CHAPTER

7

Estimation of Survival from Capture–Recapture Data Using the Cormack–Jolly–Seber Model

OUTLINE

DOI: 10.1016/B978-0-12-387020-9.00007-9

7.1 INTRODUCTION

The preceding chapters dealt with the modeling and estimation of population size and with the simplest summary of population dynamics: population trend. The following chapters focus on one of the main components of population dynamics: survival probability. Survival probability is a key demographic parameter and can have a strong impact on population dynamics (Clobert and Lebreton, 1991; Saether and Bakke, 2000). Typically, interest focuses on estimation (what is the survival in that population?) as well as on modeling, for example, to test whether survival changes with age or differs between groups of individuals or regions, and to estimate how strongly it varies over time or what proportion of temporal variability can be explained by an external covariate such as weather.

In principle, survival estimation is fairly simple—we just have to count the number of individuals alive at a given time t (C_t), and keep track of how many of them die ($D_{\Delta t}$) during the period Δt for which we wish to estimate survival. Sometimes, it may be easier to count the number of the C_t that are still alive at $t + \Delta t$, that is, the number that survived the period Δt ($L_{\Delta t}$). Then survival probability s_t is

$$s_t = \frac{C_t - D_{\Delta t}}{C_t} = \frac{L_{\Delta t}}{C_t}$$

These numbers may easily be obtained in humans, but they are difficult to get in animal or plant populations. The reason for that is because the detection of individuals is usually far from perfect, so when an individual is not seen, we don't know whether it is dead or still alive. Therefore, such simple calculations cannot often be used, and we need to account for the

observation process in our inferences about survival (an exception being data on individuals with radio tags, White and Garrott, 1990). From the number of individuals recorded at t (C_t), we typically detect only a fraction of those still alive at time $t + \Delta t$, which is $p^*L_{\Delta t}$: p is the recapture or resighting probability (depending on the context or study design), which needs to be estimated in order to obtain unbiased estimates of survival. Estimating p becomes possible if we extend the recapture study to at least one further time step ($t + 2\Delta t$). We may then have individuals that are known to have survived until $t + 2\Delta t$, but which have not been seen at time $t + \Delta t$. Intuitively, it is clear that the proportion of these individuals provides information on p.

The most common statistical method to jointly estimate recapture and survival probabilities in animal and plant populations is a class of open population capture–recapture models to which the Cormack–Jolly–Seber (CJS) model belongs (Cormack, 1964; Jolly, 1965; Seber, 1965). Re-encounters may be obtained by different methods (physical capture, sightings, genetic tracking), but the key is that individuals are identified without error. That is, we only have false negatives, but no false positives. The frequentist analysis of the CJS model is described in detail in Lebreton et al. (1992) and Williams et al. (2002). Descriptions and examples of the Bayesian analysis of the CJS model can be found in an increasing number of articles and books (Brooks et al., 2000a; McCarthy and Masters, 2005; Gimenez et al., 2007; McCarthy, 2007; Zheng et al., 2007; Royle, 2008; Royle and Dorazio, 2008; Schofield et al., 2009; Gimenez et al., 2009a; King et al., 2010).

The CJS model can be fitted using either a multinomial (Lebreton et al., 1992) or a state-space likelihood (Gimenez et al., 2007; Royle, 2008). Because these two likelihoods are just different ways of describing what is essentially the same model, they are based on the same sampling design and the same underlying model assumptions. The sampling design is as follows: A random sample of individuals from the study population is captured, all are marked individually, and released into the population again. This is repeated several times. The length of the time intervals between repeated capture occasions depends on the research question, as well as on the life history and population dynamics of the study organism, and capture should be instantaneous or over a short time period. Some marked individuals will be re-encountered, and thus, we obtain capture–recapture data that can be summarized in individual capture-histories.

The CJS model makes a number of assumptions and, as usual, their violation may bias parameter estimators. Some assumptions must be met at the design stage of a study. Tags or other marks must not be lost, otherwise survival is underestimated. If mark loss is suspected, double marking and corresponding model adaptation that account for mark loss are necessary to get unbiased estimates of survival (e.g., Smout et al., 2011).

Ideally, capture should be instantaneous, otherwise the interval between capture occasions may differ among individuals and, consequently, there will be individual heterogeneity in survival. However, simulation studies have suggested that the violation of this last assumption does often not have a strong effect on parameters estimates (Hargrove and Borland, 1994). The CJS model also assumes that the identity of the individuals is always recorded without errors. If this assumption is violated, bias can go in either direction, and there is no means to correct for it. Finally, captured and recaptured individuals are regarded as a random sample from the study population. This sounds easy, but in practice, it can be difficult to achieve. For example, in studies on birds using nest boxes, it is quite typical that adults are only captured after the young have hatched because they are likely to abandon their brood if they are disturbed at an early stage. This results in a sample that is biased toward successfully breeding adults, which may or may not be a random sample from all adults in the population.

Further assumptions of the CJS model cannot be violated or fulfilled by the design of the study, rather they are a consequence of how the model is specified. The basic model assumes that each individual within an age class or group has the same survival and recapture probability. Goodness-of-fit tests help to identify severe violation of these assumptions (e.g., trap-response: Pradel, 1993; transients: Pradel et al., 1997), and modifications to the model allow to account for these violations. Individuals must behave independently from each other in terms of survival and recapture. This may not be the case if members of the same family are included in a sample and especially if they remain in family groups. Violation of this assumption is like pseudo-replication (Hurlbert, 1984). The degree of nonindependence leading to overdispersion can be estimated and the standard errors of the estimates as well as AIC-based model selection can be adjusted accordingly (Anderson et al., 1994).

With CJS models, we estimate recapture probability (p_t; the probability of catching/resighting a marked individual at t that is alive and in the sampling population at t) and apparent (also called "local") survival probability (ϕ_t; the probability that an individual that is alive and in the population at t is still alive and in the population at time $t+1$). Mortality and permanent emigration are confounded, and therefore apparent survival is always lower than true survival whenever permanent emigration is not zero. The difference between apparent and true survival is a matter of study design. Generally, the larger a study area the closer the match between apparent and true survival because dispersing individuals have a higher probability to remain in the study area (Marshall et al., 2004). Throughout this chapter, we will often just write "survival" for ease of presentation—but it is important to remember that survival in the CJS model always refers to a study area.

In this chapter, we introduce the CJS model and illustrate how it is fitted using the state-space (Sections 7.2–7.8) and the multinomial likelihood (Sections 7.9–7.11). We highlight advantages and disadvantages of each approach. Moreover, we will repeatedly use the generalized linear (mixed) model (GLM and GLMM) formulations to describe structure in the parameters. This allows the modeling of individual and temporal effects, both of which can either be categorical or continuous, as well as fixed or random. We will also see how correlations among parameters can be modeled through correlated random effects, using a multivariate normal distribution. In most examples, we focus on the modeling of survival, yet, clearly, similar modeling can be conducted for the recapture probability, and all these GLM formulations can also be used in other capture–recapture types of models starting with Chapter 6. Finally, in Section 7.10, we introduce posterior predictive model checking (see Gelman et al., 1996, 2004; Kéry, 2010). This provides a very general framework for the assessment of goodness-of-fit of a model to a data set.

7.2 THE CJS MODEL AS A STATE-SPACE MODEL

The state-space formulation of the CJS model has been introduced by Gimenez et al. (2007) and Royle (2008). Let us assume an individual marked at time t. It may survive until time $t+1$ with probability ϕ_t. Conceptually, we can imagine the individual tossing a coin to determine whether it survives (with probability ϕ_t) or dies (with probability $1-\phi_t$). Given that the individual is still alive at time $t+1$, it may again survive until $t+2$ with probability ϕ_{t+1}. This process is continued until the individual is either dead or the study ends. Clearly, once an individual is dead, its fate is no longer stochastic, and it will remain dead with probability 1. This is the description of the state process, that is, of the states (alive, dead) of an individual over time. We would like to know survival, which requires knowledge of these states of the individuals. Yet, we typically do not have complete information about the true states. A marked individual that is alive at occasion t may be recaptured (or more generally re-encountered) with probability p_t. Again, we can imagine that a coin is tossed, determining whether the individual is recaptured (with probability p_t) or not (with probability $1-p_t$). Once an individual is dead, it cannot be recaptured anymore. This is the description of the observation process, which is conditional on the state process, and thus there is a hierarchal structure in the state-space model. In Fig. 7.1, the two processes are shown graphically.

The data observed in a capture–recapture study can be summarized in a capture-history matrix ($\mathbf{y}$), which has dimension $I \times T$, where I is the total number of marked individuals, and T is the number of capture

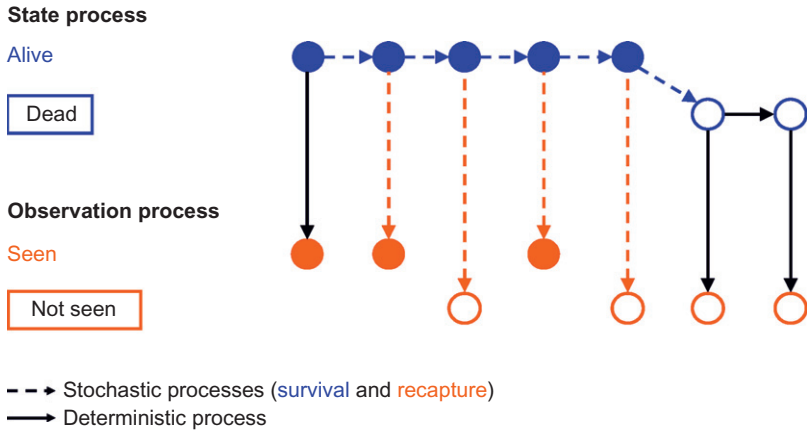

FIGURE 7.1 Example of the state and observation process of a marked individual over time for the CJS model. The sequence of true states in this individual is $z = [1, 1, 1, 1, 1, 0, 0]$, and the observed capture-history is $y = [1, 1, 0, 1, 0, 0, 0]$.

occasions. The matrix entries are either a 1 or a 0. A 1 at position i, t indicates that individual i was captured at occasion t, meaning that it was alive for sure; a 0 at position i, t shows that individual i was not captured at t, meaning that it was either dead, or alive but not caught, or not yet marked.

To estimate survival from such data, we define the latent variable $z_{i,t}$, which takes value 1 if individual i is alive at time t, and value 0 if it is dead. Thus, $z_{i,t}$ defines the true state of individual i at time t. We also define vector f_i, which denotes the occasion at which individual i is first captured (i.e., marked) because only events after first capture are modeled in the CJS model. The state of individual i at first capture ($z_{i,fi}$) is 1 with probability 1, as the individual is alive for certain. The states on subsequent occasions are modeled as Bernoulli trials. Conditional on being alive at occasion t, individual i may survive until occasion $t + 1$ with probability $\phi_{i,t}$ ($t = 1, \ldots, T - 1$). The following two equations define the state process:

$$z_{i,f_i} = 1$$

$$z_{i,t+1} \,|\, z_{i,t} \sim \text{Bernoulli}(z_{i,t}\phi_{i,t}).$$

The Bernoulli success parameter is composed of the product of survival and the state variable z. The inclusion of z ensures that a dead individual ($z = 0$) remains dead and has no further impact on the estimation of survival.

If individual i is alive at occasion t, it may be recaptured with probability $p_{i,t}$ ($t = 2, \ldots, T$). This can again be modeled as the realization of a

Bernoulli trial with success probability $p_{i,t}$. The following equation defines the observation process:

$$y_{i,t} \,|\, z_{i,t} \sim \text{Bernoulli}(z_{i,t} p_{i,t}).$$

The inclusion of the latent variable z in the Bernoulli trial ensures that dead individuals cannot be encountered. The state and the observation process are both defined for $t \geq f_i$. We repeat that the initial capture process is not modeled in the CJS model (see Fig. 7.1) because the initial observation at the time of capture does not contain any information about survival. In contrast, capture-histories at and before initial capture contain information about recruitment, which is a target of estimation of the Jolly–Seber models (see Chapter 10). Because initial capture is not modeled in the CJS model, we say that we condition on first capture.

The implementation of the CJS model in WinBUGS is straightforward. The most general likelihood is based on the above-mentioned three equations and contains different survival and recapture probabilities for each individual at each capture occasion. However, the parameters of this saturated model are not separately estimable (see Section 7.9 for more on this topic), and we need to introduce constraints. These constraints define the structure of the model fitted and may be imposed either along the time or the individual axis of the capture-history matrix, or along both (see Section 6.2). Thus, whatever model we fit, we do not need to change the likelihood, which describes the basic structure of the model, but just these constraints and the corresponding priors. This may not result in code that is the most efficient in terms of computing time and the easiest to read for a beginner. However, with a little practice, it will be seen to be an efficient way of fitting a wide array of models.

7.3 MODELS WITH CONSTANT PARAMETERS

We start with a very simple model, in which survival and recapture, respectively, are identical for all individuals at all occasions. Thus, we impose constraints along both the time and the individual axis of the capture-history matrix. We first simulate the data, and then analyze them. The function to simulate capture–recapture data (`simul.cjs`) is very general and works for all examples in this chapter. We choose the number of individuals released at each occasion. The function then evaluates for each released individual whether it survives, and if so, whether it is recaptured, by two Bernoulli trials governed by individual- and time-specific survival and recapture probabilities that we also provide as input (matrices `PHI` and `P`). Thus, the data-generating function works analogous to the analyzing model.

FIGURE 7.2 Pair of little owls (*Athene noctua*) (Photograph by H. Sylvain).

