Chapter 14

Advanced Ecological Models

Chapter Outline

14.1 Hierarchical Multinomial Model to Analyze Habitat Selection Using BUGS 225
14.2 Zero-Inflated Poisson Mixed Model for Analyzing Breeding Success Using Stan 231
14.3 Occupancy Model to Measure Species Distribution Using Stan 240
14.4 Territory Occupancy Model to Estimate Survival Using BUGS 246
14.5 Analyzing Survival Based on Mark-Recapture Data Using Stan 252

14.1 HIERARCHICAL MULTINOMIAL MODEL TO ANALYZE HABITAT SELECTION USING BUGS

Categorical response variables are quite common in ecological studies. For example, when studying habitat selection of animals, the outcome variable often is one of several habitat types in which an animal is observed at different time points. Questions can be about whether the animal uses the different habitat types proportional to their availability in its home range, whether the use of the habitat types differs between the sexes or between young and adult animals, or whether the use of habitat types is affected by any covariate such as weather or age of the animal. The statistical methods used to analyze habitat selection with respect to such questions are manifold. They range from simple preference indices to complicated multivariate methods (see overview in Manly et al., 2002) or compositional analysis (Aebischer & Robertson, 1993).

All these classical methods assume independence among observations. However, particularly in telemetry studies, this is often not the case, because single individuals are followed over time producing repeated measurements per individual. The multinomial model is a linear model with a categorical outcome variable. To account for nonindependent data we use random effects, which we introduced earlier in the framework of generalized linear mixed models.

The multinomial model can be used in any case where the outcome variable is categorical, for example, to study differences in diet composition between the sexes or to analyze choice experiments in behavioral studies where the number of possible choices is larger than two (we could use logistic regression if the number of possible choices is two). One important requirement is that the observations are counts, for example, number of locations in specific habitat types or number of items eaten by an individual. Thus, multinomial models cannot be applied when the outcome variable is a continuous variable such as the proportion of time (measured in seconds) an animal has shown different behaviors or proportion of area (measured in square meters) of different habitat types in a home range. In such cases, multivariate methods or compositional analyses have to be considered, see, for example, Legendre and Legendre (2012).

In the multinomial model, it is assumed that a total of N_i counts of observation i are distributed among K categories with proportions $\mathbf{p}_i$ (the bold letter indicates a vector). N_i is the size parameter of the multinomial distribution and $\mathbf{p}_i$ is a probability vector. The length of $\mathbf{p}_i$ is the number of categories, K. The vector $\mathbf{p}_i$ contains the probabilities of the outcome for each of the categories. The sum of $\mathbf{p}_i$ is 1. The outcome variable $\mathbf{y}_i$ is a vector of length K that contains the number of counts in each category. The sum of $\mathbf{y}_i$ is N_i.

$$\mathbf{y}_i \sim Multinomial(\mathbf{p}_i, N_i)$$

Often, the data contain the outcome as a single observation, the category name, rather than as a vector of numbers per category. Then we can use the categorical distribution.

$$y_i \sim Categorical(\mathbf{p}_i)$$

The categorical distribution corresponds to a multinomial distribution with size parameter $N_i = 1$. In the following sections we assume that the data set contains the outcome as a category $k = 1, \ldots, K$. The probability vector $\mathbf{p}_i$ contains, for observation i, the expected values. These K values are probabilities that the outcome is category k. To model the relationship between the outcome and the predictor variables, for each category k, a separate linear predictor is needed.

$$z_{i,k} = \mathbf{X}_i \boldsymbol{\beta}_k$$

where $\mathbf{X}_i$ is a vector with the values of the predictors for observation i and $\boldsymbol{\beta}_k$ contains the model coefficients for category k. The inverse link function of the value $z_{i,k}$ gives the probability, $p_{i,k}$, that an observation with predictor values as in observation i is category k.

Then, a link function is needed that guarantees that all values of $\mathbf{p}_i$, $p_{i,1}, \ldots, p_{i,K}$, are probabilities (i.e., between 0 and 1) and that the sum of $\mathbf{p}_i$ is 1.

Classically, a multivariate version of the logistic function is used as the link function. Therefore, the multinomial model is sometimes also called the multinomial logistic model, or "mlogit" model (see, e.g., the R package mlogit; Croissant, 2012).

$$p_{i,k} = \frac{e^{z_{i,k}}}{\sum_{k=1}^{K} e^{z_{i,k}}} \quad \text{which makes that } \sum_{k=1}^{K} p_{i,k} = 1$$

The constraint that the probability vector $\mathbf{p}_i$ sums to 1 has important consequences: if all $p_{i,1}$ through $p_{i,K-1}$ are known, the last element of the probability vector ($p_{i,K}$) equals one minus the sum of the others. If $K-1$ linear predictors are estimated, the last one is defined by default. Therefore, we fix the $\boldsymbol{\beta}$ vector of one linear predictor (baseline category) to be a vector of zero values.

Multinomial models can be fitted in R using frequentist methods with the function `multinom` from the package nnet or the function `mlogit` from the package mlogit. The first does not allow inclusion of random factors, whereas it is possible with the `mlogit` function. Here, we use OpenBUGS to fit a multinomial model to habitat selection data in a Bayesian framework using two random effects.

Our example data are from Bock et al. (2013). They located little owls *Athene noctua* using radio telemetry during the daytime in winter and noted the type of roosting site: nest box, tree canopy, wood stack, or tree cavity. One important question was whether the choice of the roosting site type depends on ambient temperature. Thus, temperature was the predictor and type of roosting site the outcome variable. We include the individual and family as random factors because individuals were repeatedly located and the individuals were grouped into families. We define the model using BUGS code and save the BUGS code in the file "siteselection.bugs".

BUGS Code for the Habitat Selection Model (file "siteselection.bugs")

```
model {
  ## priors
  for(k in 2:ncat) {
    ## priors for model coefficients (see Chapter 15)
    for(j in 1:2) {
      beta[j,k] ~ dnorm(0, 0.04)
    }
    ## hyperpriors for random effects (see Chapter 15)
    sigmaind[k] ~dt(0,1,2)I(0,)
    sigmafam[k] ~dt(0,1,2)I(0,)
  }
  sigmaind[1] <- 0
  sigmafam[1] <- 0
```

Continued

BUGS Code for the Habitat Selection Model (file "siteselection.bugs")—cont'd

```
## random effects
for(i in 1:nind) {rind[i, 1] <- 0}     # zero for baseline category
for(f in 1:nfam) {rfam[f, 1] <- 0}     # zero for baseline category
for(k in 2:ncat) {
  for(i in 1:nind) {
    rind[i,k]~dnorm(0, 1)              # random individual effect
  }
  for(f in 1:nfam) {
    rfam[f,k]~dnorm(0, 1)              # random family effect
  }
}

## likelihood
for(i in 1:n) {
  roost[i]~dcat(p[i,1:ncat])
  for(k in 1:ncat) {
    ## linear predictors including the random factors
    z[i,k]<-beta[1,k] + beta[2,k]*temp[i] + sigmafam[k]*rfam
              [fam[i],k] + sigmaind[k]*rind[ind[i],k]
    expz[i,k] <- exp(z[i,k])
    p[i,k] <- expz[i,k] / sum(expz[i,1:ncat])   # logit link
  }
}

## constrain coefficients of the baseline category to zero
for(j in 1:2) {
  beta[j,1] <- 0
}

## predict site selection probabilities for different
temperatures
for(k in 1:ncat) {
  for(i in 1:nnew) {
    pnew[i,k] <- expznew[i,k] / sum(expznew[i,1:ncat])
    znew[i,k] <- beta[1,k] + beta[2,k]*newtemp[i]
    expznew[i,k] <- exp(znew[i,k])
    }
  }
}
```

Note that the random effects have to be fixed to zero for the reference category. Further, we specified the random effects by a standard normal distribution (*Norm*(0,1)) and multiply these effects by the between-individual and between-family standard deviation in the linear predictor. This improves the fitting process (Gelman et al., 2014).

In the last part of the model code, we obtain the fitted values, that is, for each roosting site type, we get the probability that the type is used given specific ambient temperature values (provided in the vector "newtemp"). These fitted values are visualized in Figure 14-1. To fit the model, we first bundle the data for OpenBUGS.

```
data(roostingsiteuse)
dat <- roostingsiteuse
dat$roosting.loc <- factor(dat$roosting.loc, levels=c("nest box",
   "tree canopy", "stack of wood", "tree cavity"))
dat$temp.z <- (dat$temp-mean(dat$temp))/sd(dat$temp)
dat$roostingnum <- as.numeric(dat$roosting.loc)

# Prepare data frame for predictions
newdat <- data.frame(temp=seq(min(dat$temp), max(dat$temp),
                                length=100))
newdat$temp.z <- (newdat$temp-mean(dat$temp))/sd(dat$temp)

datax <- list(roost=dat$roostingnum, n=nrow(dat),
              ncat=nlevels(dat$roosting.loc), temp=dat$temp.z,
              fam=dat$familynum, nfam=max(dat$familynum),
              ind=dat$indnum, nind=max(dat$indnum),
              newtemp=newdat$temp.z, nnew=nrow(newdat))
```

Then, we define a function that generates initial values for the betas and the random factors. Parameters that were fixed to zero in the model (baseline category) cannot have any initial value but NAs must be given.

