.254   4.293 1.006   400
beta1        1.115 0.047   1.022   1.082   1.116   1.147   1.204 1.009   250
beta2        0.005 0.024  -0.040  -0.012   0.005   0.021   0.052 1.006   350
beta3       -0.233 0.025  -0.280  -0.250  -0.234  -0.215  -0.183 1.010   220
lambda[1]   32.258 3.231  26.499  29.880  32.160  34.450  38.980 1.005   450
lambda[2]   30.221 2.572  25.600  28.367  30.155  31.942  35.550 1.004   570
lambda[3]   28.787 2.088  25.020  27.310  28.710  30.150  33.060 1.003   830
[ ... ]
lambda[38] 169.689 5.294 159.700 166.100 169.700 173.300 179.900 1.001  2800
lambda[39] 162.276 6.248 150.397 158.000 162.200 166.500 174.700 1.002  1500
lambda[40] 152.733 7.347 138.797 147.700 152.600 157.600 167.202 1.003   920
deviance   322.348 2.785 318.900 320.300 321.700 323.600 329.400 1.004   590
[ ... ]
DIC info (using the rule, pD = Dbar-Dhat)
pD = 4.0 and DIC = 326.3
```

Convergence of all chains looks decent; the values of Rhat in the summary are close to 1, and the plots of the chains in WinBUGS look nice. Therefore, we now plot the predicted population trajectory into the previous plot (Fig. 3.4a).

```
WinBUGS.predictions <- out1$mean$lambda
lines(Year, WinBUGS.predictions, type = "l", lwd = 3, col = "blue", lty = 2)
```

3.4 POISSON GLM FOR MODELING FECUNDITY

The Poisson distribution is the standard model for any kind of unbounded count data. Counts could be alleles, individuals, family or other groups, or species. To make this very clear, we will swiftly conduct an analogous analysis for the counts of fledged young, that is, for

fecundity, in this same peregrine population (Monneret, 2006). We will use the same model and code and directly plot a Bayesian analysis of a cubic polynomial of the number of fledged young on year (Fig. 3.4b).

```
plot(Year, Eyasses, type = "b", lwd = 2, main = "", las = 1,
   ylab = "Nestling count", xlab = "Year", ylim = c(0, 260), pch = 16)

# Bundle data
mean.year <- mean(1:length(Year))     # Mean of year covariate
sd.year <- sd(1:length(Year))         # SD of year covariate
win.data <- list(C = Eyasses, n = length(Eyasses), year =
   (1: length(Year) - mean.year) / sd.year)

# Call WinBUGS from R (BRT < 1 min)
out2 <- bugs(data = win.data, inits = inits, parameters.to.save =
   params, model.file = "GLM_Poisson.txt", n.chains = nc, n.thin = nt,
   n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory = bugs.dir,
   working.directory = getwd())
```

Skimming over the plots of the Markov chains in WinBUGS and inspecting the values of Rhat in the summary (not shown) suggest that convergence has been reached. Therefore, we are satisfied to plot the estimates under the model (Fig. 3.4b).

```
# Plot predictions
WinBUGS.predictions <- out2$mean$lambda
lines(Year, WinBUGS.predictions, type = "l", lwd = 3, col = "blue")
```

3.5 BINOMIAL GLM FOR MODELING BOUNDED COUNTS OR PROPORTIONS

We saw that the Poisson distribution is the standard model for unbounded count data. However, frequently we have counts that are bounded by some upper limit. For instance, when modeling the number of females in a brood, counts of female nestlings cannot exceed the size of a brood. Similarly, when modeling the number of successful broods, counts cannot exceed the total number of broods monitored. As a special case, when modeling the number of survival events for an individual over a single time step, the count cannot exceed 1. The standard model for all these kinds of counts is the binomial distribution. It arises when N independent individuals all have the same probability p of experiencing some event (for instance, being female, successful or a survivor). The number of events counted (C) will follow a binomial distribution. A special case with $N = 1$ is called a Bernoulli distribution.

As our example for a binomial GLM, we will model the number of successful pairs (C_i) among all monitored pairs (N_i) in year i for a total

of 40 years. We will treat year as a continuous covariate and fit a quadratic polynomial. That model can be written like the following:

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

$$C_i \sim \text{Binomial}(N_i, p_i)$$

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

$$\text{logit}(p_i) = \log\left(\frac{p_i}{1-p_i}\right) = \eta_i$$

3. Systematic part of the response (linear predictor η_i):

$$\eta_i = \alpha + \beta_1{}^* X_i + \beta_2{}^* X_i^2$$

Here, p_i is the expected proportion of successful pairs on the arithmetic scale. It is the mean response for each of the N_i trials. η_i is that same proportion on the (logit) link scale. The primary parameter of the binomial distribution is p_i. It is often called the success probability because there are two events: the one of focal interest termed a success and the other a failure. In contrast, the binomial total, or trial or sample size N_i is not normally a parameter; rather, it is typically observed or a fixed element of the design.

The usual link function adopted for a binomial GLM is the logit. It maps the probability scale (i.e., the range from 0 to 1) onto the entire real line (i.e., from $-\infty$ to ∞) and ensures that a linear model does not result in probabilities outside of that admissible range, that is, below 0 or above 1. The rest of the model (the linear predictor η_i) is up to your data, your questions, and your imagination. Here, α, β_1, and β_2 are simply the three parameters of the logit-linear regression of the unobserved proportions on covariate *Year*. Next, we simulate binomial data for the proportion of successful peregrine pairs and analyze them with WinBUGS.

3.5.1 Generation and Analysis of Simulated Data

We write a function that simulates data from this simple setting that typically leads to the adoption of a binomial GLM.