In the simulation, we will mimic a study on little owls (Fig. 7.2), a small owl species living in semi-open habitats such as orchards, where it likes to occupy nest boxes. Nest boxes in a study area are checked in May in six study years, and breeding adults are ringed. In each of the six study years, 50 unmarked (new) adults are caught, along with a variable number of individuals that are already marked. Survival of adult little owls is typically around 0.65 (Schaub et al., 2006), and we assume a recapture probability of 0.4. The following R code simulates a matrix with capture-histories. We do not consider individuals first captured on the last occasion, because they do not provide information about survival and recapture.

```
# Define parameter values
n.occasions <- 6                      # Number of capture occasions
marked <- rep(50, n.occasions-1)      # Annual number of newly marked
                                        individuals
phi <- rep(0.65, n.occasions-1)
p <- rep(0.4, n.occasions-1)

# Define matrices with survival and recapture probabilities
PHI <- matrix(phi, ncol = n.occasions-1, nrow = sum(marked))
P <- matrix(p, ncol = n.occasions-1, nrow = sum(marked))

# Define function to simulate a capture-history (CH) matrix
simul.cjs <- function(PHI, P, marked){
   n.occasions <- dim(PHI)[2] + 1
```

```
    CH <- matrix(0, ncol = n.occasions, nrow = sum(marked))
    # Define a vector with the occasion of marking
    mark.occ <- rep(1:length(marked), marked[1:length(marked)])
    # Fill the CH matrix
    for (i in 1:sum(marked)){
        CH[i, mark.occ[i]] <- 1       # Write an 1 at the release occasion
        if (mark.occ[i]==n.occasions) next
        for (t in (mark.occ[i]+1):n.occasions){
            # Bernoulli trial: does individual survive occasion?
            sur <- rbinom(1, 1, PHI[i,t-1])
            if (sur==0) break         # If dead, move to next individual
            # Bernoulli trial: is individual recaptured?
            rp <- rbinom(1, 1, P[i,t-1])
            if (rp==1) CH[i,t] <- 1
            } #t
        } #i
    return(CH)
    }

# Execute function
CH <- simul.cjs(PHI, P, marked)
```

Next, we need to create vector *f*, which contains the occasion at which each individual is marked.

```
# Create vector with occasion of marking
get.first <- function(x) min(which(x!=0))
f <- apply(CH, 1, get.first)
```

Finally, we write the BUGS code for a constant model. The two linear models applied are $\phi_{i,t} = \overline{\phi}$ and $p_{i,t} = \overline{p}$. These are in fact the linear predictors (see Chapter 3), but here we call them constraints because we reduce the dimensions of the $\phi_{i,t}$ and $p_{i,t}$, that is, we constrain them. We do not include covariates or random effects, so there is no need for a transformation, and the identity link is applied. The uniform priors ensure that the parameter estimates are in the interval [0, 1]. The specification of noninformative priors is easy because a uniform (U(0, 1)) or a beta distribution (beta(1,1)) can be used. Note that the time indexing in the "Likelihood" part is slightly different to that used in the formulas (see Section 7.2). It avoids the use of separate loops for the state and the observation process.

```
# Specify model in BUGS language
sink("cjs-c-c.bug")
cat("
model {

# Priors and constraints
for (i in 1:nind){
    for (t in f[i]:(n.occasions-1)){
        phi[i,t] <- mean.phi
```

```
        p[i,t] <- mean.p
        } #t
    } #i

mean.phi ~ dunif(0, 1)          # Prior for mean survival
mean.p ~ dunif(0, 1)            # Prior for mean recapture

# Likelihood
for (i in 1:nind){
    # Define latent state at first capture
    z[i,f[i]] <- 1
    for (t in (f[i]+1):n.occasions){
        # State process
        z[i,t] ~ dbern(mu1[i,t])
        mu1[i,t] <- phi[i,t-1] * z[i,t-1]
        # Observation process
        y[i,t] ~ dbern(mu2[i,t])
        mu2[i,t] <- p[i,t-1] * z[i,t]
        } #t
    } #i
}
",fill = TRUE)
sink()

# Bundle data
bugs.data <- list(y = CH, f = f, nind = dim(CH)[1], n.occasions =
   dim(CH)[2])
```

Initial values should be given for the two structural parameters and for the latent variable z. The easiest way for the latter is just to use the observed capture-histories. We have to make sure that initial values for z are provided only after initial capture. The function below creates the required initial values based on the observed capture-histories and the vector with the occasion of first capture.

```
# Function to create a matrix of initial values for latent state z
ch.init <- function(ch, f){
    for (i in 1:dim(ch)[1]){ch[i,1:f[i]] <- NA}
    return(ch)
    }

# Initial values
inits <- function(){list(z = ch.init(CH, f), mean.phi = runif(1, 0, 1),
   mean.p = runif(1, 0, 1))}

# Parameters monitored
parameters <- c("mean.phi", "mean.p")

# MCMC settings
ni <- 10000
nt <- 6
nb <- 5000
nc <- 3
```

```
# Call WinBUGS from R (BRT 1 min)
cjs.c.c <- bugs(bugs.data, inits, parameters, "cjs-c-c.bug", n.chains =
   nc, n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE,
   bugs.directory = bugs.dir, working.directory = getwd())
```

The model does not take a long time to run and convergence is reached after just 5000 iterations. The estimates are very close to the values used for the simulations.

```
# Summarize posteriors
print(cjs.c.c, digits = 3)
```

```
          mean    sd  2.5%   25%   50%   75% 97.5%  Rhat  n.eff
mean.phi 0.679 0.043 0.601 0.649 0.677 0.707 0.767 1.003    980
mean.p   0.370 0.044 0.286 0.340 0.369 0.400 0.458 1.003   1700
```

Sometimes not all individuals recaptured are released again, for instance, when an individual dies at capture. For these individuals, we know that they have survived from initial capture until the last capture; afterward they are not in the sample anymore. This is easy to model and just requires that we define a vector h which for each individual contains the occasion after which it is not released. For individuals that stay in the sample until the end of the study, the element of vector h is just the last occasion of the study. Then, vector h must become an element of the input data and the loop in the likelihood needs to be changed from `for (t in (f[i]+1):n.occasions){ ...` to `for (t in (f[i]+1):h[i]){...`

7.3.1 Inclusion of Information about Latent State Variable

Written as state-space models, CJS models can take a long time to run because there is a loop over all individuals and occasions. The latent state variable z needs to be updated (estimated) at each MCMC iteration. So far we have treated z as if we had no information about it. The only information that we included are the observed capture-histories (Y), but they are related to z only through the observation process in the state-space model. Therefore, all elements of z (i.e., for all individuals after first capture) must be estimated, even when some of them are known.

To improve computation speed and convergence, we can add what we know about the latent state z, namely, whenever we observe a marked individual we know its latent state is $z = 1$. In addition, we know that $z = 1$ for all occasions between the first and the last observation of an individual, even if it was not seen at all occasions. To include this information in the model (i.e. to prevent estimation of what is not an unknown quantity), we create a matrix that has a value of 1 at all occasions where we know individuals were alive, and NAs elsewhere. The CJS model is conditional on first capture, so the latent state is only defined after first

capture, and thus at all first captures, we need NAs as well. The following function creates the required matrix.

```
# Function to create a matrix with information about known latent state z
known.state.cjs <- function(ch){
    state <- ch
    for (i in 1:dim(ch)[1]){
        n1 <- min(which(ch[i,]==1))
        n2 <- max(which(ch[i,]==1))
        state[i,n1:n2] <- 1
        state[i,n1] <- NA
        }
    state[state==0] <- NA
    return(state)
    }
```

This information about z is then given as data as well.

```
# Bundle data
bugs.data <- list(y = CH, f = f, nind = dim(CH)[1], n.occasions = dim(CH)[2],
   z = known.state.cjs(CH))
```

The initial values for z now also require some changes: we should not give initial values for those elements of z whose value is specified in the data; they get an NA.

```
# Function to create a matrix of initial values for latent state z
cjs.init.z <- function(ch,f){
    for (i in 1:dim(ch)[1]){
        if (sum(ch[i,])==1) next
        n2 <- max(which(ch[i,]==1))
        ch[i,f[i]:n2] <- NA
        }
    for (i in 1:dim(ch)[1]){
    ch[i,1:f[i]] <- NA
    }
    return(ch)
    }
```

Now, we give initial values for all the quantities to be estimated and run the model:

```
# Initial values
inits <- function(){list(z = cjs.init.z(CH, f), mean.phi = runif(1, 0, 1),
   mean.p = runif(1, 0, 1))}

# Parameters monitored
parameters <- c("mean.phi", "mean.p")

# MCMC settings
ni <- 10000
nt <- 6
```

```
nb <- 5000
nc <- 3

# Call WinBUGS from R (BRT <1 min)
cjs.c.c <- bugs(bugs.data, inits, parameters, "cjs-c-c.bug", n.chains =
   nc, n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE,
   bugs.directory = bugs.dir, working.directory = getwd())

# Summarize posteriors
print(cjs.c.c, digits = 3)

          mean    sd  2.5%   25%   50%   75% 97.5%  Rhat  n.eff
mean.phi 0.675 0.040 0.599 0.648 0.675 0.702 0.754 1.003   2500
mean.p   0.372 0.043 0.294 0.342 0.371 0.399 0.461 1.003   2500
```

The model now runs faster. The difference in run time in this simple case is slight, but time savings can be substantial with more complex models and larger data sets. Therefore, we recommend providing all available information about the latent state in the data. In the following sections of this chapter, we will always, and in most other chapters often, do this. Note that the inclusion of the information about the latent state z has nothing to do with the use of an informative prior, we simply avoid estimation of known quantities to speed up computation.

7.4 MODELS WITH TIME-VARIATION

So far we have fitted the simplest possible model in the CJS family. It assumes that survival and recapture probabilities remain constant over time and are identical for all individuals. In practice, we typically want to relax these strict assumptions. We also may have an interest in fitting models that combine time and individual effects and modeling these effects as additive or interactive. We next consider models with temporal variation, that is, we model the column dimension of the capture-history matrix and constrain the row dimension to be constant (all individuals are treated as identical).

The variation of survival probability from one year to another often has a strong impact on the dynamics of a population. If survival varies much from year to year (i.e., temporal variability is large), population size changes more than when survival probability changes only little over time, all other demographic processes being equal. Thus, there is an interest in measuring temporal variation. Moreover, the annual fluctuations of survival or recapture may be caused by environmental factors that we may have an interest in identifying.

The models to study temporal effects of survival or recapture assume either fixed or random temporal effects, as well as the relationship between focal parameters and temporally varying covariates (e.g., weather).

The fixed-effect time model assumes the parameters to be different at each occasion and independent of each other. This approach is used if there is interest in estimates from particular occasions. By contrast, the model that considers time to be a random effect assumes that time effects are drawn from a statistical distribution, whose parameters we aim to estimate; typically, we will use a normal distribution and estimate a mean and a variance. Therefore, annual estimates are no longer independent from one another. Interest is then not so much in the individual annual effects, but more in an estimation of the mean and the variance of the annual estimates. Fixed- and random-effects models are easily fit within the framework that we have set up (see Chapter 4). The likelihood part of the BUGS code does not need any change at all: all required modifications take place in the "Priors and constraints" section of the BUGS code. In the examples that follow, we will usually model effects on survival only, but of course similar models can be adopted for recapture, too, and any combinations are possible, for example, survival with random time effects and recapture with fixed time effects.

7.4.1 Fixed Time Effects

We now assume that survival and recapture vary independently over time, that is, we regard time as a fixed-effects factor. To implement this model, we impose the following constraints: $\phi_{i,t} = \alpha_t$ and $p_{i,t} = \beta_t$, where α_t and β_t are the time-specific survival and recapture probabilities, respectively. Here is the part of the BUGS model specification that needs to be changed.

```
# Priors and constraints
for (i in 1:nind){
    for (t in f[i]:(n.occasions-1)){
        phi[i,t] <- alpha[t]
        p[i,t] <- beta[t]
        } #t
    } #i
for (t in 1:n.occasions-1)){
    alpha[t] ~ dunif(0, 1)          # Priors for time-spec. survival
    beta[t] ~ dunif(0, 1)           # Priors for time-spec. recapture
    }
```

7.4.2 Random Time Effects

The model just shown treats time as a fixed-effects factor; for every occasion, an independent effect is estimated. To assess the temporal variability, we cannot simply take these fixed-effects estimates and calculate their variance. By doing so, we would ignore the fact that these values

are estimates that have an unknown associated error. Thus, we would assume that there is no sampling variance, and this can hardly ever be true (see, e.g., Gould and Nichols, 1998). However, when treating time as a random-effects factor, we can separate sampling (i.e., variance within years) from process variance (i.e., variance between years), exactly as we did in the state-space models in Chapter 5. We model survival or recapture probabilities on the logit scale as a realization of a random process described by a normal distribution with mean μ and variance σ^2. The logit link function ensures that the estimated probabilities remain within the interval between 0 and 1:

$$\text{logit}(\phi_{i,t}) = \mu + \varepsilon_t$$
$$\varepsilon_t \sim \text{Normal}(0, \sigma^2).$$

ε_t is the deviation from the overall mean survival probability; thus it is a "temporal residual". The temporal variance (σ^2) is on the logit scale; thus, it is the temporal variance of the logit survival. Sometimes, one needs an estimate on the probability scale, for instance, when the temporal variance should be compared with the variance of another demographic rate to decide which parameter is more variable over time. A back-transformation is possible by applying the delta method (Powell, 2007). We use

$$\sigma_\theta^2 \cong \sigma^2 \theta^2 (1-\theta)^2,$$

where $\theta = \frac{\exp(\mu)}{1+\exp(\mu)}$ and σ_θ^2 is the variance on the back-transformed scale. It is easy to estimate this quantity directly in BUGS.

To illustrate the approach, we simulate data and analyze them. In the little owl example, we assume a mean survival probability of females of 0.65 and temporal variance of 1 on the logit scale. Reasonable estimates of the temporal variance require a large number of years (>10; Burnham and White, 2002). Here, we simulate data over 20 years.

```
# Define parameter values
n.occasions <- 20                    # Number of capture occasions
marked <- rep(30, n.occasions-1)     # Annual number of newly marked
                                       individuals
mean.phi <- 0.65
var.phi <- 1                         # Temporal variance of survival
p <- rep(0.4, n.occasions-1)

# Determine annual survival probabilities
logit.phi <- rnorm(n.occasions-1, qlogis(mean.phi), var.phi^0.5)
phi <- plogis(logit.phi)

# Define matrices with survival and recapture probabilities
PHI <- matrix(phi, ncol = n.occasions-1, nrow = sum(marked), byrow = TRUE)
P <- matrix(p, ncol = n.occasions-1, nrow = sum(marked))
```

```
# Simulate capture-histories
CH <- simul.cjs(PHI, P, marked)

# Create vector with occasion of marking
get.first <- function(x) min(which(x!=0))
f <- apply(CH, 1, get.first)
```

In the BUGS model description, we only alter parts in the "Priors and constraints" sections; no change is required in the likelihood part. In particular, we have to implement the random-effects formulation (formula above). The prior choices for μ and for σ^2 need some thought. Because μ is the mean survival on the logit scale, a noninformative prior on the logit scale would be a normal distribution with a wide variance. Yet, this prior will not be noninformative on the probability scale. In the code below, we provide two options: first, a normal distribution with a wide variance for μ, and second, a uniform distribution for $\text{logit}^{-1}(\mu)$, which is noninformative on the probability scale but informative on the logit scale. A prior is also needed for σ^2. Following Gelman (2006), we use a uniform distribution for the standard deviation because this induces little information. We will fit the same model under which we generated the data, that is, model $\phi_{\underline{t}}$, p., where by the underlined index for time, we denote random time effects.

```
# Specify model in BUGS language
sink("cjs-temp-raneff.bug")
cat("
model {

# Priors and constraints
for (i in 1:nind){
    for (t in f[i]:(n.occasions-1)){
        logit(phi[i,t]) <- mu + epsilon[t]
        p[i,t] <- mean.p
        } #t
    } #i
for (t in 1:(n.occasions-1)){
    epsilon[t] ~ dnorm(0, tau)
    }

#mu ~ dnorm(0, 0.001)                  # Prior for logit of mean survival
#mean.phi <- 1 / (1+exp(-mu))          # Logit transformation
mean.phi ~ dunif(0, 1)                # Prior for mean survival
mu <- log(mean.phi / (1-mean.phi))    # Logit transformation
sigma ~ dunif(0, 10)                  # Prior for standard deviation
tau <- pow(sigma, -2)
sigma2 <- pow(sigma, 2)               # Temporal variance
mean.p ~ dunif(0, 1)                  # Prior for mean recapture

# Likelihood
for (i in 1:nind){
    # Define latent state at first capture
    z[i,f[i]] <- 1
    for (t in (f[i]+1):n.occasions){
```

```
        # State process
        z[i,t] ~ dbern(mu1[i,t])
        mu1[i,t] <- phi[i,t-1] * z[i,t-1]
        # Observation process
        y[i,t] ~ dbern(mu2[i,t])
        mu2[i,t] <- p[i,t-1] * z[i,t]
        } #t
    } #i
}
",fill = TRUE)
sink()

# Bundle data
bugs.data <- list(y = CH, f = f, nind = dim(CH)[1], n.occasions = dim(CH)[2],
   z = known.state.cjs(CH))

# Initial values
inits <- function(){list(z = cjs.init.z(CH, f), mean.phi = runif(1, 0, 1),
   sigma = runif(1, 0, 10), mean.p = runif(1, 0, 1))}

# Parameters monitored
parameters <- c("mean.phi", "mean.p", "sigma2")

# MCMC settings
ni <- 10000
nt <- 6
nb <- 5000
nc <- 3

# Call WinBUGS from R (BRT 17 min)
cjs.ran <- bugs(bugs.data, inits, parameters, "cjs-temp-raneff.bug",
   n.chains = nc, n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE,
   bugs.directory = bugs.dir, working.directory = getwd())
```