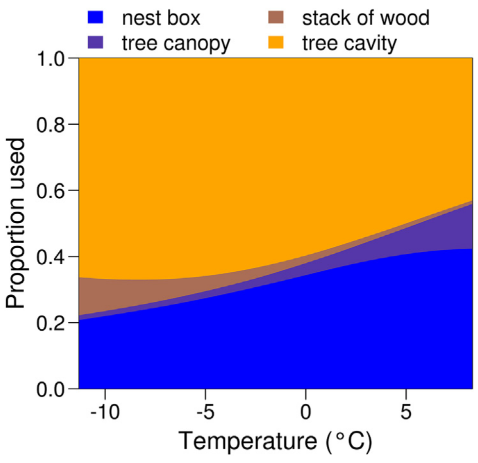

FIGURE 14-1 Proportions of four different roosting site types used by little owls *Athene noctua* during winter in relation to ambient temperature.

```
inits <- function() {
  list(beta=matrix(c(rep(NA,2), runif(2*(datax$ncat-1), -1, 1)),
                   ncol=datax$ncat, nrow=2),
       sigmafam=c(NA, runif(datax$ncat-1, 0.1, 2)),
       sigmaind=c(NA, runif(datax$ncat-1, 0.1,2)))
}
```

At last, we have to specify which parameters we want to save.

```
parameters <- c("beta", "sigmaind", "sigmafam","pnew")
```

Then, we can run the MCMC to fit the model:

```
library(R2OpenBUGS)
fit <- bugs(datax, inits, parameters, model.file="siteselection.bugs",
          n.thin=2, n.chains=2, n.burnin=1000, n.iter=10000,
          debug=FALSE)
```

When OpenBUGS has finished, we check the history plots of the Markov chains for indications of nonconvergence and, if all is ok, look at the results:

```
print(head(fit$summary, 12), 3)
```

```
                mean    sd    2.5%    25%    50%     75%  97.5% Rhat n.eff
beta[1,2]     -3.668 2.171 -8.2750 -5.103 -3.516 -2.1107  0.195 1.00  1500
beta[1,3]     -4.380 2.234 -8.8920 -5.934 -4.252 -2.7500 -0.371 1.00  2700
beta[1,4]      0.802 1.806 -2.7741 -0.378  0.795  1.9642  4.449 1.00  3500
beta[2,2]      0.988 1.177 -1.1261  0.176  0.906  1.7100  3.481 1.00  6500
beta[2,3]     -1.325 1.032 -3.6260 -1.936 -1.233 -0.6048  0.431 1.00  9000
beta[2,4]     -0.606 0.957 -2.6081 -1.194 -0.571  0.0318  1.209 1.00  5600
sigmaind[2]    2.717 1.453  0.1535  1.497  2.824  4.0052  4.909 1.00  9000
sigmaind[3]    3.396 1.154  0.7729  2.629  3.592  4.3610  4.943 1.00  6000
sigmaind[4]    2.301 1.453  0.0989  1.006  2.211  3.5595  4.846 1.00  2000
sigmafam[2]    3.288 1.312  0.3452  2.431  3.603  4.3662  4.943 1.00  7500
sigmafam[3]    3.013 1.345  0.2605  2.009  3.228  4.1640  4.912 1.00  9000
sigmafam[4]    3.948 0.896  1.6168  3.485  4.178  4.6390  4.968 1.01  2000
```

All $\widehat{R}$ are close to 1. Thus we have no indication of nonconvergence. However, the precision of the random effects is low. This is typical for random effects in models with categorical outcome variables including logistic regression. Both observations per group and the number of groups need to be high to get precise estimates of between- and within-group variances. In this example, sample size is way too small ($n = 42$) to estimate six additional variance parameters.

Here, at least one if not both of the random effects should be omitted and the model refitted (try this as an exercise). However, even if the uncertainty is high, due to too many random effects, the result is similar to the one obtained by Bock et al. (2013) based on a larger sample size. The second beta values are the estimated temperature effects: the negative values of beta[2,3] and beta[2,4] mean that wood stacks and tree cavities were used less often (compared to nest boxes) with increasing temperature. This is also apparent in Figure 14-1.

The advantage of using simulation techniques for describing the joint posterior distributions of the model parameters is that we can easily obtain a sample of simulated fitted values (of which the means are plotted in Figure 14-1) or other derived parameters. In the little owl example it is helpful to compare the proportion of used roosting sites with the corresponding proportion of available roosting sites, for example, by a preference index such as Jacob's preference index. This is done by simply calculating the Jacob's index for each simulated value of the proportion used; see Bock et al. (2013) for such an example.

14.2 ZERO-INFLATED POISSON MIXED MODEL FOR ANALYZING BREEDING SUCCESS USING STAN

Up until now we have described the outcome variable with a single distribution, such as the normal distribution in the case of linear (mixed) models, and Poisson or binomial distributions in the case of generalized linear (mixed) models. In life sciences, however, quite often the data are actually generated by more than one process.

In such cases the distribution of the data could be the result of two or more different distributions. If we do not account for these different processes our inferences are likely to be biased. In this and the next section, we introduce mixture models that explicitly include two processes that generated the data. The zero-inflated Poisson model is a mixture of a binomial and a Poisson distribution, and the site-occupancy model is a mixture of two binomial distributions. We belief that two (or more)-level models are very useful tools in life sciences because they can help uncover the different processes that generate the data we observe.

Counting animals or plants is a typical example of data that contain a lot of zero counts. For example, the number of nestlings produced by a breeding pair is often zero because the whole nest was depredated or because a catastrophic event occurred such as a flood. However, when the nest succeeds, the number of nestlings varies among the successful nests depending on how many eggs the female has laid, how much food the parents could bring to the nest, or other factors that affect the survival of a nestling in an intact nest. Thus the factors that determine how many zero counts there are in the data differ from the factors that determine how many nestlings there are, if a nest survives. Count data that are produced by two different processes—one produces the zero counts and the other the variance in the count for the ones that were not zero in the first process—are called zero-inflated data. Histograms of zero-inflated data look bimodal, with one peak at zero (Figure 14-2).

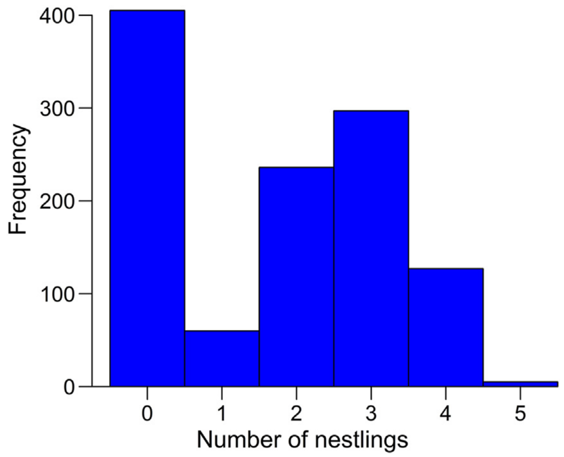

FIGURE 14-2 Histogram of the number of nestlings counted in black stork nests *Ciconia nigra* in Latvia ($n = 1130$ observations of 279 nests). The data were kindly provided by Maris Strazds.

The Poisson distribution does not fit well to such data, because the data contain more zero counts than expected under the Poisson distribution. Mullahy (1986) and Lambert (1992) formulated two different types of models that combine the two processes in one model and therefore account for the zero excess in the data and allow the analysis of the two processes separately.

The hurdle model (Mullahy, 1986) combines a left-truncated count data model (Poisson or negative binomial distribution that only describes the distribution of data larger than zero) with a zero-hurdle model that describes the distribution of the data that are either zero or nonzero. In other words, the hurdle model divides the data into two data subsets, the zero counts and the nonzero counts, and fits two separate models to each subset of the data. To account for this division of the data, the two models assume left truncation (all measurements below 1 are missing in the data) and right censoring (all measurements larger than 1 have the value 1), respectively, in their error distributions. A hurdle model can be fitted in R using the function `hurdle` from the package pscl (Jackman, 2008). See the tutorial by Zeileis et al. (2008) for an introduction.

In contrast to the hurdle model, the zero-inflated models (Mullahy, 1986; Lambert, 1992) combine a Bernoulli model (zero vs. nonzero) with a conditional Poisson model; conditional on the Bernoulli process being nonzero. Thus this model allows for a mixture of zero counts: some zero counts are zero because the outcome of the Bernoulli process was zero (these zero counts are sometimes called structural zero values), and others are zero because their outcome from the Poisson process was zero. The function `zeroinfl` from the package pscl fits zero-inflated models (Zeileis et al., 2008).

The zero-inflated model may seem to reflect the true process that has generated the data closer than the hurdle model. However, sometimes the fit of zero-inflated models is impeded because of high correlation of the model parameters between the zero model and the count model. In such cases, a hurdle model may cause less troubles.

Both functions (`hurdle` and `zeroinfl`) from the package pscl do not allow the inclusion of random factors. The functions `MCMCglmm` from the package MCMCglmm (Hadfield, 2010) and `glmmadmb` from the package glmmADMB (http://glmmadmb.r-forge.r-project.org/) provide the possibility to account for zero-inflation with a GLMM. However, these functions are not very flexible in the types of zero-inflated models they can fit; for example, `glmmadmb` only includes a constant proportion of zero values. A zero-inflation model using BUGS is described in Kéry and Schaub (2012). Here we use Stan to fit a zero-inflated model. Once we understand the basic model code, it is easy to add predictors and/or random effects to both the zero and the count model.

The example data contain numbers of nestlings in black stork *Ciconia nigra* nests in Latvia collected by Maris Stradz and collaborators at 279 nests between 1979 and 2010. Black storks build solid and large aeries on branches of large trees. The same aerie is used for up to 17 years until it collapses. The black stork population in Latvia has drastically declined over the last decades. Here, we use the nestling data as presented in Figure 14-2 to describe whether the number of black stork nestlings produced in Latvia decreased over time. We use a zero-inflated Poisson model to separately estimate temporal trends for nest survival and the number of nestlings in successful nests. Since the same nests have been measured repeatedly over 1 to 17 years, we add nest ID as a random factor to both models, the Bernoulli and the Poisson model. After the first model fit, we saw that the between-nest variance in the number of nestlings for the successful nests was close to zero. Therefore, we decide to delete the random effect from the Poisson model. Here is our final model:

$$z_{it} \sim Bernoulli(\theta_{it})$$

$$\text{logit}(\theta_{it}) = a_1 + a_2\text{year}_t + \varepsilon_{\text{nestID}[i]}$$

$$y_{it} \sim Poisson(\lambda_{it}(1 - z_{it}))$$

$$\log(\lambda_{it}) = b_1 + b_2\text{year}_t$$

$$\varepsilon_{\text{nestID}} \sim Norm(0, \sigma_n)$$

z_{it} is a latent (unobserved) variable that takes the values 0 or 1 for each nest i during year t. It indicates a "structural zero", that is, if $z_{it} = 1$ the number of nestlings y_{it} always is zero, because the expected value in the Poisson model $\lambda_{it}(1 - z_{it})$ becomes zero. If $z_{it} = 0$, the expected value in the Poisson model becomes λ_{it}.

To fit this model in Stan, we first write the Stan model code and save it in a separated text-file with name "zeroinfl.stan".