```
data.fn <- function(nyears = 40, alpha = 0, beta1 = -0.1, beta2 = -0.9){
    # nyears: Number of years
    # alpha, beta1, beta2: coefficients

    # Generate untransformed and transformed values of time covariate
    year <- 1:nyears
    YR <- (year-round(nyears/2)) / (nyears / 2)
```

```
# Generate values of binomial totals (N)
N <- round(runif(nyears, min = 20, max = 100))

# Signal: build up systematic part of the GLM
exp.p <- plogis(alpha + beta1 * YR + beta2 * (YR^2))

# Noise: generate random part of the GLM: Binomial noise around
expected counts (which is N)
C <- rbinom(n = nyears, size = N, prob = exp.p)

# Plot simulated data
plot(year, C/N, type = "b", lwd = 2, col = "black", main = "", las = 1,
   ylab = "Proportion successful pairs", xlab = "Year", ylim = c(0, 1))
points(year, exp.p, type = "l", lwd = 3, col = "red")

return(list(nyears = nyears, alpha = alpha, beta1 = beta1,
   beta2 = beta2, year = year, YR = YR, exp.p = exp.p, C = C, N = N))
}
```

We create one data set, which is inspired by the data in Section 3.5.2.

```
data <- data.fn(nyears = 40, alpha = 1, beta1 = -0.03, beta2 = -0.9)
```

The model as written in the BUGS language is a trivial variant of the Poisson GLM encountered earlier; the key word for the binomial distribution is `dbin()`. Remember that the binomial distribution in WinBUGS is specified with the success parameter (p) *before* the binomial total (N).

```
# Specify model in BUGS language
sink("GLM_Binomial.txt")
cat("
model {

# Priors
alpha ~ dnorm(0, 0.001)
beta1 ~ dnorm(0, 0.001)
beta2 ~ dnorm(0, 0.001)

# Likelihood
for (i in 1:nyears){
    C[i] ~ dbin(p[i], N[i])        # 1. Distribution for random part
    logit(p[i]) <- alpha + beta1 * year[i] + beta2 * pow(year[i],2) # Link
       function and linear predictor
    }
}
",fill = TRUE)
sink()
```

We choose a different scaling of year this time and simply subtract 20 and divide the result by 20. This leads to values of the covariate that range from about −1 to 1.

```
# Bundle data
win.data <- list(C = data$C, N = data$N, nyears = length(data$C),
   year = data$YR)

# Initial values
inits <- function() list(alpha = runif(1, -1, 1), beta1 =
   runif(1, -1, 1), beta2 = runif(1, -1, 1))

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

# MCMC settings
ni <- 2500
nt <- 2
nb <- 500
nc <- 3

# Call WinBUGS from R (BRT < 1 min)
out <- bugs(data = win.data, inits = inits, parameters.to.save = params,
   model.file = "GLM_Binomial.txt", n.chains = nc, n.thin = nt,
   n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory = bugs.dir,
   working.directory = getwd())
```

We first check convergence by looking at the plots in WinBUGS and skimming over the values in the column entitled Rhat in the summary of the analysis in R (not shown). Both tests look very satisfactory, so we add a plot of the predicted proportion of successful pairs.

```
# Plot predictions
WinBUGS.predictions <- out$mean$p
lines(1:length(data$C), WinBUGS.predictions, type = "l", lwd = 3,
   col = "blue", lty = 2)
```

3.5.2 Analysis of Real Data Set

We fit the same model to the proportion of successful peregrine pairs in the French Jura. We do not need to define the model again, since we did this in the previous section already.