The chains evolve slowly and convergence is not achieved swiftly. This can be improved if a smaller range is chosen for the prior for the standard deviation of the temporal variance. Here, we used a uniform prior in the interval between 0 and 10, thus the variance could take values between 0 and 100. Had we chosen a higher upper bound, the estimate would probability not change (recall we simulated the data with a variance of 1), but the chains would converge even more slowly. Computation for this model is more efficient for the multinomial formulation of the model (see Section 7.10).

```
# Summarize posteriors
print(cjs.ran, digits = 3)

          mean    sd  2.5%   25%   50%   75% 97.5%  Rhat  n.eff
mean.phi 0.634 0.073 0.488 0.590 0.634 0.678 0.787 1.017    120
mean.p   0.394 0.024 0.350 0.378 0.394 0.411 0.441 1.001   2000
sigma2   1.700 1.152 0.548 1.006 1.402 2.005 4.687 1.006    440

# Produce histogram
hist(cjs.ran$sims.list$sigma2, col = "gray", nclass = 35, las = 1,
   xlab = expression(sigma^2), main = "")
abline(v = var.phi, col = "red", lwd = 2)
```

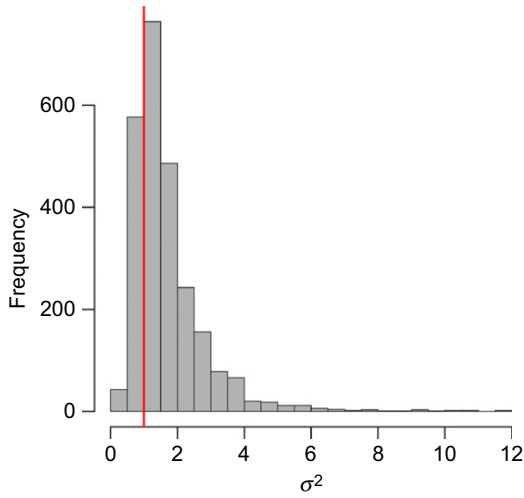

FIGURE 7.3 Posterior distribution of the temporal variance in apparent survival (red: value used for data generation).

The histogram of the posterior samples of the temporal variance for one simulated data set is shown in Fig. 7.3. The point estimate may seem biased; however, recall that this is just a single simulation, and we would need many simulations to check for any bias (see exercise 4 in Section 7.13).

7.4.3 Temporal Covariates

Often we are not only interested in getting a point estimate of survival or of its temporal variability, but also in identifying factors affecting survival. One way to do this is to see whether the observed temporal pattern in survival matches the temporal variation of an environmental factor (e.g., winter severity). From a nonzero correlation we would then infer an effect of that factor on survival. However, regardless of how the model is specified, such evidence is of correlative nature, thus causation cannot be inferred. A properly designed experiment is needed to infer causation, which is not easy in population studies (but see Schwarz, 2002).

Traditionally, so-called ultrastructural modeling has been used to model survival as a function of a covariate (x) (Lebreton et al., 1992; Link, 1999):

$$\text{logit}(\phi_{i,t}) = \mu + \beta x_t.$$

This model assumes that the entire temporal variability of survival could be explained by the covariate x; it is analogous to a linear regression model without residuals. This seems quite unrealistic, but in earlier times, this was the only way that the relationship between survival and a covariate could be modeled. A more realistic approach is to assume that only part of the temporal variability of survival is explained by the covariate, another part being unexplained random variation. Thus, we specify this model:

$$\text{logit}(\phi_{i,t}) = \mu + \beta x_t + \varepsilon_t$$
$$\varepsilon_t \sim \text{Normal}(0, \sigma^2).$$

The residual variance (σ^2) is the unexplained temporal variance. This allows us to estimate the amount of the total temporal variance which is explained by covariate x. We need to fit a model without the covariate to get an estimate of the total temporal variance (σ^2_{total}). The proportion of the variance explained by covariate x is then $(\sigma^2_{\text{total}} - \sigma^2)/\sigma^2_{\text{total}}$ (Grosbois et al., 2008).

To illustrate the model with the little owl example, we assume a mean survival of 0.65 and a negative effect of winter severity with logistic-linear slope of −0.3. The winter severity index is standardized (mean = 0, variance = 1) and the residual temporal variance not explained by winter severity has variance of 0.2.

```
# Define parameter values
n.occasions <- 20                    # Number of capture occasions
marked <- rep(15, n.occasions-1)     # Annual number of newly marked
                                       individuals

mean.phi <- 0.65
p <- rep(0.4, n.occasions-1)
beta <- -0.3                         # Slope of survival-winter
                                       relationship
r.var <- 0.2                         # Residual temporal variance

# Draw annual survival probabilities
winter <- rnorm(n.occasions-1, 0, 1^0.5)
logit.phi <- qlogis(mean.phi) + beta*winter + rnorm(n.occasions-1, 0,
   r.var^0.5)
phi <- plogis(logit.phi)

# Define matrices with survival and recapture probabilities
PHI <- matrix(phi, ncol = n.occasions-1, nrow = sum(marked),
   byrow = TRUE)
P <- matrix(p, ncol = n.occasions-1, nrow = sum(marked))
```

```
# Simulate capture-histories
CH <- simul.cjs(PHI, P, marked)

# Create vector with occasion of marking
get.first <- function(x) min(which(x!=0))
f <- apply(CH, 1, get.first)

# Specify model in BUGS language
sink("cjs-cov-raneff.bug")
cat("
model {

# Priors and constraints
for (i in 1:nind){
    for (t in f[i]:(n.occasions-1)){
        logit(phi[i,t]) <- mu + beta*x[t] + epsilon[t]
        p[i,t] <- mean.p
        } #t
    } #i
for (t in 1:(n.occasions-1)){
    epsilon[t] ~ dnorm(0, tau)
    phi.est[t] <- 1 / (1+exp(-mu-beta*x[t]-epsilon[t])) # Yearly
                                                          survival
    }
mu ~ dnorm(0, 0.001)                    # Prior for logit of mean survival
mean.phi <- 1 / (1+exp(-mu))            # Logit transformation
beta ~ dnorm(0, 0.001)I(-10, 10)        # Prior for slope parameter
sigma ~ dunif(0, 10)                    # Prior on standard deviation
tau <- pow(sigma, -2)
sigma2 <- pow(sigma, 2)                 # Residual temporal variance
mean.p ~ dunif(0, 1)                    # Prior for mean recapture

# Likelihood
for (i in 1:nind){
    # Define latent state at first capture
    z[i,f[i]] <- 1
    for (t in (f[i]+1):n.occasions){
        # State process
        z[i,t] ~ dbern(mu1[i,t])
        mu1[i,t] <- phi[i,t-1] * z[i,t-1]
        # Observation process
        y[i,t] ~ dbern(mu2[i,t])
        mu2[i,t] <- p[i,t-1] * z[i,t]
        } #t
    } #i
}
",fill = TRUE)
sink()

# Bundle data
bugs.data <- list(y = CH, f = f, nind = dim(CH)[1], n.occasions = dim(CH)[2],
   z = known.state.cjs(CH), x = winter)
```

```
# Initial values
inits <- function(){list(z = cjs.init.z(CH, f), mu = rnorm(1), sigma =
   runif(1, 0, 5), beta = runif(1, -5, 5), mean.p = runif(1, 0, 1))}

# Parameters monitored
parameters <- c("mean.phi", "mean.p", "phi.est", "sigma2", "beta")

# MCMC settings
ni <- 20000
nt <- 6
nb <- 10000
nc <- 3

# Call WinBUGS from R (BRT 12 min)
cjs.cov <- bugs(bugs.data, inits, parameters, "cjs-cov-raneff.bug",
   n.chains = nc, n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE,
   bugs.directory = bugs.dir, working.directory = getwd())
```

In general, models with random effects are more difficult to fit, and we need to run long Markov chains to achieve satisfactory convergence for all parameters. The posterior distributions of the slope and the residual environmental variation (temporal variation not explained by the covariate) are shown in Fig. 7.4.

```
# Summarize posteriors
print(cjs.cov, digits = 3)
```

```
               mean    sd   2.5%    25%    50%    75%  97.5%  Rhat n.eff
mean.phi      0.707 0.050  0.611  0.676  0.705  0.736  0.805 1.012  1400
mean.p        0.403 0.032  0.343  0.381  0.403  0.424  0.467 1.002  1800
phi.est[1]    0.686 0.121  0.423  0.610  0.696  0.769  0.902 1.003  1200
[ . . . ]
```

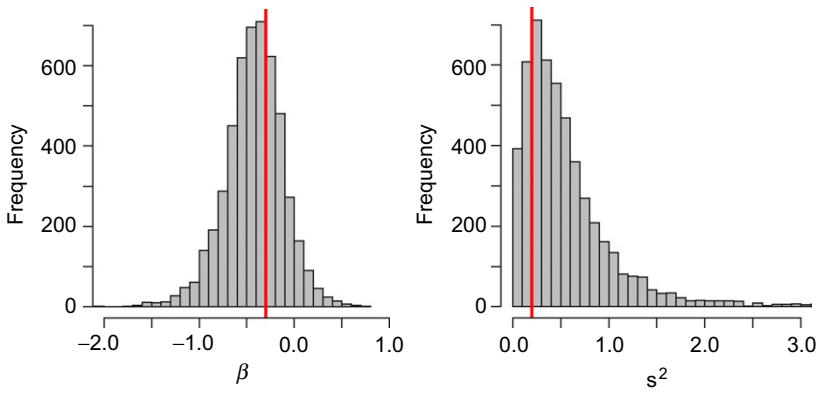

FIGURE 7.4 Posterior distributions of the covariate effect (slope parameter β) and of environmental variability (the residual temporal variance σ^2). Red lines indicate the values used for simulating the data.

```
phi.est[19]  0.681 0.120  0.427  0.603  0.690  0.764 0.902 1.003   950
sigma2       0.566 0.541  0.047  0.234  0.426  0.714 2.012 1.008   390
beta        -0.422 0.308 -1.080 -0.600 -0.410 -0.226 0.160 1.006  1400
```

```
# Produce graph
par(mfrow = c(1, 2), las = 1)
hist(cjs.cov$sims.list$beta, nclass = 25, col = "gray", main = "",
   xlab = expression(beta), ylab = "Frequency")
abline(v = -0.3, col = "red", lwd = 2)
hist(cjs.cov$sims.list$sigma2, nclass = 50, col = "gray", main = "",
   xlab = expression(sigma^2), ylab = "Frequency", xlim=c(0, 3))
abline(v = 0.2, col = "red", lwd = 2)
```

7.5 MODELS WITH INDIVIDUAL VARIATION

So far we have modeled temporal effects, assuming identical survival and recapture for all individuals. Now, we relax this assumption and model individual heterogeneity. Thus, we model effects along the row axis of the capture-history matrix. As with models for time effects, we can model individual effects in different ways—individual effects can be categorical or continuous, fixed or random, or latent or explained by measured covariates (compare with model M_h in Section 6.2). Moreover, individual effects can be constant over time, or they may change over time. Here, we only consider the simpler case, where they are constant over time.

7.5.1 Fixed Group Effects

We may specify fixed effects when we are interested in the estimates of particular groups (e.g., sex) and if the number of groups is low, as it is difficult to estimate the between-group variance with a small number of groups. We define g_i as a categorical variable with G levels (i.e., number of groups) indicating the group membership. The model for survival with a fixed group effect is

$$\phi_{i,t} = \beta_{g(i)},$$

where index $g(i)$ denotes the group g to which individual i belongs and β_g ($g = 1 \ldots G$) are the estimated fixed group effects. Because there are no other effects in the model, we can model directly on the probability scale, and the prior distribution ensures that the parameter estimates are between 0 and 1.

We illustrate the model to estimate sex-specific survival and recapture probabilities with simulated data of little owls. We first simulate two separate capture–recapture data sets; one for males and another for females.

Then we create the grouping variable g (named "group") and merge the two capture–recapture data sets.

```
# Define parameter values
n.occasions <- 12                    # Number of capture occasions
marked <- rep(30, n.occasions-1)     # Annual number of newly marked
                                       individuals
phi.f <- rep(0.65, n.occasions-1)    # Survival of females
p.f <- rep(0.6, n.occasions-1)       # Recapture prob. of females
phi.m <- rep(0.8, n.occasions-1)     # Survival of males
p.m <- rep(0.3, n.occasions-1)       # Recapture prob. of males

# Define matrices with survival and recapture probabilities
PHI.F <- matrix(phi.f, ncol = n.occasions-1, nrow = sum(marked))
P.F <- matrix(p.f, ncol = n.occasions-1, nrow = sum(marked))
PHI.M <- matrix(phi.m, ncol = n.occasions-1, nrow = sum(marked))
P.M <- matrix(p.m, ncol = n.occasions-1, nrow = sum(marked))

# Simulate capture-histories
CH.F <- simul.cjs(PHI.F, P.F, marked)
CH.M <- simul.cjs(PHI.M, P.M, marked)

# Merge capture-histories by row
CH <- rbind(CH.F, CH.M)

# Create group variable
group <- c(rep(1, dim(CH.F)[1]), rep(2, dim(CH.M)[1]))

# Create vector with occasion of marking
get.first <- function(x) min(which(x!=0))
f <- apply(CH, 1, get.first)
```

Finally, we write the model in BUGS language and fit it to the data.

```
# Specify model in BUGS language
sink("cjs-group.bug")
cat("
model {

# Priors and constraints
for (i in 1:nind){
   for (t in f[i]:(n.occasions-1)){
      phi[i,t] <- phi.g[group[i]]
      p[i,t] <- p.g[group[i]]
      } #t
   } #i
for (u in 1:g){
   phi.g[u] ~ dunif(0, 1)              # Priors for group-specific
                                         survival
   p.g[u] ~ dunif(0, 1)               # Priors for group-specific
                                         recapture
   }

# Likelihood
for (i in 1:nind){
```

```
    # Define latent state at first capture
    z[i,f[i]] <- 1
    for (t in (f[i]+1):n.occasions){
        # State process
        z[i,t] ~ dbern(mu1[i,t])
        mu1[i,t] <- phi[i,t-1] * z[i,t-1]
        # Observation process
        y[i,t] ~ dbern(mu2[i,t])
        mu2[i,t] <- p[i,t-1] * z[i,t]
        } #t
    } #i
}
",fill = TRUE)
sink()

# Bundle data
bugs.data <- list(y = CH, f = f, nind = dim(CH)[1], n.occasions = dim(CH)[2],
   z = known.state.cjs(CH), g = length(unique(group)), group = group)

# Initial values
inits <- function(){list(z = cjs.init.z(CH, f), phi.g = runif(length
   (unique(group)), 0, 1), p.g = runif(length(unique(group)), 0, 1))}

# Parameters monitored
parameters <- c("phi.g", "p.g")

# MCMC settings
ni <- 5000
nt <- 3
nb <- 2000
nc <- 3

# Call WinBUGS from R (BRT 2 min)
cjs.group <- bugs(bugs.data, inits, parameters, "cjs-group.bug",
   n.chains = nc, n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE,
   bugs.directory = bugs.dir, working.directory = getwd())
```

The parameter estimates are close to the values used to generate the data.

```
# Summarize posteriors
print(cjs.group, digits = 3)
```

```
          mean    sd  2.5%   25%   50%   75% 97.5%  Rhat n.eff
phi.g[1] 0.656 0.020 0.617 0.642 0.656 0.669 0.694 1.001  3000
phi.g[2] 0.796 0.018 0.760 0.784 0.796 0.808 0.831 1.002  1900
p.g[1]   0.599 0.029 0.541 0.578 0.599 0.619 0.654 1.002  1700
p.g[2]   0.325 0.021 0.285 0.311 0.324 0.339 0.368 1.002  1900
```

7.5.2 Random Group Effects

We may specify random group effects when we are interested in an overall mean and the variability between groups. A typical example of random group effects is provided by local populations, where we are

interested in estimating spatial variation of survival (Grosbois et al., 2009). Survival is then modeled as

$$\text{logit}(\phi_{i,t}) = \beta_{g(i)}$$
$$\beta_g \sim \text{Normal}(\bar{\beta}, \sigma^2),$$

where σ^2 is the variance of logit survival between groups, β_g are the random group effects, and $\bar{\beta}$ is the overall mean. Note that we now use the logit link function to ensure that the realized group-specific survival probabilities ($\text{logit}^{-1}(\beta_g)$) are bound in the interval [0, 1].