Stan Code for the Zero-Inflated Poisson Mixed Model

```
data {
     int<lower=0> N;                          // sample size
     int<lower=0> Nnests;                     // number of nests
     int<lower=0> y[N];                   // counts (number of youngs)
     vector[N] year;                          // numeric covariate
     int<lower=0, upper=Nnests> nest[N]; // index of nest
   }

   parameters {
     vector[2] a;                   // coef of linear pred for theta
     vector[2] b;                   // coef of linear pred for lambda
     real<lower=0> sigmanest;       // between nest sd in logit(theta)
     real groupefftheta[Nnests];  // nest effects for theta
   }

   model {
     //transformed parameters (within model to avoid monitoring)
     vector[N] theta;               // probability of zero youngs
     vector[N] lambda;              // avg. number of youngs
                                    // in successful nests
         for(i in 1:N){
         // linear predictors with random effect
         theta[i] <- inv_logit(a[1] + a[2] * year[i] +
                    sigmanest * groupefftheta[nest[i]]);
     }
     lambda <- exp(b[1] + b[2] * year);
     // priors
     a[1] ~ normal(0,5);
     a[2] ~ normal(0,5);
     b[1] ~ normal(0,5);
     b[2] ~ normal(0,5);
     sigmanest ~ cauchy(0,5);

     // random effects
     for(g in 1:Nnests) {
       groupefftheta[g] ~ normal(0,1);
     }

     // likelihood
     for (i in 1:N){
       if(y[i] == 0)
```

Stan Code for the Zero-Inflated Poisson Mixed Model—cont'd

```
        increment_log_prob(log_sum_exp(bernoulli_log(1, theta[i]),
                                    bernoulli_log(0, theta[i]) +
                                    poisson_log(y[i], lambda[i])));
      else
        increment_log_prob(bernoulli_log(0, theta[i]) +
                                    poisson_log(y[i], lambda[i]));
    }
  }
```

In the Stan model code we define ε_{nestID} by a standard normal distribution and add the product $\sigma_n \varepsilon_{\ nestID[i]}$ to the linear predictor. This parameterization of a random effect makes the model fit more stable. In the likelihood part, we use the if-else statement and define the likelihood separately for the zero and the nonzero counts. The zero count can be due to the outcome of the Bernoulli model being zero or because the count is zero. Therefore, the density function of the count of zero is the sum of the density function of the Bernoulli distribution for 1 (remember that 1 indicates zero) and the product of the Bernoulli-density for 0 (indicating not a "structural" zero) with the Poisson-density for zero. This is the expression within the `log_sum_exp` function.

The functions `bernoulli_log` and `poisson_log` define the logarithm of the density functions. The `log_sum_exp` function is the logarithm of the sum of the exponents, that is, log(exp(a) + exp(b)). The function `log_sum_exp` does this in a mathematically stable way. Finally, the function `increment_log_prob` is an alternative and flexible way of defining the likelihood in Stan; the code:

```
model{
  y ~ normal(mu, sigma);
}
```

is equivalent to:

```
model{
  increment_log_prob(normal_log(y, mu, sigma));
}
```

There are more efficient ways to code the above model in Stan. For example, the code `y ~ poisson_log(lambda)` includes the logarithm link function so that the exp function for the linear predictor can be skipped. Similarly, the `inv_logit` function can be omitted by using ~ `bernoulli_logit` instead of ~ `bernoulli`, or `bernoulli_logit_log` instead of `bernoulli_log`. Including the link function in the definition of the model by one of these functions makes the model fitting faster and more stable. However, here, we preferred to code the model so that the structure resembles the notations we have used throughout the book. We recommend the Stan manual to learn programming efficiently.

Then we start R, read and prepare the data, and run Stan to fit the model.

```
data(blackstork)
dat <- blackstork

dat$nest <- factor(dat$nest)                 # define nest as a factor
dat$year.z <- as.numeric(scale(dat$year))    # z-transform year

# Prepare data for Stan
y <- dat$njuvs
N <- nrow(dat)
nest <- as.numeric(dat$nest)
Nnests <- nlevels(dat$nest)
year <- dat$year.z
datax <- c("y", "N", "nest", "Nnests", "year")

# Run Stan
library(rstan)
 mod <- stan(file = "STAN/zeroinfl.stan", data=datax, chains=5, iter=1000)
print(mod, c("a", "b", "sigmanest"))

Inference for Stan model: zeroinfl.
5 chains, each with iter=1000; warmup=500; thin=1;
post-warmup draws per chain=500, total post-warmup draws=2500.

              mean  se_mean   sd   2.5%    25%    50%    75%  97.5%  n_eff Rhat
a[1]         -1.02     0.00 0.12  -1.26  -1.09  -1.01  -0.93  -0.79   2500    1
a[2]          0.56     0.00 0.11   0.35   0.48   0.56   0.63   0.79   2500    1
b[1]          0.89     0.00 0.03   0.84   0.88   0.89   0.91   0.95   2500    1
b[2]         -0.05     0.00 0.02  -0.10  -0.07  -0.05  -0.04  -0.01   2500    1
sigmanest     0.93     0.01 0.17   0.61   0.82   0.93   1.05   1.28    705    1

Samples were drawn using NUTS(diag_e) at Fri Sep 05 11:41:53 2014.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

The "mean" is the average of the simulated values from the marginal posterior distributions for each parameter. The "se_mean" is the standard error of these simulated values, that is, the Monte Carlo uncertainty and *not* the standard error of the parameter estimate! The sd is the sample standard deviation of the simulations. This corresponds to the standard error of the parameter estimates.

After several quantiles of the posterior distribution, the effective sample size n_eff and the potential scale reduction factor $\widehat{R}$ (see Section 12.3) are given. If we decided to draw more simulations, we would set the argument fit=mod to skip the model compilation step which saves time.

Before drawing conclusions from this model, the model fit is assessed using predictive model checking. To do so, we first simulate replicated numbers of nestlings for every year and nest in the data set based on the model fit.

```
nsim <- length(modsims$lp_)  # extract number of simulations
yrep <- matrix(nrow=length(y), ncol=nsim)
Xmat <- model.matrix(~year)
for(i in 1:nsim){
  theta <- plogis(Xmat%*%modsims$a[i,] +
             modsims$groupefftheta[i,nest])
  z <- rbinom(length(y), prob=theta, size=1)
  lambda <- (1-z) *exp(Xmat%*%modsims$b[i,])
  yrep[,i] <- rpois(length(y), lambda=lambda)
}
```

First, we visualize the observed and the first four simulated replicated data sets graphically, by plotting the number of nestlings for each nest and year (Figure 14-3).

The comparison of the simulated data with the observed data shows quite similar patterns, indicating that the model captures the general structure of the data. However, we can recognize that there are fewer zero values observed at the beginning of the study period than assumed by the model (the observed data shows fewer black dots in the lower left part of the panel than the simulated data). We could go back and fit a model allowing for a nonlinear relationship between year and the proportion of successful nests, or we can accept the small lack of fit. The overall proportion of zero values was the same in the replicated data (median 36%) as they were in the observed data (36%).

```
# Proportion of zeros for the observed data
mean(y==0)
[1] 0.3584071

# Proportion of zeros for the nsim simulated data
propzeros <- apply(yrep, 2, function(x) mean(x==0))
quantile(propzeros, prob=c(0.025, 0.5, 0.975))
     2.5%       50%     97.5%
0.3221239 0.3610619 0.4000000
```

The result provides strong evidence that the proportion of failed nests increased over the years, because the estimated effect of year in the zero-model was 0.56 (95% CrI 0.36–0.77). We extract these numbers from the

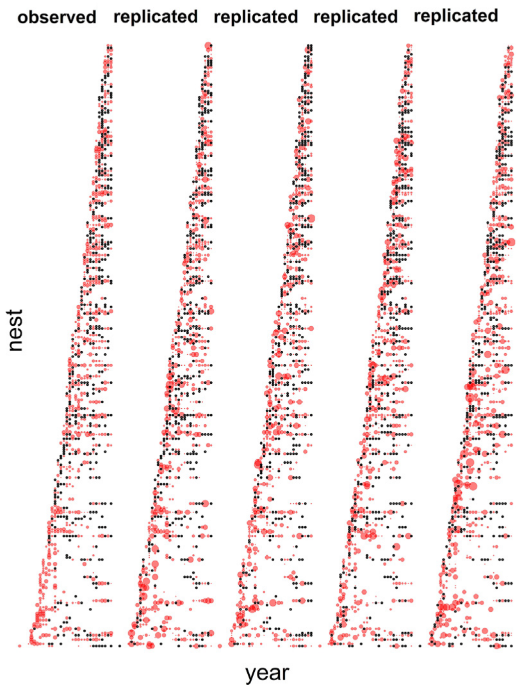

FIGURE 14-3 Numbers of nestlings for every nest in the different years. The smallest red dot corresponds to one, the largest to nine nestlings. The black circles indicate nests with zero nestlings. The nests are sorted according to their first year with a record in the data set. The leftmost panel is the observed data, the other panels are the first four (of 5000) simulated replicated data sets.

object mod1 that contains 5000 simulations from each model parameter using `extract` and `apply`:

```
modsims <- extract(mod1)
apply(modsims$a, 2, quantile, prob=c(0.025, 0.5, 0.957))
                 [,1]       [,2]
  2.5%   -1.2670004  0.3556119
  50%    -1.0105234  0.5623700
  95.7%  -0.8169778  0.7650336
```

Further, there is also evidence that the number of nestlings of the successful nests decreased over time because the estimated slope of the regression coefficient in the Poisson model is −0.05 (−0.10 to −0.02).

```
apply(modsims$b, 2, quantile, prob=c(0.025, 0.5, 0.957))
              [,1]         [,2]
  2.5%  0.8430767  -0.09894172
  50%   0.8948438  -0.05489263
  95.7% 0.9402984  -0.01526097
```

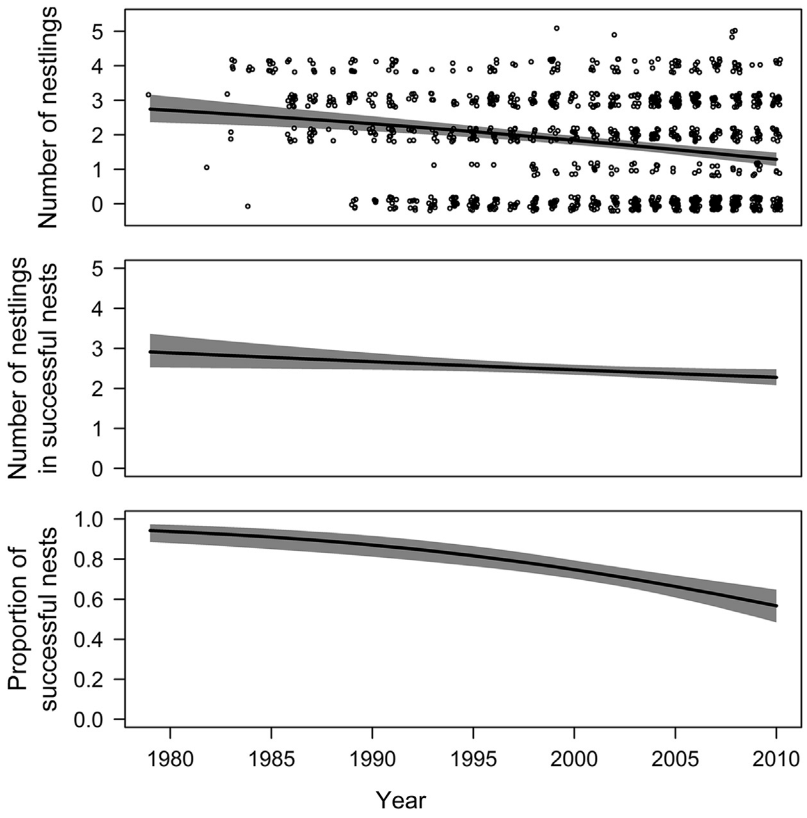

FIGURE 14-4 Fitted regression lines from the zero-inflated model with the number of nestlings as dependent variable: the proportion of successful nests (lowest panel), the number of nestlings given nest success (medium panel), and the average number of nestlings (averaged over all nests; top panel). The shaded area gives the 95% CrI. Sample size was 1130 observations of 279 nests.

To visualize the result more intuitively, we can draw three regression lines: (1) the proportion of successful nests, (2) the number of nestlings from the successful nests, and (3) the average number of nestlings, which is the product of the last two (Figure 14-4). To do so, we proceed as we have learned in the first part of the book: we create a new data frame that contains the predictor variable “year”. We transform this variable as we have done with the original variable before fitting the model. We create a new model matrix.

Then, we extract the means of the posterior distributions for the coefficients of the Bernoulli (“ahat”) and the Poisson model (“bhat”). We subtract the fitted values of the Bernoulli model from 1 to get the estimated proportions of nests that survived (remember that $z_{it} = 1$ means that the nest has died). We add the fitted value from the Poisson model, which is the average number of nestlings for the successful nests. The average number of nestlings per year (averaged over all nests including the ones that failed) is the product of the proportion of nests that survived and the average number of nestlings for successful nests.