```
# Read data and attach them
peregrine <- read.table("falcons.txt", header = TRUE)
   attach(peregrine)

# Bundle data (note yet another standardization for year)
win.data <- list(C = R.Pairs, N = Pairs, nyears = length(Pairs),
   year = (Year-1985)/ 20)

# Initial values
inits <- function() list(alpha = runif(1, -1, 1), beta1 = runif(1, -1, 1),
   beta2 = runif(1, -1, 1))

# Parameters monitored
params <- c("alpha", "beta1", "beta2", "p")
```

```
# MCMC settings
ni <- 2500
nt <- 2
nb <- 500
nc <- 3

# Call WinBUGS from R (BRT < 1 min)
out3 <- bugs(data = win.data, inits = inits, parameters.to.save =
   params, model.file = "GLM_Binomial.txt", n.chains = nc, n.thin = nt,
   n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory = bugs.dir,
   working.directory = getwd())

# Summarize posteriors and plot estimates
print(out3, dig = 3)
plot(Year, R.Pairs/Pairs, type = "b", lwd = 2, col = "black", main = "",
   las = 1, ylab = "Proportion successful pairs", xlab = "Year",
   ylim = c(0,1))
lines(Year, out3$mean$p, type = "l", lwd = 3, col = "blue")
```

Convergence looks wonderful: the plots in WinBUGS look "grassy" (Link and Barker, 2010) and all Rhat values are close to 1. The proportion of successful pairs increased and then declined again over the years (Fig. 3.4c). The initial increase may be due to the recovery from pesticide effects and the decline to density dependence and the spread of a predator, the eagle owl.

3.6 SUMMARY AND OUTLOOK

In this chapter, we have covered much and important ground. We have illustrated the concept of a statistical model as being composed of a statistical distribution, to account of the noise, and a linear predictor for the signal. We have also introduced generalized linear models (GLMs); they allow distributions other than the normal to model the noise in a response. GLMs do so by using a link function, a transformation of the mean response that linearizes the relationship between the transformed mean response and covariates. We examined two common GLMs: Poisson and binomial. With the Poisson GLM, we saw how a classical analysis using maximum likelihood typically yields estimates that numerically closely match those from a Bayesian analysis with vague priors. In every example, we have illustrated convergence assessment of Markov chains by visual and formal means. We have modeled the temporal variation in counts exclusively, but the Poisson and binomial distributions may also be used to describe spatial variation of counts. We expect that much of this is a repetition for you rather than the first time you encounter the concept of a GLM. Otherwise it will be beneficial to first read more specific references that deal with this essential topic of applied statistical

modeling, for example, Crawley (2005), Dobson and Barnett (2008), and Kéry (2010). In the next chapter, we will generalize the GLM to include so-called random effects and to become a generalized linear mixed model (GLMM), an example of a hierarchical model. One other interesting extension of the GLM, to include nonlinear, "wiggly" terms to become a generalized additive model (GAM; Hastie and Tibshirani, 1990), is not dealt with in this book. However, such spline models can be implemented in WinBUGS as sort of random-effects model. For code examples, see Gimenez et al. (2006a, b) and Grosbois et al. (2009).

3.7 EXERCISES

1. Adapt the first data-generation function in this chapter to generate the data using coefficients that refer to the values of standardized covariate values and repeat the analysis in R and in WinBUGS.
2. Take the following toy data set and fit a logistic regression of the number of successes r among n trials as a function of covariate X. Also write out the GLM for this data set.

```
n <- c(22, 8, 10, 7, 10, 6, 11)
r <- c(20, 7, 10, 6, 0, 1, 2)
X <- c(0, 3, 1, 4, 5, 8, 10)
```

3. The Bernoulli distribution is a special case of the binomial with trial size equal to 1. It has only one parameter, the success probability p. The Bernoulli distribution is a conventional model for species distributions, where observed detection or nondetection data are related to explanatory (e.g., habitat) variables in a linear or other fashion with a logit link. Write an R function to assemble "presence or absence" data collected at 200 sites, where the success probability (i.e., occurrence probability) is related to habitat variable X (ranging from −1 to 1) on the logit-linear scale with intercept −2 and slope 5. Then write a WinBUGS program to "break down" the simulated data (i.e., analyze them) and thus recover these parameter values.
4. In Section 3.5.2, we used a binomial GLM to describe the proportion of successful peregrine pairs per year in the French Jura mountains. To see the connections between three important types of GLMs, first use a Poisson GLM to model the number of successful pairs (thus disregarding the fact that the binomial total varies by year), and second, use a normal GLM to do the same. In the same graph, compare the predicted numbers of successful pairs for every year under all three models (binomial, Poisson, and normal GLMs). Do this in both R and WinBUGS.