Because most BUGS code is identical to that in Section 7.5.1, we just show the part which needs modification:

```
# Priors and constraints
for (i in 1:nind){
    for (t in f[i]:(n.occasions-1)){
        logit(phi[i,t]) <- beta[group[i]]
        p[i,t] <- mean.p
        } #t
    } #i
for (u in 1:g){
    beta[u] ~ dnorm(mean.beta, tau)
    phi.g[u] <- 1 / (1+exp(-beta[u])) # Back-transformed
                                        group-specific survival
    }
mean.beta ~ dnorm(0, 0.001)          # Prior for logit of mean survival
mean.phi <- 1 / (1+exp(-mean.beta)) # Back-transformed mean survival
sigma ~ dunif(0, 10)                 # Prior for sd of logit of survival
                                       variability
tau <- pow(sigma, -2)
mean.p ~ dunif(0, 1)                 # Prior for mean recapture
```

7.5.3 Individual Random Effects

As an extreme case of a random group effect, we could also consider each individual as belonging to its own group. This model would not be identifiable when groups are treated as fixed affects, but it is when we treat groups as random effects. Conceptually, we imagine that there is an average survival, around which there is individual-specific noise. To specify individual random effects, we write

$$\text{logit}(\phi_{i,t}) = \mu + \varepsilon_i$$
$$\varepsilon_i \sim \text{Normal}(0, \sigma^2),$$

where σ^2 is the variance of logit survival among individuals, and μ is the overall mean logit survival. The interest of an analysis with individual random effects may be in estimating the mean, the variance, or even the realized survival "residuals" of each individual (sometimes called

"frailty"; Cam et al., 2002). Such a model also provides the base for modeling survival as a function of an individual covariate x_i (e.g., size of an individual). This model can be written as

$$\text{logit}(\phi_{i,t}) = \mu + \beta x_i + \varepsilon_i$$
$$\varepsilon_i \sim \text{Normal}(0, \sigma^2),$$

where β is the slope of covariate x on logit survival.

Models with random individual variation in survival are particularly important for the study of senescence (Cam et al., 2002). If individual variation is not included, senescence could easily be overlooked because a decline with age may be offset by increasing proportions of high-quality individuals in the population (Service, 2000; van de Pol and Verhulst, 2006). Sometimes, such a model may also be adopted for recapture probabilities because they are likely to differ among individuals in a similar manner.

Capture–recapture data are often subject to overdispersion, which may be due to a lack of independence among individuals (Lebreton et al., 1992). Overdispersion can be detected with a goodness-of-fit test (Lebreton et al., 1992; Choquet et al., 2001). If overdispersion is not corrected for, parameter estimators tend to be unbiased, but their variances (e.g., standard errors) will be too small (Anderson et al., 1994). The frequentist solution is to calculate a variance inflation factor from the goodness-of-fit test that is called c-hat in the capture–recapture literature, and to compute the true variance of the estimates as the product of the apparent variance and c-hat. An analogous Bayesian solution is to use a model with individual random effects (see also chapter 14 in Kéry 2010 and Section 4.2). The advantage of the Bayesian solution is the flexibility in specifying lack of independence in either survival only, recapture only, or in both parameters.

We now simulate little owl data and analyze them. We assume mean survival of 0.65 and individual variability with a variance of 0.5.

```
# Define parameter values
n.occasions <- 20                  # Number of capture occasions
marked <- rep(30, n.occasions-1)   # Annual number of newly marked
                                     individuals
mean.phi <- 0.65
p <- rep(0.4, n.occasions-1)
v.ind <- 0.5

# Draw annual survival probabilities
logit.phi <- rnorm(sum(marked), qlogis(mean.phi), v.ind^0.5)
phi <- plogis(logit.phi)

# Define matrices with survival and recapture probabilities
PHI <- matrix(phi, ncol = n.occasions-1, nrow = sum(marked),
   byrow = FALSE)
P <- matrix(p, ncol = n.occasions-1, nrow = sum(marked))
```

```
# Simulate capture-histories
CH <- simul.cjs(PHI, P, marked)

# Create vector with occasion of marking
get.first <- function(x) min(which(x!=0))
f <- apply(CH, 1, get.first)

# Specify model in BUGS language
sink("cjs-ind-raneff.bug")
cat("
model {

# Priors and constraints
for (i in 1:nind){
    for (t in f[i]:(n.occasions-1)){
        logit(phi[i,t]) <- mu + epsilon[i]
        p[i,t] <- mean.p
        } #t
    } #i
for (i in 1:nind){
    epsilon[i] ~ dnorm(0, tau)
    }
mean.phi ~ dunif(0, 1)                # Prior for mean survival
mu <- log(mean.phi / (1-mean.phi))    # Logit transformation
sigma ~ dunif(0, 5)                   # Prior for standard deviation
tau <- pow(sigma, -2)
sigma2 <- pow(sigma, 2)
mean.p ~ dunif(0, 1)                  # Prior for mean recapture

# Likelihood
for (i in 1:nind){
    # Define latent state at first capture
    z[i,f[i]] <- 1
    for (t in (f[i]+1):n.occasions){
        # State process
        z[i,t] ~ dbern(mu1[i,t])
        mu1[i,t] <- phi[i,t-1] * z[i,t-1]
        # Observation process
        y[i,t] ~ dbern(mu2[i,t])
        mu2[i,t] <- p[i,t-1] * z[i,t]
        } #t
    } #i
}
",fill = TRUE)
sink()

# Bundle data
bugs.data <- list(y = CH, f = f, nind = dim(CH)[1], n.occasions = dim(CH)[2],
   z = known.state.cjs(CH))

# Initial values
inits <- function(){list(z = cjs.init.z(CH, f), mean.phi = runif(1, 0, 1),
   mean.p = runif(1, 0, 1), sigma = runif(1, 0, 2))}
```

```
# Parameters monitored
parameters <- c("mean.phi", "mean.p", "sigma2")

# MCMC settings
ni <- 50000
nt <- 6
nb <- 20000
nc <- 3

# Call WinBUGS from R (BRT 73 min)
cjs.ind <- bugs(bugs.data, inits, parameters, "cjs-ind-raneff.bug",
   n.chains = nc, n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE,
   bugs.directory = bugs.dir, working.directory = getwd())
```

We need relatively long runs to reach satisfactory convergence. We note that the random-effects distribution could also be truncated like `epsilon[i] ~ dnorm(0, tau)I(-15, 15)` to improve mixing of the chains (see Appendix 1, tipp 16). The posterior distributions of the two parameters, mean survival and variability among individuals, show good agreement with the simulated parameters (Fig. 7.5).

```
# Summarize posteriors
print(cjs.ind, digits = 3)

           mean    sd  2.5%   25%   50%   75% 97.5%  Rhat  n.eff
mean.phi 0.640 0.026 0.587 0.623 0.641 0.658 0.688 1.001  15000
mean.p   0.410 0.021 0.368 0.396 0.410 0.424 0.451 1.001  13000
sigma2   0.586 0.244 0.176 0.410 0.560 0.739 1.132 1.012   1800

# Produce graph
par(mfrow = c(1, 2), las = 1)
hist(cjs.ind$sims.list$mean.phi, nclass = 25, col = "gray", main = "",
   xlab = expression(bar(phi)), ylab = "Frequency")
abline(v = mean.phi, col = "red", lwd = 2)
```

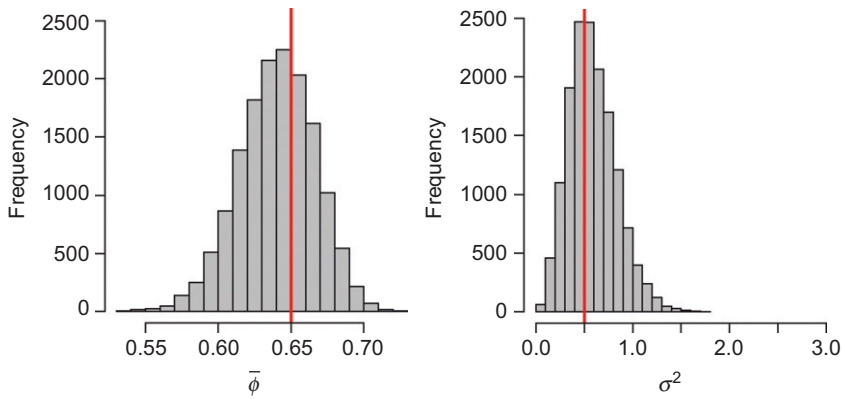

FIGURE 7.5 Posterior distributions of mean survival and of the individual variance in survival. Red lines indicate the values used for data simulation.

```
hist(cjs.ind$sims.list$sigma2, nclass = 15, col = "gray", main = "",
   xlab = expression(sigma^2), ylab = "Frequency", xlim = c(0, 3))
abline(v = v.ind, col = "red", lwd = 2)
```

If we wanted to estimate survival as a function of an individual covariate x, then we just have to adapt a small part in the code:

```
# Priors and constraints
for (i in 1:nind){
    for (t in f[i]:(n.occasions-1)){
        logit(phi[i,t]) <- mu + beta*x[i] + epsilon[i]
        p[i,t] <- mean.p
        } #t
    } #i
for (i in 1:nind){
    epsilon[i] ~ dnorm(0, tau)
    }
mean.phi ~ dunif(0, 1)                # Prior for mean survival
mu <- log(mean.phi / (1-mean.phi))    # Logit transformation
beta ~ dnorm(0, 0.001)                # Prior for covariate slope
sigma ~ dunif(0, 5)                   # Prior for standard deviation
tau <- pow(sigma, -2)
sigma2 <- pow(sigma, 2)
mean.p ~ dunif(0, 1)                  # Prior for mean recapture
```

Of course, we also have to give initial values for the new stochastic node beta, to include the covariate x in `bugs.data`, and to monitor beta.

Individual covariates may also change over time, such as, for example, body mass. The difficulty is that the covariate is unknown at occasions when the individual was not captured. Estimating the effects of individual time-varying covariates on survival is a challenge and different approaches have been proposed (Bonner and Schwarz, 2006; Catchpole et al., 2008; King et al., 2010).

7.6 MODELS WITH TIME AND GROUP EFFECTS

7.6.1 Fixed Group and Time Effects

Clearly we can combine the two concepts introduced in Sections 7.4 and 7.5 and model structure both along the time and along the individual axis of the capture-history matrix. The changes needed in the model code are merely an explicit GLM formulation of effects. This offers great flexibility as we can consider interacting or additive time and group effects, and we can treat either or both as random. The different combinations are straightforward and easy to implement, so we now focus in detail on one particular model that is often used, an additive model with fixed time and group effects.

Consider two groups of individuals (e.g., males and females) whose survival varies in parallel over time. Denoting sex by g (for group) and time by t, we can call this model $\{\phi_{g+t}, p_g\}$. Using the GLM formulation, we specify the survival model as

$$\text{logit}(\phi_{i,t}) = \beta_{g(i)} + \gamma_t,$$

where β_g is the effect of the sex g of individual i and γ_t are the fixed time effects. Written in this way, the model is overparameterized. We must either specify the β_g as the survival probabilities of the first year, and thus set $\gamma_1 = 0$, or we specify that γ_t are the survival probabilities of the first group and set $\beta_1 = 0$. Consequently, β_2 is then the difference in survival between the first and the second group. Such constraints must be specified in the BUGS model code, and are usually called corner constraints (Ntzoufras, 2009; Kéry, 2010).

For the simulation example, we assume constant recapture probabilities that are higher for females than for males. We simulate two capture-history data sets, one for males and one for females, merge them, create a group variable, and finally fit the model.

```
# Define parameter values
n.occasions <- 12                    # Number of capture occasions
marked <- rep(50, n.occasions-1)     # Annual number of newly marked
                                       individuals
phi.f <- c(0.6, 0.5, 0.55, 0.6, 0.5, 0.4, 0.6, 0.5, 0.55, 0.6, 0.7)
p.f <- rep(0.6, n.occasions-1)
diff <- 0.5        # Difference between male and female survival on logit
                     scale
phi.m <- plogis(qlogis(phi.f) + diff)
p.m <- rep(0.3, n.occasions-1)

# Define matrices with survival and recapture probabilities
PHI.F <- matrix(rep(phi.f, sum(marked)), ncol = n.occasions-1,
   nrow = sum(marked), byrow = TRUE)
P.F <- matrix(rep(p.f, sum(marked)), ncol = n.occasions-1,
   nrow = sum(marked), byrow = TRUE)
PHI.M <- matrix(rep(phi.m, sum(marked)), ncol = n.occasions-1,
   nrow = sum(marked), byrow = TRUE)
P.M <- matrix(rep(p.m, sum(marked)), ncol = n.occasions-1,
   nrow = sum(marked), byrow = TRUE)

# Simulate capture-histories
CH.F <- simul.cjs(PHI.F, P.F, marked)
CH.M <- simul.cjs(PHI.M, P.M, marked)

# Merge capture-histories
CH <- rbind(CH.F, CH.M)

# Create group variable
group <- c(rep(1, dim(CH.F)[1]), rep(2, dim(CH.M)[1]))
```

```
# Create vector with occasion of marking
get.first <- function(x) min(which(x!=0))
f <- apply(CH, 1, get.first)
```