```
newdat <- data.frame(year=1979:2010)
newdat$year.z <- (newdat$year - mean(dat$year))/sd(dat$year)
Xmat <- model.matrix(~year.z, data=newdat)
ahat <- apply(modsims$a, 2, mean)
bhat <- apply(modsims$b, 2, mean)
newdat$propsfit <- 1-plogis(Xmat %*% ahat)
newdat$nnestfit <- exp(Xmat %*% bhat)
newdat$avnnestfit <- newdat$propsfit*newdat$nnestfit
```

Now, repeat the preceding calculations many times (nsim) using each set of model parameters (i.e., a new iteration of the MCMC) once. Finally, we use, for every fitted value, the 2.5% and 97.5% quantiles of the nsim fitted values as lower and upper limits of a 95% CrI.

```
nsim <- length(modsims$lp_)
propsmat <- matrix(ncol=nsim, nrow=nrow(newdat))
nnestmat <- matrix(ncol=nsim, nrow=nrow(newdat))
avnnestmat <- matrix(ncol=nsim, nrow=nrow(newdat))
for(i in 1:nsim){
  propsmat[,i] <- 1-plogis(Xmat %*% modsims$a[i,])
  nnestmat[,i] <- exp(Xmat %*% modsims$b[i,])
  avnnestmat[,i] <- propsmat[,i]*nnestmat[,i]
}
newdat$propslwr <- apply(propsmat, 1, quantile, prob=0.025)
newdat$propsupr <- apply(propsmat, 1, quantile, prob=0.975)
newdat$nnestlwr <- apply(nnestmat, 1, quantile, prob=0.025)
newdat$nnestupr <- apply(nnestmat, 1, quantile, prob=0.975)
newdat$avnnestlwr <- apply(avnnestmat, 1, quantile, prob=0.025)
newdat$avnnestupr <- apply(avnnestmat, 1, quantile, prob=0.975)
```

14.3 OCCUPANCY MODEL TO MEASURE SPECIES DISTRIBUTION USING STAN

When surveying animal or plant populations, it is usually impossible to detect or capture all individuals in the study area. Inevitably, the probability of detecting or capturing an individual will differ between groups of individuals (e.g., between sexes), or may depend on habitat or other factors. If we do not account for these differences in detection or capture probability, the results drawn from such data are likely to be biased. There is a long tradition with a whole range of different models to account for the capture probability in studies that capture and individually mark animals (see the Further Reading section at the end of this chapter).

In contrast, for a long time the issue of imperfect detection (i.e., detection probability < 1) has been neglected in observational studies on animal and plant populations. However, mainly during the last two decades, a lot of effort has been put into the development of models that account for imperfect detection in animal and plant surveys, and nowadays it is becoming the gold standard to account for imperfect detection in most observational surveys.

Occupancy models are an important class of models that account for imperfect detection in animal and plant surveys. Occupancy models deal with the occupancy of a sampling unit by one or more species of interest (MacKenzie et al., 2006). In such studies, a number of sites are usually visited several times by a researcher to infer the presence of a species. Let's say we survey N sites (indexed by i) and visit each site J times (each visit is indexed by j). Our data (let's use the matrix y_{ij} for the data) is then a 1 if a species is observed and a 0 if the species is not observed. We assume that each site is occupied with probability ψ (which we usually call the "probability of occupancy"). Thus, for the true occupancy state x_i, we assume a Bernoulli process:

$$x_i \sim Bernoulli(\psi_i) \tag{14-1}$$

Note, that we can only partly observe whether or not a site is occupied: if we observe the species in a site, we know that this site is occupied, whereas if we do not observe a species in a site the site might not be occupied or we might not have seen the species although the site in fact was occupied. That is why we need to account for the detection process as well. To do so, the main trick is that we visit the site several times within a short period of time. We need to assume that the true occupancy state x_i does not change between the visits. Such an assumption is very common in ecological models that account for the observation process and is usually termed the "closed population assumption". To account for the observation process we assume a second Bernoulli process conditional on the true occupancy state x_i:

$$y_{ij} \sim Bernoulli(x_i p_{ij}) \tag{14-2}$$

Thus, during a visit we observe the species in site i with probability p only if the site is actually occupied by the species ($x_i = 1$). If the site is not occupied ($x_i = 0$), independent of the value of p_{ij}, we cannot observe the species. Therefore, the detection probability p_{ij} is defined as the probability of observing a species during a visit given the site was actually occupied. The Equations 14-1 and 14-2 form the basic structure of an occupancy model.

We now aim to analyze data from the amphibian monitoring in the canton of Aargau in Switzerland. For this monitoring, each of 572 water bodies was visited two times to infer the presence of yellow-bellied toad *Bombina variegata*. The presence of toads is mostly indicated by acoustic cues. Since the calling activity is likely to change over the breeding season, we expect detection probability of yellow-bellied toads to depend on the day when the survey was conducted. Thus, we add the day of the year as a linear and quadratic covariate of the detection probability p_{ij}:

$$\text{logit}\left(p_{ij}\right) = \beta_0 + \beta_1 * \text{DAY}_{ij} + \beta_2 * \text{DAY}_{ij} * \text{DAY}_{ij}$$

And all days? Is this reasonable?

And we assume that occupancy probability is equal among all sites.

$$\text{logit}(\psi_i) = \alpha_0$$

We can now use Stan to obtain estimates of the parameters. We write the Stan model code and save it in a text file with the name "occupancy.stan".

Stan Code for the Site-Occupancy Model (file "occupancy.stan")

```
data {
  int<lower=0> N;
  int<lower=2> J;
  int<lower=0, upper=1> y[N, J];
  int<lower=0, upper=1> x[N];
  real DAY[N, J];
}

parameters {
  real a0;
  real b0;
  real b1;
  real b2;
}

transformed parameters {
  real<lower=0,upper=1> psi[N];
  real<lower=0,upper=1> p[N, J];
  for(i in 1:N) {
    psi[i] <- inv_logit(a0);  //You may add covariates here
    for(j in 1:J) {
      p[i, j] <- inv_logit(b0 + b1*DAY[i, j] +
                            b2*DAY[i, j]*DAY[i, j]);
    }
  }
}
```

Stan Code for the Site-Occupancy Model (file "occupancy.stan")—cont'd

```
  model {
    // Priors (see Chapter 15)
    a0 ~ normal(0, 5);
    b0 ~ normal(0, 5);
    b1 ~ normal(0, 5);
    b2 ~ normal(0, 5);

  // likelihood
  for(i in 1:N) {
    if(x[i]==1) {
      1 ~ bernoulli(psi[i]);
      y[i] ~ bernoulli(p[i]);
    }
    if(x[i]==0) {
      increment_log_prob(log_sum_exp(log(psi[i]) + log1m(p[i,1]) +
                            log1m(p[i,2]), log1m(psi[i])));
      }
    }
  }
```

In Stan, the code usually looks a bit more complex than when using BUGS. This is mainly because we have to strictly define parameters and data structures. Furthermore, discrete latent variables such as x_i are allowed in BUGS but not (yet) in Stan. In Stan we need to find a way around it. We do so by using the if-statement and separately formulating the likelihood for sites where the species has been observed at least once, and sites where the species has not been observed.

Now, we are ready to estimate the parameters using the yellow-bellied toad data. We load the data, assign shorter names, and calculate some additional data that are needed in the Stan model.

```
data(yellow_bellied_toad)
y <- yellow_bellied_toad$y          # Observational data
x <- as.integer(apply(y, 1, sum)>0) # Observed (naive) occupancies
N <- dim(y)[1]                      # Number of sites
J <- dim(y)[2]                      # Number of visits
DAY <- yellow_bellied_toad$DAY      # Julian days of observations
DAY <- (DAY-150)/10              # scale DAY for easier convergence
```

We need to bundle the data and run the model using Stan.

```
datax <- c("y", "N", "J", "x", "DAY")
fit <- stan(file = "occupancy.stan", data = datax, iter = 2000,
            chains = 2)
```

A summary of the results is printed to the R console with the function `print`.