The next piece of code writes the model in BUGS language, and the remaining R code fits the model:

```
# Specify model in BUGS language
sink("cjs-add.bug")
cat("
model {

# Priors and constraints
for (i in 1:nind){
    for (t in f[i]:(n.occasions-1)){
        logit(phi[i,t]) <- beta[group[i]] + gamma[t]
        p[i,t] <- p.g[group[i]]
        } #t
    } #i
# for survival parameters
for (t in 1:(n.occasions-1)){
    gamma[t] ~ dnorm(0, 0.01)I(-10, 10)          # Priors for time
                                                   effects
    phi.g1[t] <- 1 / (1+exp(-gamma[t]))          # Back-transformed
                                                   survival of males
    phi.g2[t] <- 1 / (1+exp(-gamma[t]-beta[2])) # Back-transformed
                                                   survival of females
    }
beta[1] <- 0                                  # Corner constraint
beta[2] ~ dnorm(0, 0.01)I(-10, 10)            # Prior for difference in male
                                                and female survival
# for recapture parameters
for (u in 1:g){
    p.g[u] ~ dunif(0, 1)                      # Priors for group-spec.
                                                recapture
    }

# Likelihood
for (i in 1:nind){
    # Define latent state at first capture
    z[i,f[i]] <- 1
    for (t in (f[i]+1):n.occasions){
        # State process
        z[i,t] ~ dbern(mu1[i,t])
        mu1[i,t] <- phi[i,t-1] * z[i,t-1]
        # Observation process
        y[i,t] ~ dbern(mu2[i,t])
        mu2[i,t] <- p[i,t-1] * z[i,t]
        } #t
    } #i
}
```

```
",fill = TRUE)
sink()

# Bundle data
bugs.data <- list(y = CH, f = f, nind = dim(CH)[1], n.occasions = dim(CH)[2],
   z = known.state.cjs(CH), g = length(unique(group)), group = group)

# Initial values
inits <- function(){list(z = cjs.init.z(CH, f), gamma =
   rnorm(n.occasions-1), beta = c(NA, rnorm(1)), p.g = runif(length
   (unique(group)), 0, 1))}

# Parameters monitored
parameters <- c("phi.g1", "phi.g2", "p.g", "beta")

# MCMC settings
ni <- 5000
nt <- 3
nb <- 2000
nc <- 3

# Call WinBUGS from R (BRT 7 min)
cjs.add <- bugs(bugs.data, inits, parameters, "cjs-add.bug",
   n.chains = nc, n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE,
   bugs.directory = bugs.dir, working.directory = getwd())

# Summarize posteriors
print(cjs.add, digits = 3)

            mean    sd  2.5%   25%   50%   75% 97.5%  Rhat  n.eff
phi.g1[1]  0.614 0.088 0.451 0.554 0.611 0.672 0.789 1.002   2800
phi.g1[2]  0.461 0.065 0.343 0.416 0.459 0.504 0.592 1.001   2800
[...]
phi.g2[10] 0.752 0.055 0.642 0.716 0.753 0.790 0.859 1.010    260
phi.g2[11] 0.823 0.079 0.683 0.770 0.818 0.868 0.999 1.030     90
p.g[1]     0.567 0.034 0.499 0.545 0.567 0.590 0.633 1.006    350
p.g[2]     0.318 0.022 0.277 0.302 0.317 0.333 0.361 1.005    450
beta[2]    0.603 0.127 0.360 0.515 0.605 0.687 0.848 1.008    300

# Figure of male and female survival
lower.f <- upper.f <- lower.m <- upper.m <- numeric()
for (t in 1:(n.occasions-1)){
    lower.f[t] <- quantile(cjs.add$sims.list$phi.g1[,t], 0.025)
    upper.f[t] <- quantile(cjs.add$sims.list$phi.g1[,t], 0.975)
    lower.m[t] <- quantile(cjs.add$sims.list$phi.g2[,t], 0.025)
    upper.m[t] <- quantile(cjs.add$sims.list$phi.g2[,t], 0.975)
    }
plot(x=(1:(n.occasions-1))-0.1, y = cjs.add$mean$phi.g1, type = "b",
   pch = 16, ylim = c(0.2, 1), ylab = "Survival probability",
   xlab = "Year", bty = "n", cex = 1.5, axes = FALSE)
axis(1, at = 1:11, labels = rep(NA,11), tcl = -0.25)
axis(1, at = seq(2,10,2), labels = c("2","4","6","8","10"))
axis(2, at = seq(0.2, 1, 0.1), labels = c("0.2", NA, "0.4", NA, "0.6", NA,
   "0.8", NA, "1.0"), las = 1)
segments((1:(n.occasions-1))-0.1, lower.f, (1:(n.occasions-1))-0.1,
   upper.f)
```

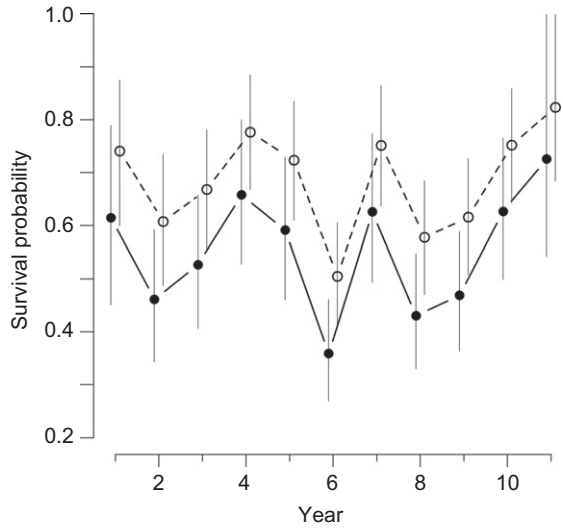

FIGURE 7.6 Posterior means (with 95% CRIs) of male (open circles) and female survival (closed symbols) under the additive model.

```
points(x = (1:(n.occasions-1))+0.1, y = cjs.add$mean$phi.g2,
    type = "b", pch = 1, lty = 2, cex = 1.5)
segments((1:(n.occasions-1))+0.1, lower.m, (1:(n.occasions-1))+0.1,
   upper.m)
```

The posterior means of male and female survival estimated under the additive model are shown in Fig. 7.6. Survival of the two sexes varies in parallel over time, but on the logit scale. Hence, on the probability scale the two curves are not parallel—as the difference becomes smaller the closer the estimates are to 1 or 0.

To fit a model with an interaction between sex and time (i.e., survival of each sex varies independently from each other over time), we would change the "Priors and constraints" part of the model as follows:

```
# Priors and constraints
for (i in 1:nind){
    for (t in f[i]:(n.occasions-1)){
        phi[i,t] <- eta.phi[group[i],t]
        p[i,t] <- p.g[group[i]]
        } #t
    } #i
# for survival parameters
for (u in 1:g){
    for (t in 1:(n.occasions-1)){
        eta.phi[u,t] ~ dunif(0, 1)    # Prior for time and group-spec.
                                        survival
```

```
        } #t
    } #g
# for recapture parameters
for (u in 1:g){
    p.g[u] ~ dunif(0, 1)              # Priors for group-spec. recapture
    }
```

7.6.2 Fixed Group and Random Time Effects

We may combine fixed group and random time effects to estimate temporal variability of survival (or recapture) in each group separately. As for the interacting model before, such a model would assume that the temporal variability of each group is independent of that in the other group(s).

$$\text{logit}(\phi_{i,t}) = \mu_{g(i)} + \varepsilon_{g(i),t}$$
$$\varepsilon_{g,t} \sim \text{Normal}(0, \sigma_g^2),$$

where μ_g are the group-specific means and σ_g^2 the group-specific temporal variances. The model code again only needs changes to the "Priors and constraints" part and looks like:

```
# Priors and constraints
for (i in 1:nind){
    for (t in f[i]:(n.occasions-1)){
        logit(phi[i,t]) <- eta.phi[group[i],t]
        p[i,t] <- p.g[group[i]]
        } #t
    } #i
# for survival parameters
for (u in 1:g){
    for (t in 1:(n.occasions-1)){
        eta.phi[u,t] <- mu.phi[u] + epsilon[u,t]
        epsilon[u,t] ~ dnorm(0, tau[u])
        } #t
        mean.phi[u] ~ dunif(0, 1)         # Priors on mean group-spec.
                                            survival
        mu.phi[u] <- log(mean.phi[u] / (1-mean.phi[u]))
        sigma[u] ~ dunif(0, 10)           # Priors for group-spec. sd
        tau[u] <- pow(sigma[u], -2)
        sigma2[u] <- pow(sigma[u], 2)
    } #g
# for recapture parameters
for (u in 1:g){
    p.g[u] ~ dunif(0,1)                   # Priors for group-spec.
                                            recapture
    }
```

An alternative way to write the same model is to treat the residuals as a realization from a multivariate normal distribution

$$\varepsilon_{g,t} \sim \text{MVN}(0, \Sigma_{g,t}),$$

where $\Sigma_{g,t}$ is the variance–covariance matrix that describes the temporal variance of and the temporal covariance among groups. As we assume independence among groups, the covariance between groups is zero, and the matrix for two groups is as follows:

$$\Sigma_{g,t} = \begin{pmatrix} \sigma^2_{g1} & 0 \\ 0 & \sigma^2_{g2} \end{pmatrix}.$$

Temporal variability in survival is usually induced by environmental factors (e.g., weather, food supply). As such, we do not expect survival of groups of individuals from the same population (e.g., sexes or age classes) to vary independently over time. Therefore, we may want to fit a sort of additive model, but where the temporal variance is treated as random. This can be done by considering a correlation of the temporal variability of each group, that is treating two sets of random effects as correleted (Link and Barker, 2005). The advantage of such a model is that (1) the temporal correlation is interpretable as a biological parameter (the extent to which survival varies in common among groups) and (2) the estimates of temporal variability become more precise because information is shared among groups. The temporal correlation of parameters also needs to be included in stochastic population models. Generally, the population growth rate becomes smaller with increasing positive correlation between survival parameters (Caswell, 2001).

With two groups, this model is written as follows:

$$\begin{aligned} &\text{logit}(\phi_{i,t}) = \mu_{g(i)} + \varepsilon_{g(i),t} \\ &\varepsilon_{g,t} \sim \text{MVN}(0, \Sigma_{g,t}) \\ &\Sigma_{g,t} = \begin{pmatrix} \sigma^2_{g1} & \rho\sigma_{g1}\sigma_{g2} \\ \rho\sigma_{g1}\sigma_{g2} & \sigma^2_{g2} \end{pmatrix}, \end{aligned}$$

where ρ is the temporal correlation coefficient between the two groups. Note that the correlation coefficient between two variables g_1 and g_2 is

$$\rho = \frac{\text{cov}(g1, g2)}{\sqrt{\sigma^2_{g1}\sigma^2_{g2}}}, \text{ and thus } \Sigma_{g,t} \text{ could also be written as}$$

$$\Sigma_{g,t} = \begin{pmatrix} \sigma^2_{g1} & \text{cov}(g1, g2) \\ \text{cov}(g1, g2) & \sigma^2_{g2} \end{pmatrix}.$$

Estimating correlation coefficients (or covariances) is challenging, in particular, if there are more than two parameters. This is because several conditions must be met. For example, all correlations must be in the interval –1 and 1, and they are jointly constrained in a complicated way. A standard choice for the prior of the elements of matrix Σ is the inverse

Wishart distribution, which ensures that the estimated parameters have the desired properties.

The inverse Wishart distribution (IW(R, df)) has two parameters: the scale matrix (R), with dimension $K \times K$ for K modeled parameters, and the degrees of freedom (df). Depending on the choice of these parameters, we incorporate into the analysis prior information about the correlation coefficients or about the variances (Link and Barker, 2005; Gelman and Hill, 2007). For a uniform prior on the correlation coefficients, we must fix $df = K + 1$. The values of the scale matrix R have an effect on the priors for the variances; large values of R set the prior means of the variances to large values. Because the specification of the priors for matrix Σ is difficult, we recommend conducting sensitivity analyses. The BUGS code to fit this model is as follows:

```
# Specify model in BUGS language
sink("cjs-temp-corr.bug")
cat("
model {

# Priors and constraints
for (i in 1:nind){
    for (t in f[i]:(n.occasions-1)){
        logit(phi[i,t]) <- eta.phi[t,group[i]]
        p[i,t] <- p.g[group[i]]
        } #t
    } #i
# for survival parameters
for (t in 1:(n.occasions-1)){
    eta.phi[t,1:g] ~ dmnorm(mu.phi[], Omega[,])
    } #t
for (u in 1:g){
    mean.phi[u] ~ dunif(0, 1)       # Priors on mean group-spec. survival
    mu.phi[u] <- log(mean.phi[u] / (1-mean.phi[u]))
    } #g
Omega[1:g, 1:g] ~ dwish(R[,], df) # Priors for variance-covariance
                                    matrix
Sigma[1:g, 1:g] <- inverse(Omega[,])

# for recapture parameters
for (u in 1:g){
    p.g[u] ~ dunif(0, 1)            # Priors for group-spec. recapture
    }

# Likelihood
for (i in 1:nind){
    # Define latent state at first capture
    z[i,f[i]] <- 1
    for (t in (f[i]+1):n.occasions){
        # State process
        z[i,t] ~ dbern(mu1[i,t])
        mu1[i,t] <- phi[i,t-1] * z[i,t-1]
        # Observation process
        y[i,t] ~ dbern(mu2[i,t])
```

```
      mu2[i,t] <- p[i,t-1] * z[i,t]
      } #t
    } #i
}
",fill = TRUE)
sink()
```

The parameters of the inverse Wishart distribution (R, df) are provided as data. Here, we choose parameters that result in an uninformative prior for the correlation.

```
# Bundle data
bugs.data <- list(y = CH, f = f, nind = dim(CH)[1], n.occasions = dim(CH)[2],
   z = known.state.cjs(CH), g = length(unique(group)), group = group,
   R = matrix(c(5, 0, 0, 1), ncol = 2), df = 3)

# Initial values
inits <- function(){list(z = cjs.init.z(CH, f), p.g = runif(length
   (unique(group)), 0, 1), Omega = matrix(c(1, 0, 0, 1), ncol = 2))}

# Parameters monitored
parameters <- c("eta.phi", "p.g", "Sigma", "mean.phi")

# MCMC settings
ni <- 5000
nt <- 3
nb <- 2000
nc <- 3

# Call WinBUGS from R (BRT 5 min)
cjs.corr <- bugs(bugs.data, inits, parameters, "cjs-temp-corr.bug",
   n.chains = nc, n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE,
   bugs.directory = bugs.dir, working.directory = getwd())

# Summarize posteriors
print(cjs.corr, digits = 3)
```

```
                mean    sd   2.5%    25%   50%   75% 97.5%  Rhat n.eff
eta.phi[1,1]   0.457 0.391 -0.257  0.190 0.434 0.688 1.304 1.003   870
eta.phi[1,2]   0.794 0.384  0.103  0.537 0.770 1.020 1.605 1.001  3000
[ ... ]
eta.phi[11,1] 0.945 0.445  0.219  0.647 0.892 1.194 1.995 1.010   420
eta.phi[11,2] 0.800 0.363  0.165  0.554 0.762 1.031 1.546 1.002  1100
p.g[1]         0.572 0.032  0.511  0.550 0.572 0.594 0.636 1.002  1600
p.g[2]         0.327 0.023  0.283  0.311 0.327 0.343 0.375 1.003   970
Sigma[1,1]     0.790 0.391  0.323  0.523 0.704 0.957 1.793 1.001  3000
Sigma[1,2]     0.073 0.156 -0.197 -0.014 0.057 0.146 0.440 1.003  3000
Sigma[2,1]     0.073 0.156 -0.197 -0.014 0.057 0.146 0.440 1.003  3000
Sigma[2,2]     0.243 0.154  0.082  0.144 0.205 0.295 0.631 1.002  1100
mean.phi[1]    0.549 0.067  0.419  0.504 0.549 0.594 0.678 1.002  1400
mean.phi[2]    0.669 0.039  0.593  0.644 0.669 0.694 0.749 1.001  2200
```

Σ_{11} (note that this is sigma[1,1] in the table above) is the temporal variance of the logit male survival, and Σ_{22} is that for logit female survival.

The elements Σ_{12} and Σ_{21} are the temporal covariances of logit male and logit female survival. This quantity may not be easy to interpret, and we may want to compute the temporal correlation of male and female survival:

```
corr.coef <- cjs.corr$sims.list$Sigma[,1,2] / sqrt(cjs.corr$sims.
  list$Sigma[,1,1] * cjs.corr$sims.list$Sigma[,2,2])
```

The mean and the credible interval of the correlation coefficient (ρ) are 0.16 (−0.43, 0.67), and the probability that $\rho > 0$ is 0.71. As usual these quantities are computed from the posterior distribution of `corr.coef`.