```
print(fit, c("a0", "b0", "b1", "b2"))
 Inference for Stan model: occupancy.
 2 chains, each with iter=2000; warmup=1000; thin=1;
 post-warmup draws per chain=1000, total post-warmup draws=2000.

      mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
a0  -1.16    0.00 0.11 -1.38 -1.23 -1.16 -1.09 -0.96   863    1
b0   1.53    0.01 0.29  0.97  1.33  1.52  1.73  2.10   772    1
b1  -0.77    0.01 0.20 -1.17 -0.90 -0.77 -0.63 -0.40   704    1
b2  -0.50    0.00 0.09 -0.68 -0.57 -0.50 -0.44 -0.34   638    1

Samples were drawn using NUTS(diag_e) at Fri Sep 05 12:07:34 2014.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

All $\widehat{R}$ are close to 1. Thus, convergence is likely to be fine. To graphically assess convergence, the Markov chains can be plotted using the function `historyplot` (package blmeco) or `traceplot` (rstan).

```
historyplot(fit, "a0")     # figure not shown
```

Now, we are ready to draw inferences. First, we can make a figure of the change in detection probability over the season (Figure 14-5). We calculate fitted values for detection probability from 1 May to 1 July, which corresponds to day 120 to day 181 of the year. We extract the simulated values from the posterior distribution of β_0, β_1, and β_2, and for each of these simulated values

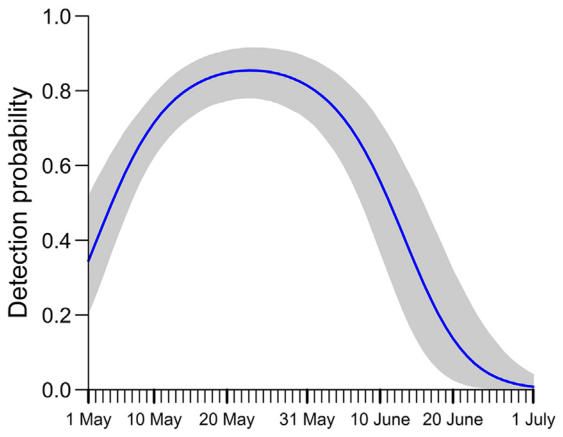

FIGURE 14-5 Fitted detection probability of yellow-bellied toads *Bombina variegata* over the breeding season. The shaded area gives the 95% CrI. Sample size was 572 sites.

and each day between 1 May and 1 July we calculate the fitted value of the detection probability and store them in the matrix fitp.

```
new.DAY <- 120:181
new.trDAY <- (new.DAY-150)/10
modsims <- extract(fit)
nsim <- dim(modsims$b0)[1]
fitp <- array(dim=c(nsim, length(new.DAY)))
for(i in 1:nsim)
  fitp[i,] <- plogis(modsims$b0[i] + modsims$b1[i]*new.trDAY +
                        modsims$b2[i]*new.trDAY^2)
```

Now, we can extract the mean and the lower and upper limits of the 95% CrI and make the figure.

Apparently, detection probability of yellow-bellied toads peaks in the last week of May: a researcher visiting an occupied site during the end of May will detect the species with a probability > 0.8. Since the presence of the species is usually inferred from calling males, we believe that the pattern of detection probability over the season strongly mirrors the calling activity of yellow-bellied toads. Acoustic signaling of many species can strongly change over the course of the day or within a short time period during the breeding season, and detection probability will change accordingly. The yellow-bellied toad data set is thus a good example to show the importance of accounting for detection probability in observational studies: if one does not account for detection probability, visiting the different sites at different dates would bias the results.

Now, we aim to draw inferences on the proportion of sites occupied by yellow-bellied toads, which is the main parameter of interest. In our occupancy model we implemented the true occupancy on the logit scale. Therefore, to draw inferences about occupancy probability we need to back-transform the estimates of α_0:

```
modsims <- extract(fit)
quantile(plogis(modsims$a0), prob=c(0.025, 0.5, 0.957))
     2.5%         50%       95.7%
0.2046530  0.2384540  0.2737034

# observed (naive) occupancy
mean(x)
[1] 0.220279
```

Thus, when accounting for detection probability in our toad example, we estimate that 23.8% (95% CrI: 20.5–27.4%) of the sites were truly occupied. This is not much different from the 22% of sites where we actually observed toads. Such a high detection rate could only be achieved because most of the visits were conducted during the period of highest detection probability and repeated visits were done, which increases the per-year detection probability compared to the per-visit detection probability shown in Figure 14-5.

14.4 TERRITORY OCCUPANCY MODEL TO ESTIMATE SURVIVAL USING BUGS

Estimates of survival and other demographic parameters are usually obtained from data on individually marked animals (Section 14.5). However, a major drawback of these methods is that they are usually invasive because individuals need to be captured and marked. This can be stressful for the animals and time- and labor-intensive for researchers. In contrast, noninvasive observational data from yearly population counts are rarely used to estimate survival because the individuals are not individually marked. Obtaining survival estimates from data of unmarked individuals is currently a very active field of research (e.g., Roth & Amrhein, 2010; Zipkin et al., 2014). The R package unmark is entirely devoted to the statistical analysis of data from surveys of unmarked animals (Fiske & Chandler, 2011).

Our example is from a 10-year population study on nightingales *Luscinia megarhynchos*. Each year during the breeding season, 55 nightingale territories are surveyed several times to check whether they are occupied by male nightingales (Amrhein et al., 2002). Territories were relatively stable across years irrespective of the identity of the territory holder, because nightingales frequently use the edges of bushes, paths, or rivers as territory borders. From these territory occupancy data we aimed to extract information about the survival of the individual male nightingales that occupied the territories.

The estimation of individual survival is complicated because of two main problems: first, territories that are occupied during two consecutive years might not be occupied by the same individual both years. Second, it might be possible that the territory holder was not detected during the surveys and, thus, some occupied territories might be assumed to be unoccupied.

We suggest that the territories that were surveyed for the presence of nightingales might be regarded as the single sites of site occupancy models that are surveyed for the presence of a species (Section 14.3). We thus decided to adapt a multiseason site occupancy model (MacKenzie et al., 2006) to account for the fact that some territory holders might switch territories from year to year, while some other territory holders may not survive to the next year and the territory might be occupied by a new territory holder. We denote the observed nightingale territory occupancy data with y_{itj}: it contains a 1 if a singing nightingale was heard in territory i during year t and visit j and if no nightingale was observed it contains a 0. We assume that three main processes can describe these data.

The first is the survival of individual nightingales from year $t-1$ to year t and returning to the same territory (denoted as Φ). Further, a territory can only be occupied by the same individual as in the previous years if the individual has been present in the previous year, that is, $x_{it\text{-}1} = 1$.

$$z_{it} \sim Bernoulli(x_{it-1}\Phi)$$

The variable z_{it} is a state variable that indicates whether a territory is occupied by the same nightingale as in the previous year. The second process is the colonization of an unoccupied territory by a new individual. The colonization probability is *r*.

$$x_{it} \sim Bernoulli(z_{it} + r[1 - z_{it}])$$

The state variable x_{it} indicates whether a territory i is occupied in year t either by a new individual or by the same one from the previous year. We believe that a territory is occupied by a nightingale if the territory owner from the previous year occupies the territory again (i.e., z_{it} =1) or if the territory is occupied by a new territory owner with probability *r*.

Finally, since a nightingale is not always singing, we might not detect the nightingale in an occupied territory, which is the third important process influencing our data. Consequently, we need to account for this observation process as well. We assume that we observe the territory owner of an occupied territory with probability *p*.

$$y_{itj} \sim Bernoulli(x_{it}p)$$

We now have described how we think our data (y_{itj}) have been generated. The entire model thus includes three Bernoulli distributions with three different parameters (Φ, r, p). Note, however, that z_{it} depends on the territory occupancy of the previous year ($x_{it\text{-}1}$), for which we have no data during the first year. Thus we need yet another model to describe the true occupancy of nightingales during the first year (x_{i1}) and we then start to estimate survival and colonization probability from the second year. We included the additional parameter Ω which is the probability that a territory is occupied during the first study year.

$$x_{il} \sim Bernoulli(\Omega)$$

Now, everything is defined and we can use OpenBUGS to obtain the posterior distributions of the model parameters. We write the BUGS model code and save it in a text file with name "territoryoccupancy.txt".

BUGS Code for the Territory-Occupancy Model (file: "territoryoccupancy.txt")

```
model{
# priors
omega ~ dunif(0,1)
phi ~ dunif(0,1)
r ~ dunif(0,1)
p ~ dunif(0,1)
```

Continued

BUGS Code for the Territory-Occupancy Model (file: "territoryoccupancy.txt")—cont'd

```
# likelihood
for(i in 1:N){
  x[i,1] ~ dbern(omega)
  for(t in 2:T) {
    mu.phi[i,t] <- x[i,t-1]*phi
    z[i,t] ~ dbern(mu.phi[i,t])
    mu.r[i,t] <- z[i,t] + (1-z[i,t])*r
    x[i,t] ~ dbern(mu.r[i,t])
    for(j in 1:J) {
      mu.p[i,t,j] <- x[i,t]*p
      y[i,t,j] ~ dbern(mu.p[i,t,j])
    } # close j
  } # close t
} # close i
}
```

Note that during the first study year we only estimate which of the territories are occupied. Only from the second year (the *t*-loop starts with "2") can we estimate Φ and r. Now, let's see whether we can estimate the parameters using the nightingale data. We load the data using

```
data(nightingales)
```

As usual, we assign a shorter name and look at the data.

```
y <- nightingales
class(y)

[1] "array"

dim(y)

[1] 55 10 8
```

The data comes in the form of a three-dimensional array: the first dimension contains the 55 territories, the second dimension contains the 10 years of the study (2000—2009), and the third dimension contains the eight visits.

To fit the model, we need to calculate the number of territories, the number of study years, and the number of yearly visits, and then bundle the data for OpenBUGS.

```
N <- dim(y)[1]        # Number of territories
T <- dim(y)[2]        # Number of study years
J <- dim(y)[3]        # Number of yearly visits
datax <- list(y=y, N=N, T=T, J=J)
```

Then, we define a function that generates initial values for the four parameters (Ω, Φ, r, p). We should also give some sensible starting values for x, which helps to avoid OpenBUGS from getting stuck from time to time.

```
inits <- function() {
  x <- array(NA, dim = c(N,T))
  for(t in 1:T) x[,t] <- as.integer(apply(y[,t,], 1, sum)>0)
  list(omega = runif(1,0,1),
        phi = runif(1,0,1),
        r = runif(1,0,1),
        p = runif(1,0,1),
        x=x)
}
```

As always, we have to specify which parameters we want to save.

```
parameters <- c("phi", "r", "p", "omega")
```

Then, we can run the MCMC to fit the model:

```
bugsmod <- bugs(datax, inits, parameters,
  model.file="territoryoccupancy.txt",
  n.thin=2, n.chains=2, n.burnin=1000, n.iter=5000,
  working.directory=bugsworkingdir)
```

When OpenBUGS has finished, we check the history plots of the Markov chains for indications of nonconvergence.

```
print(bugsmod$summary[,c("mean", "2.5%", "97.5%", "Rhat")], 3)
                mean      2.5%      97.5%  Rhat
  phi          0.591     0.507     0.670     1
  r            0.256     0.202     0.315     1
  p            0.487     0.466     0.508     1
  omega        0.797     0.684     0.891     1
  deviance 3189.221  3171.000  3224.000     1
```

All $\widehat{R}$ are close to 1. Thus, there is no indication of nonconvergence. However, before drawing conclusions from this model, we do some predictive model checking to see how well our model fits the data. To do so, we simulate replicated territory occupancy data for each of the simulated parameter values in the model fit. We simulate nsim replicated sets of territory occupancy data and summarize each set of territory occupancy data using four different test statistics.

The first (nobstot) is the total number of times a territory was observed to be occupied, the second (meanobster) is the average number of years the territory was occupied, the third (obsy1) is the number of observations during the first visit, and the fourth (obsy5) is the number of observations during the fifth visit. First, we need to prepare the vectors where we can store the results of the summary statistics:

```
nsim <- bugsmod$n.sims
nobstot <- integer(nsim)
meanobster <- integer(nsim)
obsy1 <- integer(nsim)
obsy5 <- integer(nsim)
```

Then we start to simulate replicated territory occupancy data and calculate the summary statistics using the following code:

```
for(i in 1:nsim){
  x <- array(dim = c(N, T))
  z <- array(dim = c(N, T))
  yrep <- array(dim = c(N, T, J))
  x[,1] <- rbinom(N, 1, bugsmod$sims.matrix[, "omega"][i])
  for(j in 1:J) {
    yrep[,1,j] <- rbinom(N, 1, x[,1]*bugsmod$sims.matrix[, "p"][i])
  }
  for(t in 2:T) {
    z[,t] <- rbinom(N, 1, x[,t-1]*bugsmod$sims.matrix[, "phi"][i])
    x[,t] <- rbinom(N, 1, z[,t] + (1-z[,t])*
                       bugsmod$sims.matrix[, "r"][i])
    for(j in 1:J) {
       yrep[,t,j] <- rbinom(N, 1, x[,t]*bugsmod$sims.matrix[, "p"][i])
    }
  }
  nobstot[i] <- sum(yrep)
  nobster <- rep(0,N)
  for(t in 1:T) {
     nobster <- nobster + as.integer(apply(yrep[,t,], 1, sum)>0)
  }
  meanobster[i] <- mean(nobster)
  obsy1[i] <- sum(yrep[,,1])
  obsy5[i] <- sum(yrep[,,5])
}
```

Now, we are ready to compare the summary statistics from the simulated values with the summary statistics calculated from the real data. For the summary statistics of the simulated values we calculate 1% and 99% quantiles.

```
quantile(nobstot, probs = c(0.01, 0.99))
      1%      99%
  875.98  1345.01
sum(y)
[1] 1120
```

During the surveys we observed a territory-holding nightingale 1120 times. This is well within the 1% and 99% quantiles of the replicated data (876 and 1345). In this respect, our model is doing a good job. So let's have a look at the average number of years the territories were occupied.

```
quantile(meanobster, probs = c(0.01, 0.99))
      1%       99%
4.127273  6.200000
nobster <- rep(0,N)
for(t in 1:T) {
  nobster <- nobster + as.integer(apply(y[,t,], 1, sum)>0)
}
mean(nobster)
[1] 5.2
```

Again the 5.2 years the territories were, on average, occupied during the study period is well within the 1% and 99% quantiles of the replicated data. Again, we can be satisfied with our model. So we can do the last check. What if we compare only the number of observations during the first visit or only the observations during the fifth visit to each of the territories?

```
quantile(obsy1, probs = c(0.01, 0.99))
     1%    99%
    105    174
sum(y[,,1])
  [1] 60
quantile(obsy5, probs = c(0.01, 0.99))
    1%    99%
   105    175
sum(y[,,5])
[1] 145
```

Now, we have detected an aspect that might be of concern: we only made 60 observations of territory-holding nightingales during the first visit, which is well below the number of observations in the replicated data. In contrast, during the fifth visit the replicated data again fit well with the observations. So what went wrong during the first visit? We assumed constant detection probability during each visit; however, this might be unrealistic as singing activity of nightingales is lower during the beginning of the breeding season than later on in the breeding season (Roth et al., 2012). Thus, we might considerably improve our model by fitting survey-specific detection probabilities. We encourage the reader to do this as an exercise.

Nevertheless, we draw some inferences using the model with constant detection probability. Our estimate for the probability that an individual survives from one year to the next and settles in the same territory again is 0.59 (95% CrI: 0.50–0.67). This estimate is very similar to the survival we estimated from a subsample of individually marked nightingales of the same study population using mark-recapture analyses (Roth & Amrhein, 2010). However, the territory occupancy model allowed including a larger number of individuals in the study because unmarked individuals could also be used.

Therefore, conclusions can be made for a broader population. Of course, the individual-level information about survival is weaker for a sample of unmarked individuals than for the sample of marked individuals. Thus, a higher sample size is needed to obtain the same precision of the survival estimate. If both marked and unmarked individuals exist in a population, an elegant solution would be to combine marked and unmarked individuals in the same model. Using BUGS, it is straightforward to expand the previous model to such a combined model. To learn more about how to combine data from different data sources, we suggest scanning the recent literature about "integrated population models" (e.g., Schaub et al., 2007).

14.5 ANALYZING SURVIVAL BASED ON MARK-RECAPTURE DATA USING STAN

To study the underlying factors that cause animal populations to increase or decrease, individual animals are usually marked and subsequently recaptured to infer information about the demographic parameters such as individual survival or population size. The difficulty with mark-recapture data is that not all marked individuals are recaptured during each capture occasion. If an individual has not been captured, it may have died, emigrated from the study area, or it may have escaped being captured. For the study of survival it is important to disentangle mortality and emigration from capture probability. To do so, Cormack (1964), Jolly (1965), and Seber (1965) developed a statistical model, the Cormack–Jolly–Seber model (CJS), which became the method used by myriad population biologists.

The CJS model explicitly models the capture process (the observation process) conditional on a survival model (the biological process). By including an observation process model, the recapture probability and thus the proportion of individuals still alive and present but not recaptured is estimated and taken into account while estimating the probability that an individual survives and stays in the study area, the so-called apparent survival. The individuals do not need to be recaptured physically, an observation of an individual is sufficient to know that the individual is still alive. In such cases, resighting probabilities instead of recapture probabilities are estimated. To formulate the CJS model, we use a partially observed variable z_{it} that indicates whether individual i is alive and present at time t. The model assumes a Bernoulli process for z_{it}:

$$z_{it} \sim Bernoulli(z_{it-1}\Phi_{it})$$

In other words, an individual i is alive and in the study area at time t ($z_{it} = 1$) with probability Φ_{it}, if it was alive and in the study area at time $t - 1$ ($z_{it\text{-}1} = 1$), or 0, if it was dead (or had permanently emigrated from the study area) at time $t - 1$ ($z_{it\text{-}1} = 0$). The parameter Φ_{it} is the apparent survival probability. The name "apparent" indicates that mortality and emigration from the study area cannot be separated from capture-recapture data (from one study area). A linear predictor for the logit (or another link function) of Φ_{it} can be added to study factors affecting apparent survival.

$$\text{logit}(\Phi_{it}) = \mathrm{X}_{it}\boldsymbol{\beta}$$

This part of the model looks like an autoregressive logistic regression. However, the variable z is only partly observed. It is known that an individual is alive ($z_{it} = 1$) when it has been recaptured or resighted at time t or at any time point later ($y_{it} = 1$). When an individual is not recaptured ($y_{it} = 0$) during any subsequent capture occasion, we do not know whether it has died or whether it is still alive but has not been captured or resighted. The capture process is added to the model as a second Bernoulli process.

$$y_{it} \sim Bernoulli\left(z_{it}p_{it}\right)$$

where p_{it} is the probability that individual i that is alive and within the study area is captured during capture occasion t. Also, a linear predictor can be added for the logit of p_{it} to study factors affecting the capture probability.

In the original CJS both Φ and p were fully time dependent, that is, for each capture occasion t, independent Φ and p values were estimated. In this fully time dependent model, the Φ and p are not estimable for the last capture occasion, only their product Φp is estimable. Including linear predictors for apparent survival probability or recapture probability makes the two parameters estimable also for the last capture occasion. But they add the constraint that apparent survival probability or recapture probability is constant when all the covariates are constant. This resembles the complete pooling case (Section 7.1.2): the parameter is estimated as an overall mean assuming no differences between the capture occasions. On the other hand, the fully time dependent variant corresponds to the no pooling case, because for each capture occasion an independent parameter is estimated.

Often the reality is something between the two cases, and given we remember what we learned in Section 7.1.2, we use random effects. The advantage of random effects in mark-recapture models was recognized several years ago (Burnham & White, 2002; Royle & Link, 2002). The possibility to use capture occasion as a random effect has been implemented in standard mark-recapture software such as the program MARK (Burnham & White, 2002). The inclusion of individual-level random effects has become feasible only recently with improved computer capacities and MCMC algorithms (Pledger et al., 2003; Royle, 2008, Gimenez & Choquet, 2010; Ford et al., 2012).

However, individual-level random effects are still not widely used, mainly because the fit of such models in user-friendly software such as MARK or BUGS is extremely time consuming. For example, to fit the model we present in the following in BUGS took several days. With Stan we obtained a similar number of effective samples from the posterior distributions in only about three hours. Though this comparison does not meet scientific standards, it is an encouraging observation and motivation for describing here how to fit a CJS model with random effects in Stan.

The example data are from a four-year radio-telemetry study on the post-fledging survival of barn swallows *Hirundo rustica* (Grüebler & Naef-Daenzer, 2008; Grüebler & Naef-Daenzer, 2010). We selected data for the first brood only. The data can be loaded using

```
data(survival_swallows)
```

For convenience, we give a shorter name and look at the data.

```
datax <- survival_swallows
str(datax)
List of 8
$ CH      : int [1:322, 1:18] 1 1 1 1 1 1 1 1 1 1 ...
    ..- attr(*, "dimnames")=List of 2
    .. ..$ : NULL
    .. ..$ : chr [1:18] "day0" "day1" "day2" "day3" ...
$ I       : int 322
$ K       : int 18
$ carez   : num [1:322] -0.584 -0.584 -0.584 -0.584 -0.584 ...
$ year    : num [1:322] 1 1 1 1 1 1 1 1 1 1 ...
$ agec    : num [1:18] -8.5 -7.5 -6.5 -5.5 -4.5 -3.5 ...
$ family  : num [1:322] 5 5 5 5 5 1 1 1 4 4 ...
$ nfam    : num 72
```

The data comes in the format of a list as it is used for BUGS. The element "CH" is a matrix. Every row of the matrix is a capture history of one individual. Every column corresponds to one day after fledging. The first column is the day of fledging. Only individuals that have fledged have been studied. Therefore, the first column contains a 1 for every individual. The entries in columns 2 to 18 indicate whether the individuals have been observed alive at day $t = 1, \ldots, 17$. "I" is the total number of individuals, that is, the number of rows of "CH", and "K" is the number of occasions (here including the day of fledging). "carez" is a family-level covariate that indicates how long the parents have cared for the fledglings after fledging. This variable was z-transformed. "year" contains the numbers 1 to 4, indicating the year of the study. "agec" is the day after fledging (correlating with the age of the individuals) centered around 0. This variable is used as a covariate for the detection probability. The centering eliminates the correlation between the estimate for the intercept and the slope and therefore improves the model-fitting process (better mixing of the MCMC, higher number of effective samples in less time). "family" indicates the family the individual belongs to and "nfam" is the total number of families.