7.7 MODELS WITH AGE EFFECTS

Survival often changes with age. For most species, survival in their first year of life is lower than later. In addition, with senescence, survival may decline in older age classes. Therefore, we might want to estimate different survival parameters for each age class. To model age effects on survival, individuals must be aged when they are first captured, although recently developed models allow relaxing this assumption for some of the individuals (Pledger et al., 2009). We create a matrix $x_{i,t}$, indicating the age at each time t for each individual i. For example, assume a study over 6 years and two individuals that are first captured at the second occasion. The elements of matrix x would then be [NA 1 2 3 4] for the first individual that was born at the second occasion and [NA 2 3 4 5] for the second individual that was 1-year old at the second occasion. We can then model survival as a function of age x as follows:

$$\text{logit}(\phi_{i,t}) = \mu + \beta x_{i,t} + \varepsilon_i$$
$$\varepsilon_i \sim \text{Normal}(0, \sigma^2).$$

This model can be adapted very flexibly. First, we may include an individual random effect (ε_i) as already shown above. Inclusion of individual "frailty" can be important, if we aim at estimating senescence. Second, we may assume that survival changes linearly with age (as in the formula above), that it changes nonlinearly with age (e.g., Gaillard et al., 2004), or that x is a categorical variable. The last is perhaps the most frequent type of model for age effects adopted in practice. If we distinguish only two age classes (i.e., separate survival during the first year of life vs. later), the elements of matrix x would become [NA 1 2 2 2] for the first individual above and [NA 2 2 2 2] for the second individual. Finally, one might include time effects in addition to age effects. A general formation might then be

$$\text{logit}(\phi_{i,t}) = \beta_{x(i,t)} + \varepsilon_i$$
$$\varepsilon_i \sim \text{Normal}(0, \sigma^2),$$

where $\beta_{x(i,t)}$ are the effects of age class x of individual i at time t and ε_i are individual frailty terms. Note that a principal difference between the first and the second model is that the age variable x is continuous in the first but categorical in the second model.

To illustrate the model, we consider a simple example, in which juvenile and adult little owls are marked. We assume that survival in the first year of life (from age 0 to age 1 year) is different from survival in subsequent age classes (from age 1 year onward). Thus, we need a model with two age classes for survival. We simulate data first, by creating two data sets, one for individuals marked as juveniles, and one for individuals marked as adults. We then construct matrix x for each age class and merge the two data sets and matrices (x).

```
# Define parameter values
n.occasions <- 10                       # Number of capture occasions
marked.j <- rep(200, n.occasions-1)     # Annual number of newly marked
                                          juveniles
marked.a <- rep(30, n.occasions-1)      # Annual number of newly marked
                                          adults
phi.juv <- 0.3                          # Juvenile annual survival
phi.ad <- 0.65                          # Adult annual survival
p <- rep(0.5, n.occasions-1)            # Recapture
phi.j <- c(phi.juv, rep(phi.ad, n.occasions-2))
phi.a <- rep(phi.ad, n.occasions-1)

# Define matrices with survival and recapture probabilities
PHI.J <- matrix(0, ncol = n.occasions-1, nrow = sum(marked.j))
for (i in 1:length(marked.j)){
    PHI.J[(sum(marked.j[1:i])-marked.j[i]+1):sum(marked.j[1:i]),
      i:(n.occasions-1)] <- matrix(rep(phi.j[1:(n.occasions-i)],
      marked.j[i]), ncol = n.occasions-i, byrow = TRUE)
    }
P.J <- matrix(rep(p, sum(marked.j)), ncol = n.occasions-1,
   nrow = sum(marked.j), byrow = TRUE)
PHI.A <- matrix(rep(phi.a, sum(marked.a)), ncol = n.occasions-1,
   nrow = sum(marked.a), byrow = TRUE)
P.A <- matrix(rep(p, sum(marked.a)), ncol = n.occasions-1,
   nrow = sum(marked.a), byrow = TRUE)

# Apply simulation function
CH.J <- simul.cjs(PHI.J, P.J, marked.j)
CH.A <- simul.cjs(PHI.A, P.A, marked.a)

# Create vector with occasion of marking
get.first <- function(x) min(which(x!=0))
f.j <- apply(CH.J, 1, get.first)
f.a <- apply(CH.A, 1, get.first)

# Create matrices X indicating age classes
x.j <- matrix(NA, ncol = dim(CH.J)[2]-1, nrow = dim(CH.J)[1])
x.a <- matrix(NA, ncol = dim(CH.A)[2]-1, nrow = dim(CH.A)[1])
for (i in 1:dim(CH.J)[1]){
```

```
    for (t in f.j[i]:(dim(CH.J)[2]-1)){
        x.j[i,t] <- 2
        x.j[i,f.j[i]] <- 1
        } #t
    } #i
for (i in 1:dim(CH.A)[1]){
    for (t in f.a[i]:(dim(CH.A)[2]-1)){
        x.a[i,t] <- 2
        } #t
    } #i
```

Next, we combine the two data sets into a common set.

```
CH <- rbind(CH.J, CH.A)
f <- c(f.j, f.a)
x <- rbind(x.j, x.a)
```

Finally, we define the model in BUGS language and fit it to the data. We treat age as a categorical variable, so we use the identity link.

```
# Specify model in BUGS language
sink("cjs-age.bug")
cat("
model {

# Priors and constraints
for (i in 1:nind){
    for (t in f[i]:(n.occasions-1)){
        phi[i,t] <- beta[x[i,t]]
        p[i,t] <- mean.p
        } #t
    } #i
for (u in 1:2){
    beta[u] ~ dunif(0, 1)              # Priors for age-specific survival
    }
mean.p ~ dunif(0, 1)                   # Prior for mean recapture

# Likelihood
for (i in 1:nind){
    # Define latent state at first capture
    z[i,f[i]] <- 1
    for (t in (f[i]+1):n.occasions){
        # State process
        z[i,t] ~ dbern(mu1[i,t])
        mu1[i,t] <- phi[i,t-1] * z[i,t-1]
        # Observation process
        y[i,t] ~ dbern(mu2[i,t])
        mu2[i,t] <- p[i,t-1] * z[i,t]
        } #t
    } #i
}
",fill = TRUE)
sink()
```

```
# Bundle data
bugs.data <- list(y = CH, f = f, nind = dim(CH)[1], n.occasions =
dim(CH)[2], z = known.state.cjs(CH), x = x)

# Initial values
inits <- function(){list(z = cjs.init.z(CH, f), beta = runif(2, 0, 1),
mean.p = runif(1, 0, 1))}

# Parameters monitored
parameters <- c("beta", "mean.p")

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

# Call WinBUGS from R (BRT 3 min)
cjs.age <- bugs(bugs.data, inits, parameters, "cjs-age.bug", n.chains =
   nc, n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE,
   bugs.directory = bugs.dir, working.directory = getwd())
```

The model runs slowly, but convergence is achieved after only 1000 samples. The parameter estimates are close to the parameters used for the simulations.

```
print(cjs.age, digits = 3)

          mean    sd  2.5%   25%   50%   75% 97.5%  Rhat  n.eff
beta[1]  0.317 0.015 0.287 0.306 0.318 0.328 0.347 1.002    880
beta[2]  0.666 0.015 0.638 0.657 0.667 0.676 0.695 1.001   1000
mean.p   0.486 0.019 0.452 0.473 0.486 0.499 0.525 1.005    410
```

It is straightforward to include other models for the age effect. Depending on the models that we want to fit, matrix x needs to be adapted. If we want to model survival as a linear function of age, x must indicate the true age in each year. If the goal is to treat age as a categorical variable, x must include as many categories as we want to distinguish (e.g., two above). Then the GLM, which relates survival to x, needs to be adapted. For example, if survival is modeled as a linear function of age, we first create x and only include into the analysis individuals marked as juveniles.

```
# Create matrix X indicating age classes
x <- matrix(NA, ncol = dim(CH)[2]-1, nrow = dim(CH)[1])
for (i in 1:dim(CH)[1]){
    for (t in f[i]:(dim(CH)[2]-1)){
        x[i,t] <- t-f[i]+1
        } #t
    } #i
```

As usual, the BUGS model needs a few changes in the "Priors and constraints" part:

```
# Priors and constraints
for (i in 1:nind){
    for (t in f[i]:(n.occasions-1)){
        logit(phi[i,t]) <- mu + beta*x[i,t]
        p[i,t] <- mean.p
        } #t
    } #i
mu ~ dnorm(0, 0.01)              # Prior for mean of logit survival
beta ~ dnorm(0, 0.01)            # Prior for slope parameter
for (i in 1:(n.occasions-1)){
    phi.age[i] <- 1 / (1+exp(-mu -beta*i))  # Logit back-transformation
    }
mean.p ~ dunif(0, 1)                        # Prior for mean recapture
```

Age effects can also be combined with time effects in a very similar way as we have seen with group effects (see Section 7.6). Models can be specified in which survival of the defined age classes vary independently from each other across time, in which the temporal pattern of the age classes is additive, or in which only survival of one age class is time-dependent. Models with random time effects are also useful, allowing the temporal variability of survival of each age class to be modeled independently, or in which the temporal correlation is estimated. It is also possible to consider cohort effects (Reid et al., 2003), that is, the survival of individuals born in one cohort (year) is different from the survival of individuals born in another cohort. This requires that we define a variable indicating the cohort for each individual. In fact, for individuals that are young when marked, our vector *f* already includes this information. Survival is then modeled as a function of *f*, we may consider it to be fixed or random, and we can combine it with additional time and/or age effects. Care must be taken with model specification because a model with cohort x time interaction is the same as a model with age x time interaction or one with cohort x age interaction.

7.8 IMMEDIATE TRAP RESPONSE IN RECAPTURE PROBABILITY

One assumption of standard capture–recapture models is that all marked animals alive and available for capture at a given occasion have the same capture probability. Sometimes, this assumption is violated in a very specific way, namely when individuals captured at time $t - 1$ have a different recapture probability at time t than individuals not captured at time $t - 1$. This is called immediate trap response (see also Section 6.2.3).

If recapture probability at time t for individuals captured at $t-1$ is higher than for individuals not captured at $t-1$, this is "trap-happiness" and if recapture probability is lower, then it is called "trap-shyness". Trap-happiness can occur if baited traps are used, and trap-shyness can occur if the interval between capture occasions is short (Pradel, 1993). These effects may also be induced by the sampling method and not reflect a behavioral change of the individuals. However, trap response must be modeled; otherwise, survival estimates will be biased. To account for immediate trap response, a multistate model can be used (Gimenez et al., 2003; Schaub et al., 2009; Appendix 2.2), but here we will use a single-state model and model recapture as a function of whether or not an individual was captured at the preceding occasion. We need, therefore, to construct a matrix m that contains this information. The element of m for individual i at time t takes value 1 if individual i was captured at $t-1$, and value 2 otherwise. The recapture probability is then modeled as

$$p_{i,t} = \beta_{m(i,t)},$$

where β_m takes two values, depending on whether $m_{i,t}$ is 1 or 2. We may also include additive time effects and use the logit link function,

$$\text{logit}(p_{i,t}) = \beta_{m(i,t)} + \gamma_t.$$

The model with interaction between time and behavioral response is parameter-redundant (Gimenez et al., 2003).

Simulating such data is best done with a multistate model (see Appendix 2.2). For illustration, we imagine that we wish to estimate survival of red-backed shrikes (Fig. 7.7), a beautiful bird species of hedgerows. We catch adults during the breeding season, mark them with color rings to facilitate resighting in subsequent years, and survey all potential breeding territories each year. Typically, we focus on breeding territories that were occupied in previous years. If time allows, we search for other, newly established territories. Thus, marked individuals that survive and return to their territory have a higher chance of being resighted, while individuals that establish new territories are less likely to be found. However, once they are found, their chances of being resighted in the next year increase. Such a sampling protocol, which is not uncommon in studies of color-marked birds, induces a "trap-happy effect" which biases survival unless accounted for. For data simulation, we assume survival $\phi = 0.55$ and resighting probabilities $p_{ss} = 0.75$ following a sighting in the preceding year and $p_{ns} = 0.35$ otherwise.

```
# Import data
CH <- as.matrix(read.table(file = "trap.txt", sep = " "))

# Compute vector with occasion of first capture
get.first <- function(x) min(which(x!=0))
f <- apply(CH, 1, get.first)
```

FIGURE 7.7 Male red-backed shrike (*Lanius collurio*) feeding a fledgling (Photograph by D. Studler).

```
# Create matrix m indicating when an individual was captured
m <- CH[,1:(dim(CH)[2]-1)]
u <- which(m==0)
m[u] <- 2
```

The capture-histories of the first four individuals are as follows:

```
1 1 0 0 0 0 0 0
1 0 0 0 0 0 0 0
1 1 0 0 1 1 1 0
1 1 0 1 1 0 1 0
```

and the corresponding matrix *m* for these individuals is

```
1 1 2 2 2 2 2
1 2 2 2 2 2 2
1 1 2 2 1 1 1
1 1 2 1 1 2 1
```

Here a 1 denotes that an individual was captured at the preceding occasion, and a 2 denotes that it was not captured at the preceding occasion. Matrix *m* has as many columns as there are recapture parameters, thus one fewer than the total number of capture occasions.

The BUGS code to fit the trap-response model is as follows:

```
# Specify model in BUGS language
sink("cjs-trap.bug")
cat("
model {

# Priors and constraints
for (i in 1:nind){
    for (t in f[i]:(n.occasions-1)){
        phi[i,t] <- mean.phi
        p[i,t] <- beta[m[i,t]]
        } #t
    } #i
mean.phi ~ dunif(0, 1)              # Prior for mean survival
for (u in 1:2){
    beta[u] ~ dunif(0, 1)           # Priors for recapture
    }

# Likelihood components
for (i in 1:nind){
    # Define latent state at first capture
    z[i,f[i]] <- 1
    for (t in (f[i]+1):n.occasions){
        # State process
        z[i,t] ~ dbern(mu1[i,t])
        mu1[i,t] <- phi[i,t-1] * z[i,t-1]
        # Observation process
        y[i,t] ~ dbern(mu2[i,t])
        mu2[i,t] <- p[i,t-1] * z[i,t]
        } #t
    } #i
}
",fill = TRUE)
sink()

# Bundle data
bugs.data <- list(y = CH, f = f, nind = dim(CH)[1], n.occasions =
   dim(CH)[2], z = known.state.cjs(CH), m = m)

# Initial values
inits <- function(){list(z = cjs.init.z(CH, f), mean.phi = runif(1, 0,
   1), beta = runif(2, 0, 1))}

# Parameters monitored
parameters <- c("mean.phi", "beta")

# MCMC settings
ni <- 20000
nt <- 3
nb <- 10000
nc <- 3
```

```
# Call WinBUGS from R (BRT 1 min)
cjs.trap <- bugs(bugs.data, inits, parameters, "cjs-trap.bug",
   n.chains = nc, n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE,
   bugs.directory = bugs.dir, working.directory = getwd())
```

The estimated parameters are close to the parameters used for the simulation.

```
print(cjs.trap, digits = 3)
```

```
          mean    sd  2.5%   25%   50%   75% 97.5%  Rhat  n.eff
mean.phi 0.567 0.076 0.462 0.515 0.552 0.602 0.763 1.006   2400
beta[1]  0.756 0.091 0.547 0.701 0.770 0.823 0.897 1.006   2700
beta[2]  0.379 0.207 0.063 0.210 0.359 0.527 0.814 1.003   4500
```

This approach is again very flexible and can be extended easily. For example, if an individual may be captured more than once during an occasion, those captured more may have a higher capture probability. By including the information about how many times an individual was captured, we can adjust for this sort of capture heterogeneity (Fletcher, 1994). The matrix m then contains the number of times an individual is caught at an occasion, and recapture is modeled as a function of m.