In this exercise we would like to estimate, for every day after fledging, an independent daily survival probability averaged over the four years and corrected for the effort the parents put into the care of their fledglings. We further include family as a random effect because individuals from the same family are not independent, for example, they received the same parental care.

The linear predictor in the survival model contains one intercept per day t, a linear effect of carez, and a year- and family-level modulation of the intercept:

Main variable of interest

$$\text{logit}(\Phi_{it}) = \alpha_{0t} + \alpha_1 \text{carez}_i + \varepsilon_{\text{year}[i]} + \gamma_{\text{family}[i]}$$

We assume a normal distribution for the year and family random effects.

$$\varepsilon_{\text{year}} \sim Norm(0, \sigma_{\Phi\text{year}})$$

$$\gamma_{\text{family}} \sim Norm(0, \sigma_{\Phi\text{family}})$$

In the linear predictor of the capture probability (in this case we should call this an encounter probability), we include time since fledging (agec) as a covariate because the behavior of the birds is gradually changing as they get older. Because different types of radio transmitters, which differed in coverage, were used in the four different years, the effect of age on observation probability may be different in the four years. We, therefore, included year-specific intercepts and slopes without partial pooling (i.e., agec, year, and their interaction are fixed effects). As a random effect, we add family, because the individuals are moving in family flocks. Thus, if one individual is detected, the probability of also detecting the other members of this family is very high.

$$\text{logit}(p_{it}) = \beta_{0\text{year}[i]} + \beta_{1\text{year}[i]}\text{agec}_i + \delta_{\text{family}[i]}$$

$$\delta_{\text{family}} \sim Norm(0, \sigma_{p\text{family}})$$

In the Stan code, we specify the random effects as standard normal distributions ($Norm(0,1)$) and multiply the random effects with the sigma-parameter within the linear predictor. This parameterization results in a faster and more reliable fit when using MCMC. An important difference between Stan and BUGS is that Stan cannot (yet) handle discrete latent (= unobserved) variables. In the CJS model, after the last recapture of individual i, z_{it} (for $t > last_i$) is an unobserved discrete variable. The problem is circumvented by discarding all observations after the last recapture of each individual by running the likelihood in the model code only until the last observation (`k in 1:last[i]` instead of `k in 1:K`). Then, we add the logarithm of the probability that the individual is never recaptured between its last recapture and the end of the study to the log-likelihood in a separate step. The variable $last_i$ is created in the “transformed data” block. The probability that individual i is never recaptured between its last recapture and the end of the study is defined in the “transformed parameters” block, and named “chi”.

Stan Code for the CJS Model with Random Effects (file "CJS_swallows-stan")

```
data {
  int<lower=2> K;                          // capture events
  int<lower=0> I;                          // number of individuals
  int<lower=0,upper=1> CH[I,K];            // CH[i,k]: individual i
                                           // captured at k
  int<lower=0> nfam;                       // number of families
  int<lower=0, upper=nfam> family[I];// index of group variable
  vector[I] carez;         // duration of parental care, z-trans.
  int<lower=1,upper=4> year[I];            // index of year
  vector[K] agec;                        // age of fledling, centered
}
transformed data {
  int<lower=0,upper=K+1> last[I];      // last[i]: ind i last
                                       //capture
  last <- rep_array(0,I);
  for (i in 1:I) {
    for (k in 1:K) {
      if (CH[i,k] == 1) {
        if (k > last[i]) last[i] <- k;
      }
    }
  }
}
parameters {
  real b0[4];                              // intercepts per year for p
  real b1[4];                        // slope for age per year for p
  real a[K-1];                             // intercept of phi
  real a1;                                 // coef of phi
  real<lower=0> sigmaphi;   // between family sd in logit(phi)
  real<lower=0> sigmayearphi;   // between-year sd in logit(phi)
  real<lower=0> sigmap;          // between family sd in logit(p)
  real fameffphi[nfam];                    // family effects for phi
  real fameffp[nfam];                      // family effects for p
  real yeareffphi[4];                      // year effect on phi
}
transformed parameters {
  real<lower=0,upper=1>p[I,K];             // capture probability
  real<lower=0,upper=1>phi[I,K-1];   // survival probability
```

Stan Code for the CJS Model with Random Effects (file "CJS_swallows-stan")—cont'd

```
    real<lower=0,upper=1>chi[I,K+1];  // probability that an
            // individual is never recaptured after its last capture
    {
      int k;
      for(ii in 1:I){
        for(tt in 1:(K-1)) {
          // linear predictor with random effect for phi:
          // add fixed and random effects here
          phi[ii,tt] <- inv_logit(a[tt] +a1*carez[ii] +
            sigmayearphi * yeareffphi[year[ii]] +
            sigmaphi*fameffphi[family[ii]]);
      }
    }
    for(i in 1:I) {
      // linear predictor with random effect
      // for p: add fixed and random effects here
      p[i,1] <- 1; // first occasion is marking occasion
      for(kk in 2:K)
        p[i,kk] <- inv_logit(b0[year[i]] + b1[year[i]]*agec[kk]+
          sigmap*fameffp[family[i]]);

      // probability that an individual is never recaptured after its
      // last capture
      chi[i,K+1] <- 1.0;
      k <- K;
      while (k > 1) {
        chi[i,k] <- (1 - phi[i,k-1]) + phi[i,k-1] * (1 – p[i,k]) *
          chi[i,k+1];
        k <- k – 1;
      }
      chi[i,1] <- (1 – p[i,1]) * chi[i,2];
      }
    }
  }
  model {
    // priors
    for(j in 1:4){
      b0[j] ~ normal(0, 5);
      b1[j] ~ normal(0, 5);
    }
    for(v in 1:(K-1)){
      a[v]~normal(0,5);
```

Continued

Stan Code for the CJS Model with Random Effects (file "CJS_swallows-stan")—cont'd

```
    }
    a1~normal(0,5);
    sigmaphi ~ student_t(2,0,1);
    sigmayearphi ~ student_t(2,0,1);
    sigmap ~ student_t(2,0,1);

    // random effects
    for(g in 1:nfam) {
      fameffphi[g] ~ normal(0, 1);
      fameffp[g] ~ normal(0,1);
    }
    for(ye in 1:4){
      yeareffphi[ye] ~ normal(0, 1);
    }
    // likelihood
    for (i in 1:I) {
      if (last[i]>0) {
        for (k in 1:last[i]) {
          if(k>1) 1 ~ bernoulli(phi[i, k-1]);
          CH[i,k] ~ bernoulli(p[i,k]);
        }
      }
      increment_log_prob(log(chi[i,last[i]+1]));
    }
  }
```

As for the zero-inflated model in Section 14.2, we could code the model more efficiently, for example, by omitting the `inv_logit` function by using `bernoulli_logit`. But here we use a less efficient code that allows for recognition of the model structure more easily. The model code is saved in the text file "CJS_swallows.stan" and the model is fitted to the data using Stan:

```
library(rstan)
mod <- stan(file = "CJS_swallows.stan", data=datax, chains=4,
            iter=5000)
```

A summary of the results is printed to the R console with the function `print`.

```
print(mod, c("a", "a1", "b0", "b1","sigmayearphi", "sigmaphi",
      "sigmap"))
Inference for Stan model: CJS_swallows_neu.
4 chains, each with iter=5000; warmup=2500; thin=10;
post-warmup draws per chain=250, total post-warmup draws=1000.
```

```
               mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
a[1]           3.94    0.02 0.55  2.81  3.61  3.94  4.29  5.00   633 1.00
a[2]           3.86    0.02 0.57  2.67  3.55  3.87  4.20  4.95   512 1.01
a[3]           3.24    0.02 0.49  2.12  2.97  3.27  3.56  4.09   588 1.01
a[4]           3.27    0.02 0.52  2.17  2.97  3.29  3.58  4.25   634 1.01
a[5]           3.43    0.02 0.54  2.27  3.12  3.44  3.76  4.44   679 1.00
a[6]           3.95    0.02 0.65  2.73  3.56  3.96  4.31  5.31   806 1.00
a[7]           3.47    0.02 0.58  2.13  3.13  3.48  3.85  4.53   554 1.00
a[8]           3.86    0.03 0.70  2.52  3.42  3.82  4.28  5.25   774 1.00
a[9]           3.23    0.02 0.58  1.95  2.88  3.21  3.61  4.36   652 1.00
a[10]          2.26    0.02 0.48  1.14  2.01  2.30  2.55  3.11   577 1.00
a[11]         10.21    0.16 5.10  3.78  6.19  9.27 13.08 22.89   966 1.01
a[12]          2.11    0.02 0.50  0.94  1.83  2.14  2.42  2.95   512 1.01
a[13]          2.24    0.02 0.54  1.01  1.94  2.26  2.56  3.27   687 1.01
a[14]          4.69    0.13 3.73  1.84  2.80  3.49  4.66 17.05   851 1.00
a[15]          4.27    0.13 4.07  1.16  2.25  2.86  4.14 18.28   915 1.00
a[16]          1.29    0.02 0.54  0.14  0.97  1.30  1.65  2.27   604 1.00
a[17]          2.24    0.14 2.96  0.26  1.12  1.61  2.21 10.35   420 1.01
a1             0.72    0.00 0.13  0.46  0.63  0.71  0.81  0.98  1000 1.01
b0[1]          2.18    0.01 0.36  1.47  1.93  2.18  2.42  2.92   890 1.00
b0[2]          1.97    0.01 0.22  1.53  1.82  1.95  2.10  2.40  1000 1.00
b0[3]          3.43    0.01 0.27  2.93  3.24  3.41  3.61  3.97  1000 1.00
b0[4]          3.38    0.01 0.39  2.66  3.13  3.37  3.64  4.15  1000 1.00
b1[1]          0.04    0.00 0.04 −0.03  0.02  0.04  0.07  0.13  1000 1.00
b1[2]         −0.31    0.00 0.03 −0.37 −0.33 −0.31 −0.29 -0.26  1000 1.00
b1[3]         −0.41    0.00 0.04 −0.49 −0.44 −0.41 −0.38 -0.34  1000 1.00
b1[4]         −0.07    0.00 0.05 -0.18 −0.11 −0.07 −0.03  0.03   899 1.01
sigmayearphi   0.70    0.02 0.50  0.13  0.35  0.55  0.89  2.09   767 1.00
sigmaphi       0.48    0.01 0.16  0.11  0.37  0.48  0.59  0.76   878 1.00
sigmap         0.89    0.00 0.14  0.66  0.78  0.87  0.97  1.20   981 1.00
```

```
Samples were drawn using NUTS(diag_e) at Sun Sep 14 13:46:47 2014.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

We first check whether the MCMCs have converged. Then, we assess model fit and, lastly, we draw conclusions. The $\widehat{R}$ seem to be close to 1. Thus there is no indication of nonconvergence. Also, the Monte Carlo error (se_mean) is close to 0 at least for the parameters belonging to the first 10 days, where sample size is high. To graphically assess convergence, the Markov chains can be plotted within the same plot. To plot the whole chains (inclusive of the values discarded due to thinning, alternatively with or without the warm-up phase), use

```
traceplot(mod, "sigmap")     # figure not shown
```

To look at only the saved values to base inferences on, use, for example, the function `historyplot`.

```
historyplot(mod, "sigmap")    # figure not shown
historyplot(mod, "sigmaphi")  # figure not shown
historyplot(mod, "a")         # figure not shown
```

To assess whether the model fits the data, we simulate replicated data from the model (posterior predictive model checking). To do so, we use the 1000 values from the posterior distributions of Φ and p, here called Φ^{*}_{rit} and p^{*}_{rit}, where r is the index for the simulation and the star is added to distinguish these values from the theoretical (unknown) true Φ_{it} values and from the estimated parameter values $\widehat{\Phi}_{it}$. The Φ^{*}_{rit} and p^{*}_{rit} values can be accessed by the function extract.

```
modsims <- extract(mod)
```

First, we define arrays for the replicated observations (y^{rep}_{rit}) and the replicated state variable (z^{rep}_{rit}).

```
nsim <- dim(modsims$phi)[1]  # extract the number of simulations
yrep <- array(dim=c(nsim, datax$I, datax$K))
zrep <- array(dim=c(nsim, datax$I, datax$K))
```

Second, the state and observation at the first capture occasion (day of fledging) is filled with 1s for every individual i and simulation r.

```
yrep[,,1] <- 1    # first occasion is 1 for all individuals
zrep[,,1] <- 1
```

Third, we go through all subsequent capture occasions and simulate states z^{rep}_{rit} based on $z^{rep}_{rit\text{-}1}$ and Φ^{*}_{rit}, and new observations y^{rep}_{rit} based on z^{rep}_{rit} and p^{*}_{rit}.

```
for(j in 1:(datax$K-1)){
  zrep[,,j+1] <- rbinom(nsim*datax$I, size=zrep[,,j],
                        prob=modsims$phi[,,j])
  yrep[,,j+1] <- rbinom(nsim*datax$I, size=zrep[,,j+1],
                        prob=modsims$p[,,j])
}
```

The yrep object contains 1000 new capture histories for the 322 individuals in the data set. All of these data sets reflect possible observations assuming that the structure of the model is true but taking into account the uncertainty in the estimated model parameters.

In the last step, we compare the replicated data sets to the observed data. There are many ways to do this comparison. When the interest is in an overall mean survival probability, we may compare the average time until the last capture between the observed and simulated data; but when the interest is in temporal patterns of survival we may do better by visualizing each individual capture history. As an example here, we assess whether the between-family variance assumed in the model fits the data. To do so, we count the number of observations per individual and calculate the mean, minimum, and maximum of this number per family. We do this first for the observed data.

```
nobspind <- apply(datax$CH, 1, sum)
     # number of observations per individual
mpfam <- tapply(nobspind, datax$family, mean)
minpfam <- tapply(nobspind, datax$family, min)
maxpfam <- tapply(nobspind, datax$family, max)
```

Then, we do the same for the 1000 replicated data sets.

```
repnpind <- apply(yrep, c(1,2), sum)
repmpfam <- matrix(nrow=nsim, ncol=datax$nfam)
repminpfam <- matrix(nrow=nsim, ncol=datax$nfam)
repmaxpfam <- matrix(nrow=nsim, ncol=datax$nfam)

for(f in 1:nsim) {
  repmpfam[f,] <- tapply(repnpind[f,], datax$family, mean)
  repminpfam[f,] <- tapply(repnpind[f,], datax$family, min)
  repmaxpfam[f,] <- tapply(repnpind[f,], datax$family, max)
}
```

The observed data are then compared with the simulated data graphically. In Figure 14.6 the average number of observations is given per family and ordered increasingly (bold black line). In addition the familywise minimum and maximum number of observations per individual are given (thin black lines). The blue lines give the same statistics as for the observed data, but averaged over the 1000 simulated data sets. We see that the replicated data from the model resemble the observed data quite well, particularly for families with an average number of observations. Maybe, for families with low numbers of observations, the model predicts slightly higher numbers of observations whereas for families with many observations, the model predicts slightly lower numbers. This may indicate that the model slightly underestimates the between-family variance either in recapture probability or in survival probability. This predictive model checking is not exhaustive. See Chapter 10 for more on predictive model checking.

After we have decided that our model may adequately describe the process that generated the data, the results can be extracted from the modsims object (obtained by the function `extract`). The estimates for daily survival probabilities are directly calculated from the parameter α_0 (Figure 14-7).

```
ests <- plogis(apply(modsims$a, 2, mean))
ests.lwr <- plogis(apply(modsims$a, 2, quantile, prob=0.025))
ests.upr <- plogis(apply(modsims$a, 2, quantile, prob=0.975))
```

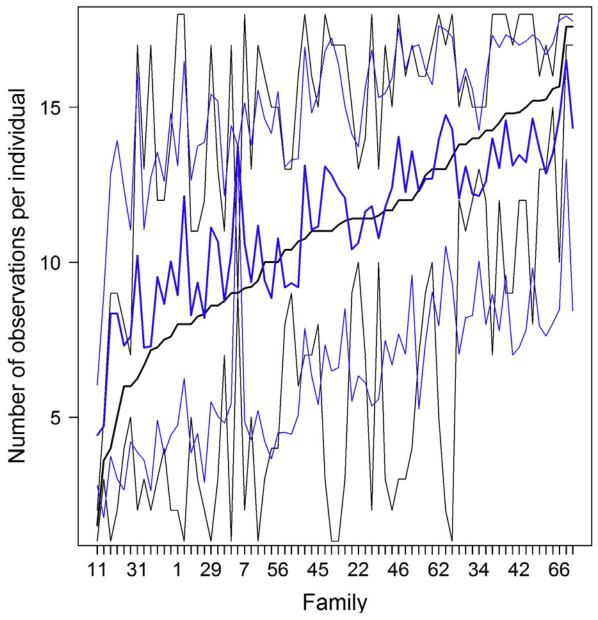

FIGURE 14-6 Number of times an individual has been observed during the study. Shown is the average per family (bold black line) in ascending order. For every family, the observed minimum and the maximum is given in thin black lines. The blue lines give the same numbers averaged over 1000 simulated data sets (i.e., mean of the posterior predictive distribution for the specific statistic).

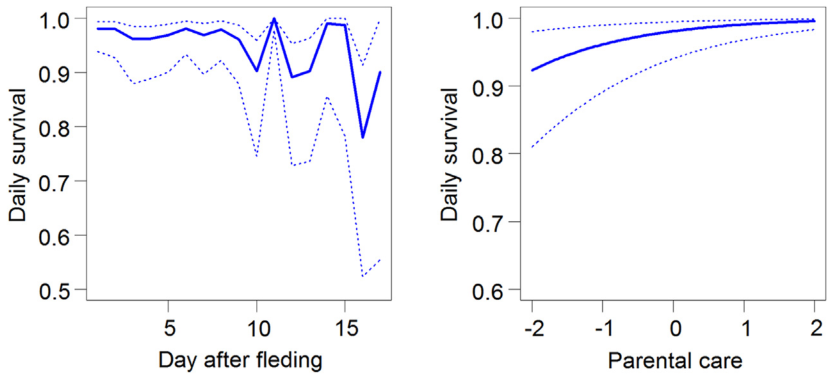

FIGURE 14-7 Estimated daily survival probability for every day after fledging (left) and effect of parental care on daily survival of the fledglings (right). Bold lines = mean of the posterior distribution; dotted lines = 2.5% and 97.5% quantiles of the posterior distribution.

To visualize the effect of parental care on survival, we average the intercept over the first 12 days and extract the slope for carez from the modsim object. Then we obtain the fitted values in the same way as we calculate the regression line in a logistic regression.

```
ma <- apply(modsims$a[,1:12], 1, mean) # mean over the first 12 days
newdat <- data.frame(carez=seq(-2, 2, length=100))
b <- c(mean(ma), mean(modsims$a1))     # model coefficients
Xmat <- model.matrix(~carez, data=newdat)
newdat$fit <- plogis(Xmat %*% b)
nsim <- nrow(modsims$a)
fitmat <- matrix(ncol=nsim, nrow=nrow(newdat))
for(i in 1:nsim) fitmat[,i] <- plogis(Xmat %*%
                                     c(ma[i], modsims$a1[1]))
newdat$lwr <- apply(fitmat, 1, quantile, prob=0.025)
newdat$upr <- apply(fitmat, 1, quantile, prob=0.975)
```

We see that duration of parental care has a positive effect on daily survival of the fledglings (Figure 14-7). The posterior probability that parental care has a positive effect on fledgling survival is the proportion of positive values among the simulated values from the posterior distribution of the parameter α_1, which is larger than 0.999.

```
mean(modsims$a1>0)
[1] 1
```

All 1000 simulated values from the posterior distribution of α_1 are positive. Because we do not know whether the 1001st simulation would have produced a negative number (which would have led to a posterior probability of 0.999001), we can report our results as $\Pr(\alpha_1 > 0|y) > 0.999$ and conclude that we found a well supported positive effect of parental care on fledgling survival.

FURTHER READING

An informative and well-written chapter about multinomial models in WinBUGS is given by Ntzoufras (2009).

In the book about analysis of count data, Cameron and Trivedi (2013) give an overview of models for data with excess zeros. Zuur et al. (2012) is an applied textbook on zero-inflated models for ecologists. Poulsen et al. (2011) apply a zero-inflated Poisson model to disentangle different effects on animal communities. Lecomte et al. (2013) use zero-inflated models in spatial analyses of biomass data.

Royle and Dorazio (2008) introduce a variety of different ecological models that explicitly model the observation process conditional on the biological process. In their recent book, they expand these models to allow for studying the spatial movements of animals (Royle et al., 2014). Link and Barker (2010) give an introduction to the Bayesian theory with applications in ecology.

The territory occupancy model is discussed in Roth and Amrhein (2010).

Besides the original literature on CSJ models (Cormack, 1964; Jolly, 1965; Seber, 1965), a multitude of authors have discussed variations of these models. A seminal review is Lebreton et al. (1992). Practical introductions for biologists are given by White and Burnham (1999) (software Program MARK), Royle and Dorazio (2008) (R and WinBUGS), and Kéry and Schaub (2012) (WinBUGS). Theoretical background is found in King et al. (2010) and King (2014).