7.9 PARAMETER IDENTIFIABILITY

In principle, we are quite free to specify any among a large number of models, especially when using BUGS. However, there is no guarantee that all parameters in a fitted model are indeed identified, that is, can be estimated. In fact, it is common that some parameters are not identifiable. There are two kinds of nonidentifiability: intrinsic and extrinsic. A model has intrinsically identifiable parameters if the same likelihood for the data cannot be obtained by a smaller number of parameters, while parameter-redundant models (those with at least one unidentified parameter) can be expressed in terms of fewer than the original number of parameters (Catchpole and Morgan, 1997). Extrinsic nonidentifiability refers to the situation where a parameter should be identifiable given the structure of a model but is not because the particular data set is insufficient in some regard. Thus, intrinsic nonidentifiability is a feature of a model while extrinsic nonidentifiability is a feature of a data set. Intrinsic nonidentifiability of models can be studied without data using symbolic algebra (Catchpole and Morgan, 1997; Catchpole et al., 2001; Gimenez et al., 2003, 2004) or the analysis of "perfect" data (analytic-numeric method; Burnham et al., 1987), while extrinsic nonidentifiability is best studied using simulation (e.g., Schaub et al., 2004a; Schaub, 2009; Bailey et al., 2010). Of course, intrinsic and extrinsic nonidentifiability may occur together for a particular model and data set.

In the Bayesian framework, the topic of nonidentifiability is slightly different. Because the posterior is a combination of the likelihood and the prior, the posterior is defined (provided that the prior is proper; Gelman et al., 2004). However, if the information in the data is very low for a particular parameter (i.e., there is extrinsic nonidentifiability) or if the likelihood surface is completely flat for a parameter (intrinsic nonidentifiability), then the posterior will simply reflect the prior for that parameter. Therefore, a prior sensitivity analysis can give insights into the identifiability of a parameter. Gimenez et al. (2009b) developed an approach to assess parameter identifiability based on this idea. Using flat priors for parameters, they compared the overlap between the prior and the posterior. If the overlap between the two distributions is large, a parameter is weakly identifiable.

Here, we illustrate this with a well-known example. In the classical, fully time-dependent CJS model $\{\phi_t, p_t\}$, the last survival and the last recapture probability are not identifiable—it is only possible to estimate the product of the two (Lebreton et al., 1992). Thus, this is an intrinsic identifiability problem. In the following example, we fit the model $\{\phi_t, p_t\}$ to the data and use flat priors for all parameters. We then inspect the posterior and the prior of some survival and recapture parameters.

```
# Define parameter values
n.occasions <- 12                          # Number of capture occasions
marked <- rep(30, n.occasions-1)           # Annual number of newly marked
                                             individuals
phi <- c(0.6, 0.5, 0.55, 0.6, 0.5, 0.4, 0.6, 0.5, 0.55, 0.6, 0.7)
p <- c(0.4, 0.65, 0.4, 0.45, 0.55, 0.68, 0.66, 0.28, 0.55, 0.45, 0.35)

# Define matrices with survival and recapture probabilities
PHI <- matrix(phi, ncol = n.occasions-1, nrow = sum(marked), byrow = TRUE)
P <- matrix(p, ncol = n.occasions-1, nrow = sum(marked), byrow = TRUE)

# Simulate capture-histories
CH <- simul.cjs(PHI, P, marked)

# Create vector with occasion of marking
get.first <- function(x) min(which(x!=0))
f <- apply(CH, 1, get.first)

# Specify model in BUGS language
sink("cjs-t-t.bug")
cat("
model {

# Priors and constraints
for (i in 1:nind){
   for (t in f[i]:(n.occasions-1)){
      phi[i,t] <- phi.t[t]
      p[i,t] <- p.t[t]
      } #t
   } #i
```

```
for (t in 1:(n.occasions-1)){
    phi.t[t] ~ dunif(0, 1)          # Priors for time-spec. survival
    p.t[t] ~ dunif(0, 1)            # Priors for time-spec. recapture
    }

# Likelihood
for (i in 1:nind){
    # Define latent state at first capture
    z[i,f[i]] <- 1
    for (t in (f[i]+1):n.occasions){
        # State process
        z[i,t] ~ dbern(mu1[i,t])
        mu1[i,t] <- phi[i,t-1] * z[i,t-1]
        # Observation process
        y[i,t] ~ dbern(mu2[i,t])
        mu2[i,t] <- p[i,t-1] * z[i,t]
        } #t
    } #i
}
",fill = TRUE)
sink()

# Bundle data
bugs.data <- list(y = CH, f = f, nind = dim(CH)[1], n.occasions = dim(CH)[2],
   z = known.state.cjs(CH))

# Initial values
inits <- function(){list(z = cjs.init.z(CH, f), phi.t = runif((dim(CH)
   [2]-1), 0, 1), p.t = runif((dim(CH)[2]-1), 0, 1))}

# Parameters monitored
parameters <- c("phi.t", "p.t")

# MCMC settings
ni <- 25000
nt <- 3
nb <- 20000
nc <- 3

# Call WinBUGS from R (BRT 7 min)
cjs.t.t <- bugs(bugs.data, inits, parameters, "cjs-t-t.bug", n.chains =
      nc, n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE,
      bugs.directory = bugs.dir, working.directory = getwd())

# Plot posterior distributions of some phi and p
par(mfrow = c(2, 2), cex = 1.2, las = 1, mar=c(5, 4, 2, 1))
plot(density(cjs.t.t$sims.list$phi.t[,6]), xlim = c(0, 1), ylim = c(0, 5),
   main = "", xlab = expression(phi[6]), ylab = "Density", frame = FALSE,
   lwd = 2)
abline(h = 1, lty = 2, lwd = 2)
par(mar=c(5, 3, 2, 2))
plot(density(cjs.t.t$sims.list$phi.t[,11]), xlim = c(0, 1),
    ylim =c(0, 5), main = "", xlab = expression(phi[11]), ylab ="",
    frame = FALSE, lwd = 2)
abline(h = 1, lty = 2, lwd = 2)
par(mar=c(5, 4, 2, 1))
```

```
plot(density(cjs.t.t$sims.list$p.t[,6]), xlim = c(0, 1), ylim = c(0, 5),
   main = "", xlab = expression(p[6]), ylab = "Density", frame = FALSE,
   lwd = 2)
abline(h = 1, lty = 2, lwd = 2)
par(mar=c(5, 3, 2, 2))
plot(density(cjs.t.t$sims.list$p.t[,11]), xlim = c(0, 1), ylim =
   c(0, 5), main = "", xlab = expression(p[11]), ylab ="", frame = FALSE,
   lwd = 2)
abline(h = 1, lty = 2, lwd = 2)
```

To inspect the result, we plot the posterior and prior densities of some parameters (Fig. 7.8). It is obvious that ϕ_6 and p_6 are identifiable: their posterior is nicely peaked and does not overlap much with the prior distribution. By contrast, the posterior distributions of ϕ_{11} and p_{11} do not have a clear peak and the overlap with the prior is large. These

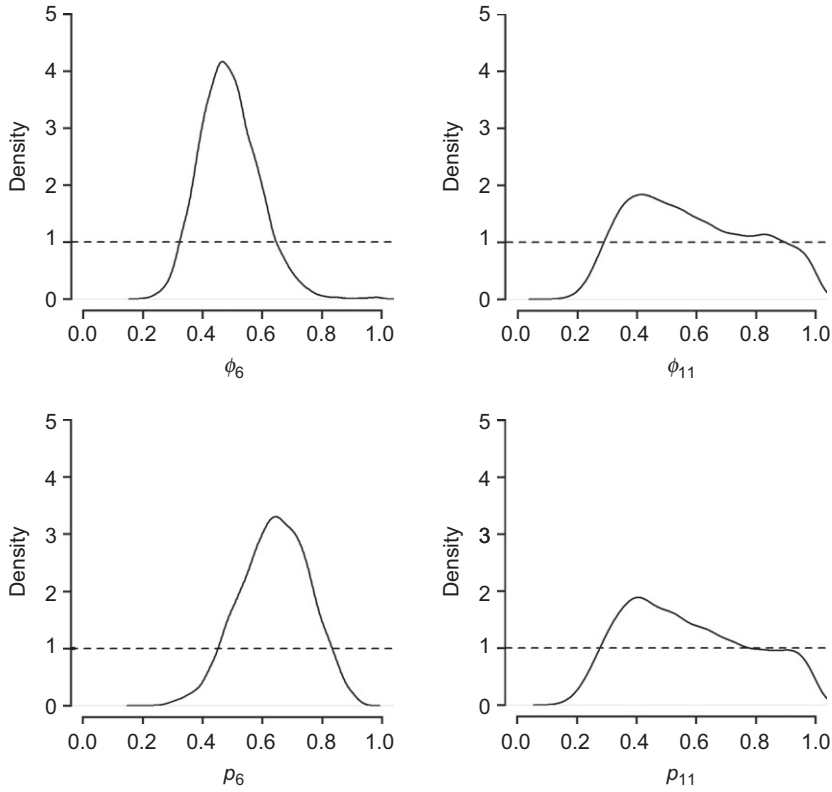

FIGURE 7.8 Posterior density plots of the sixth and the last survival and recapture probabilities. The dotted line shows the prior density. The last parameters (ϕ_{11} and p_{11}) are not separately identifiable.

parameters are not identifiable. Gimenez et al. (2009b) developed a quantitative guideline based on the degree of overlap between posterior and prior to decide when a parameter is identifiable. Note that when the year effects are not fixed as above, but random, the problem of nonidentifiability disappears because information from survival and recapture from the complete data set is used to estimate the last parameters. You may want to try this!

In any analysis of capture–recapture models (or actually, of *any* model), you should be aware that some parameters might not be estimable, although WinBUGS (or another software) may give you estimates for all parameters (Lunn et al., 2010). Obviously, no inference can be made about nonidentifiable parameters. This challenge is even greater for multistate capture–recapture models (see Chapter 9).

7.10 FITTING THE CJS TO DATA IN THE M-ARRAY FORMAT: THE MULTINOMIAL LIKELIHOOD

7.10.1 Introduction

So far we have analyzed the individual capture-histories using a state-space formulation of the CJS model. This is a very general framework within which a multitude of different kinds of models can be formulated, but it comes at a computational cost. As all capture-histories are analyzed individually, a loop over all individuals is necessary. In addition, every unknown latent state (e.g., individual survival) needs to be estimated. Capture–recapture data can, however, also be summarized in the so-called m-array (Burnham et al., 1987). The CJS model is then fitted using a multinomial likelihood. This has the advantage of much faster computation, but the disadvantage of reduced flexibility in the modeling. In particular, models with individual effects can no longer be fitted.

We first introduce the m-array format, by considering the following example—we have capture-histories of seven individuals:

```
1 0 1 0
1 1 0 0
1 1 0 0
1 0 0 0
0 1 1 1
0 1 0 0
0 0 1 0
```

The m-array tabulates the number of individuals released at one occasion that are next recaptured on each subsequent occasion. It is a triangular matrix, in which rows refer to release occasions and columns refer

to recapture occasions. An additional column tallies up the individuals that are not recaptured. To create the m-array, the capture-histories of all individuals are broken into fragments. The number of captures equals the number of fragments. Each fragment considers the last release occasion and the next recapture occasion. For example, the capture-history `[1 0 1 0]` is broken into the two fragments `[1 0 1 0]` and `[0 0 1 0]`. The first fragment shows that the individual was released at occasion 1 and first recaptured at occasion 3. The second fragment shows that the individual was released on occasion 3 and was never recaptured. The m-array for the seven capture-histories above is:

	Recapture Occasion			
Release Occasion	**2**	**3**	**4**	**Never Recaptured**
1	**2**	**1**	**0**	**1**
2	–	**1**	**0**	**3**
3	–	–	**1**	**2**

Fitting the CJS model to the data using the m-array implicitly assumes the absence of any individual effects on survival and recapture probabilities. By summarizing the data in this form, it is evident that effects of individual covariates cannot be fitted because the capture-histories of the individuals are broken up. Age as a special class of individual covariate can be considered but requires a different format of the m-array (see Section 7.10.3). Otherwise, all the information that originally was included in the individual capture-histories is kept; it is just summarized in the form of minimal sufficient statistics.

The following R function converts capture-histories into the m-array format.

```
# Function to create a m-array based on capture-histories (CH)
marray <- function(CH){
    nind <- dim(CH)[1]
    n.occasions <- dim(CH)[2]
    m.array <- matrix(data = 0, ncol = n.occasions+1, nrow =
      n.occasions)

    # Calculate the number of released individuals at each time period
    for (t in 1:n.occasions){
        m.array[t,1] <- sum(CH[,t])
        }
    for (i in 1:nind){
        pos <- which(CH[i,]!=0)
        g <- length(pos)
```

```
    for (z in 1:(g-1)){
        m.array[pos[z],pos[z+1]] <- m.array[pos[z],pos[z+1]] + 1
        } #z
    } #i

# Calculate the number of individuals that is never recaptured
for (t in 1:n.occasions){
    m.array[t,n.occasions+1] <- m.array[t,1] -
       sum(m.array[t,2:n.occasions])
    }
out <- m.array[1:(n.occasions-1),2:(n.occasions+1)]
return(out)
}
```

The expected values of the entries of the m-array are given based on the underlying model parameters (ϕ_t and p_t) and the number of released individuals. These define the cell probabilities of the multinomial distributions for each release occasion.

Release Occasion	Recaptured at Occasion 2	3	4	Never Recaptured
1	$\phi_1 p_1$	$\phi_1(1-p_1)\phi_2 p_2$	$\phi_1(1-p_1)\phi_2(1-p_2)\phi_3 p_3$	$1-\phi_1 p_1-\phi_1(1-p_1)\phi_2 p_2-\phi_1(1-p_1)\phi_2(1-p_2)\phi_3 p_3 = 1-\Sigma$(rel. occ 1)
2	0	$\phi_2 p_2$	$\phi_2(1-p_2)\phi_3 p_3$	$1-\phi_2 p_2-\phi_2(1-p_2)\phi_3 p_3 = 1-\Sigma$(rel. occ 2)
3	0	0	$\phi_3 p_3$	$1-\phi_3 p_3 = 1-\Sigma$(rel. occ 3)

Note: The entry in cell (1,3) is the product $\phi_1(1-p_1)\phi_2 p_2$ (and likewise for the other cells).

7.10.2 Time-Dependent Models

The rows of the observed m-array data follow a multinomial distribution with index equal to the number of released individuals at each occasion and the cell probabilities that are functions of survival and recapture parameters, as shown in the table above. Fitting this model in BUGS is straightforward: essentially, we only need to define the cell probabilities of the m-array.

Using the m-array formulation of the CJS model, it is also quite easy to assess the fit of the model, that is, to compute a Bayesian p-value based on the posterior predictive distribution of a goodness-of-fit (GOF) statistic (see Gelman et al., 1996, 2004; Section 12.3). This technique for GOF assessment is also called posterior predictive checking because its rationale is based on a comparison of data simulated (predicted) under the model, and the actual data set that is analyzed using that

model. Simulated data sets under a model are obtained easily as part of the MCMC updating from the posterior predictive distribution of the data. Usually, some discrepancy measure is calculated that measures how "far apart" the data are from their expected values under the model. Often, omnibus test statistics such as chi-squared are used as a discrepancy measure, but other statistics may be chosen to specifically highlight how well a model fits the data in some particular manner, for instance, how well it describes extreme values (Gelman et al., 1996). This discrepancy measure is calculated for both the simulated and the actual data set. The values of both discrepancy measures change at each iteration of the MCMC simulation algorithm because the parameter values change with each iteration as well and they are used both to generate a replicate data set and to compute the expected values for the data. At the end of the posterior sampling, one has as many draws from the posterior distribution of the chosen discrepancy measure for the simulated (perfect) data sets as for the actual data sets. The simulated data sets are "perfect" in the sense that they were generated under exactly the same model that is used for parameter estimation in the observed data and using the exact parameter values obtained in that analysis. The posterior draws of the discrepancy measure for the replicate data, therefore, provide the reference distribution for the discrepancy measure under the null hypothesis that the model fits our data. The proportion of times that the discrepancy measure for the simulated data sets is more extreme than that for the actual data set is called a Bayesian *p*-value. Under the null hypothesis that the model in question is the data-generating model, this should happen about 50% of times; hence, Bayesian *p*-values close to 0 or 1 are suspicious. A graph of the values of the discrepancy measure from the replicate data sets plotted against those for the actual data sets may be even more informative than the scalar Bayesian *p*-value to point out ways how a model may not fit. The value of the *p*-value represents the proportion of points that lie above the 1:1 line of equality.

Bayesian *p*-values have been criticized for several reasons. First, they use the data twice (once, to generate replicate data sets and then to compute the expected data and compare that with both the replicate and the actual data sets). They may thus not be strict enough and not reject often enough the hypothesis of a fitting model. Second, it is unclear what value of a Bayesian *p*-value represents a good fit. For instance, there would be no objective way of saying that values outside of the interval (0.05, 0.95) represent models that do not fit the data. Thus, Bayesian *p*-values are a descriptive technique only. And finally, the rationale underlying a Bayesian *p*-value is intrinsically frequentist: Learning from the data is not limited to the information content in the actual data set but instead based on hypothetical replicate data sets as well. This may be offensive to hardcore Bayesians who adhere to the so-called likelihood principle (Lindley, 2006),

which says that all information about a data set is contained in the likelihood function. Our own position in this respect is pragmatic: We like Bayesian p-values as a simple and very flexible way of pointing out ways in which a model may not fit a data set.

In the current example of a survival analysis, we could not test the GOF of a state-space model for binary responses (observed vs. not observed). The reason for this is that discrepancy measures such as the deviance are uninformative about model fit for binary responses (McCullagh and Nelder, 1989). GOF can, however, be assessed for some summary of binary responses and the m-array represents just one such summary. So here now, we create replicate data (i.e., m-arrays, e_{ij}), and compare the observed (x_{ij}) and the expected m-arrays using a discrepancy measure. We could use the χ^2-dispcrepancy as in Chapter 12, but instead follow Brooks et al. (2000b) and use the Freeman-Tukey statistic ($D = \Sigma(x_{ij}^{1/2} - e_{ij}^{1/2})^2$). It makes unnecessary to pool cells with small expected values. The Freeman–Tukey statistic is computed for the observed and simulated data.

We use the data as created in Section 7.9 to illustrate the use of the model. The following code fits the CJS model using the multinomial likelihood and includes the posterior predictive check.

```
# Specify model in BUGS language
sink("cjs-mnl.bug")
cat("
model {

# Priors and constraints
for (t in 1:(n.occasions-1)){
   phi[t] ~ dunif(0, 1)          # Priors for survival
   p[t] ~ dunif(0, 1)            # Priors for recapture
   }

# Define the multinomial likelihood
for (t in 1:(n.occasions-1)){
   marr[t,1:n.occasions] ~ dmulti(pr[t, ], r[t])
   }

# Calculate the number of birds released each year
for (t in 1:(n.occasions-1)){
   r[t] <- sum(marr[t, ])
   }

# Define the cell probabilities of the m-array
# Main diagonal
for (t in 1:(n.occasions-1)){
   q[t] <- 1-p[t]                # Probability of non-recapture
   pr[t,t] <- phi[t]*p[t]
   # Above main diagonal
   for (j in (t+1):(n.occasions-1)){
      pr[t,j] <- prod(phi[t:j])*prod(q[t:(j-1)])*p[j]
      } #j
```

```
    # Below main diagonal
    for (j in 1:(t-1)){
        pr[t,j] <- 0
        } #j
    } #t
# Last column: probability of non-recapture
for (t in 1:(n.occasions-1)){
    pr[t,n.occasions] <- 1-sum(pr[t,1:(n.occasions-1)])
    } #t

# Assess model fit using Freeman-Tukey statistic
# Compute fit statistics for observed data
for (t in 1:(n.occasions-1)){
    for (j in 1:n.occasions){
        expmarr[t,j] <- r[t]*pr[t,j]
        E.org[t,j] <- pow((pow(marr[t,j], 0.5)-pow(expmarr[t,j],
           0.5)), 2)
        } #j
    } #t

# Generate replicate data and compute fit stats from them
for (t in 1:(n.occasions-1)){
    marr.new[t,1:n.occasions] ~ dmulti(pr[t, ], r[t])
    for (j in 1:n.occasions){
        E.new[t,j] <- pow((pow(marr.new[t,j], 0.5)-pow(expmarr[t,j],
           0.5)), 2)
        } #j
    } #t
fit <- sum(E.org[,])
fit.new <- sum(E.new[,])
}
",fill = TRUE)
sink()

# Create the m-array from the capture-histories
marr <- marray(CH)

# Bundle data
bugs.data <- list(marr = marr, n.occasions = dim(marr)[2])

# Initial values
inits <- function(){list(phi = runif(dim(marr)[2]-1, 0, 1),
   p = runif(dim(marr)[2]-1, 0, 1))}

# Parameters monitored
parameters <- c("phi", "p", "fit", "fit.new")

# MCMC settings
ni <- 10000
nt <- 3
nb <- 5000
nc <- 3

# Call WinBUGS from R (BRT 1 min)
cjs <- bugs(bugs.data, inits, parameters, "cjs-mnl.bug", n.chains =
   nc, n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE,
   bugs.directory = bugs.dir, working.directory = getwd())
```

```
print(cjs, digits = 3)

           mean    sd  2.5%   25%   50%   75%  97.5%  Rhat n.eff
phi[1]    0.632 0.167 0.331 0.505 0.621 0.754  0.960 1.001  5000
[ ... ]
p[11]     0.577 0.206 0.261 0.406 0.547 0.742  0.968 1.006   380
fit      10.563 1.674 7.773 9.378 10.400 11.570 14.320 1.001  5000
fit.new 12.671 2.744 8.095 10.720 12.420 14.330 18.830 1.002  2400
```

The model converges quickly and the MCMC samples are obtained in a short time (to compare, you may use the same data and run the corresponding state-space model of Section 7.3). The comparison of the discrepancy between the observed and the simulated data (Fig. 7.9) shows that they are similar, suggesting that the model is adequate for the data set. This is confirmed by a Bayesian *p*-value of 0.75. For more discussion about checking of capture–recapture models, see Brooks et al. (2000a, 2000b) and King et al. (2010).

```
# Evaluation of fit
plot(cjs$sims.list$fit, cjs$sims.list$fit.new, xlab = "Discrepancy
   actual data", ylab = "Discrepancy replicate data", las = 1,
ylim = c(5, 25), xlim = c(5, 25), bty ="n")
abline(0, 1, col = "black", lwd = 2)
mean(cjs$sims.list$fit.new > cjs$sims.list$fit)
```

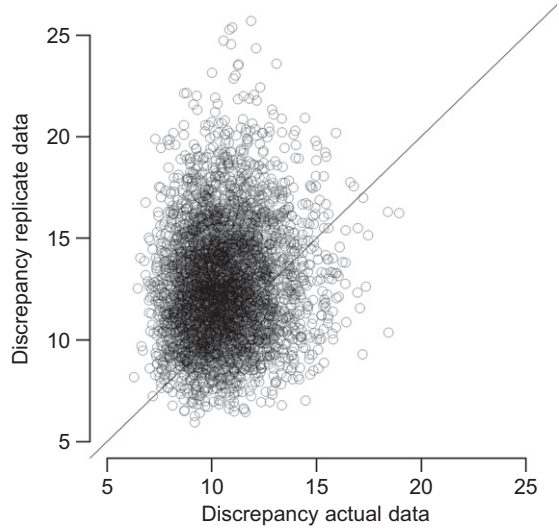

FIGURE 7.9 Posterior predictive check of model fit by a scatter plot of the discrepancy measure for replicate (simulated) versus actual (observed) data in a CJS model. The Bayesian *p*-value is the proportion of points above the 1:1 line.

Construction of models with time effects (fixed or random) using the multinomial likelihood requires changes in the "Priors and constraints" part of the model code, in exactly the same way as we have introduced for the state-space formulation. However, the formulation of models with groups is slightly different. We need to create an m-array for each group and to write a separate likelihood (with different parameters) for each data set. Once this is done, we can constrain the group-specific parameters in the same way as with models using the state-space formulation (i.e., we can regard groups as fixed or as random, and we can combine them with time effects). See exercise 2 in Section 7.13 for an example.

7.10.3 Age-Dependent Models

Models with age-dependent survival fitted with the multinomial likelihood need some adaptations (m-array, analyzing code), and we show this in detail using an example. We look at the situation in which young and adult little owls are marked and assume that survival in the first year of life (from age 0 to age 1 year) is different from survival in subsequent age classes (from age 1 onward). We first start with the simulation of the data. We will create two data sets, one for individuals marked as juveniles, and another for individuals marked as adults.

```
# Define parameter values
n.occasions <- 12                       # Number of capture occasions
marked.j <- rep(200, n.occasions-1)     # Annual number of newly marked
                                          juveniles
marked.a <- rep(30, n.occasions-1)      # Annual number of newly marked
                                          adults
phi.juv <- 0.3                          # Juvenile annual survival
phi.ad <- 0.65                          # Adult annual survival
p <- rep(0.5, n.occasions-1)            # Recapture
phi.j <- c(phi.juv, rep(phi.ad,n.occasions-2))
phi.a <- rep(phi.ad, n.occasions-1)

# Define matrices with survival and recapture probabilities
PHI.J <- matrix(0, ncol = n.occasions-1, nrow = sum(marked.j))
for (i in 1:(length(marked.j)-1)){
    PHI.J[(sum(marked.j[1:i])-
    marked.j[i]+1):sum(marked.j[1:i]),i:(n.occasions-1)] <-
    matrix(rep(phi.j[1:(n.occasions-i)], marked.j[i]),
    ncol = n.occasions-i, byrow = TRUE)
    }
P.J <- matrix(rep(p, n.occasions*sum(marked.j)), ncol =
   n.occasions-1, nrow = sum(marked.j), byrow = TRUE)
PHI.A <- matrix(rep(phi.a, sum(marked.a)), ncol = n.occasions-1,
   nrow = sum(marked.a), byrow = TRUE)
P.A <- matrix(rep(p, sum(marked.a)), ncol = n.occasions-1,
   nrow = sum(marked.a), byrow = TRUE)
```

```
# Apply simulation function
CH.J <- simul.cjs(PHI.J, P.J, marked.j)
CH.A <- simul.cjs(PHI.A, P.A, marked.a)
```

Next, we create two m-arrays, one for juveniles and another for adults. The difficulty is that whenever an individual initially marked as a juvenile is recaptured, it has become an adult. Thus, it must be "released" in the m-array of the individuals initially marked as adults. To achieve this goal, we first split the capture-histories of individuals marked as juveniles based on whether or not they were ever recaptured (recaptured at least once: `CH.J.R`, never recaptured: `CH.J.N`). The first capture of `CH.J.R` is then removed, the resulting capture-histories added to the capture-histories of the individuals marked as adults and the m-array computed. Next, all recaptures after the first recapture of the original `CH.J.R` matrix are removed and the m-array computed. Because all these individuals are released as adults, the last columns of the m-array summarizing the number of individuals never recaptured have to be set to zero. Finally, we create the m-array for `CH.J.N` and add it to the previous m-array. The following code performs these data manipulations.

```
cap <- apply(CH.J, 1, sum)
ind <- which(cap >= 2)
CH.J.R <- CH.J[ind,]      # Juvenile CH recaptured at least once
CH.J.N <- CH.J[-ind,]     # Juvenile CH never recaptured

# Remove first capture
first <- numeric()
for (i in 1:dim(CH.J.R)[1]){
    first[i] <- min(which(CH.J.R[i,]==1))
    }
CH.J.R1 <- CH.J.R
for (i in 1:dim(CH.J.R)[1]){
    CH.J.R1[i,first[i]] <- 0
    }

# Add grown-up juveniles to adults and create m-array
CH.A.m <- rbind(CH.A, CH.J.R1)
CH.A.marray <- marray(CH.A.m)

# Create CH matrix for juveniles, ignoring subsequent recaptures
second <- numeric()
for (i in 1:dim(CH.J.R1)[1]){
    second[i] <- min(which(CH.J.R1[i,]==1))
    }
CH.J.R2 <- matrix(0, nrow = dim(CH.J.R)[1], ncol = dim(CH.J.R)[2])
for (i in 1:dim(CH.J.R)[1]){
    CH.J.R2[i,first[i]] <- 1
    CH.J.R2[i,second[i]] <- 1
    }

# Create m-array for these
CH.J.R.marray <- marray(CH.J.R2)
```

```
# The last column ought to show the number of juveniles not recaptured
   again and should all be zeros, since all of them are released as adults
CH.J.R.marray[,dim(CH.J)[2]] <- 0
# Create the m-array for juveniles never recaptured and add it to the
   previous m-array
CH.J.N.marray <- marray(CH.J.N)
CH.J.marray <- CH.J.R.marray + CH.J.N.marray
```

Now we write the BUGS code for the age-dependent model. We specify two component likelihoods, one for the m-array of adults and another for the m-array of juveniles. The code for adults is exactly the same as before (Section 7.10.2), but the code for juveniles has some twists. Here, the first survival for each release cohort (juvenile survival) is different from subsequent survival (which is that of adults).

```
# Specify model in BUGS language
sink("cjs-mnl-age.bug")
cat("
model {
# Priors and constraints
for (t in 1:(n.occasions-1)){
    phi.juv[t] <- mean.phijuv
    phi.ad[t] <- mean.phiad
    p[t] <- mean.p
    }
mean.phijuv ~ dunif(0, 1)          # Prior for mean juv. survival
mean.phiad ~ dunif(0, 1)           # Prior for mean ad. survival
mean.p ~ dunif(0, 1)               # Prior for mean recapture
# Define the multinomial likelihood
for (t in 1:(n.occasions-1)){
    marr.j[t,1:n.occasions] ~ dmulti(pr.j[t,], r.j[t])
    marr.a[t,1:n.occasions] ~ dmulti(pr.a[t,], r.a[t])
    }
# Calculate the number of birds released each year
for (t in 1:(n.occasions-1)){
    r.j[t] <- sum(marr.j[t,])
    r.a[t] <- sum(marr.a[t,])
    }
# Define the cell probabilities of the m-arrays
# Main diagonal
for (t in 1:(n.occasions-1)){
    q[t] <- 1-p[t]           # Probability of non-recapture
    pr.j[t,t] <- phi.juv[t]*p[t]
    pr.a[t,t] <- phi.ad[t]*p[t]
    # Above main diagonal
    for (j in (t+1):(n.occasions-1)){
        pr.j[t,j] <- phi.juv[t]*prod(phi.ad[(t+1):j])*prod(q[t:
           (j-1)])*p[j]
        pr.a[t,j] <- prod(phi.ad[t:j])*prod(q[t:(j-1)])*p[j]
        } #j
```

```
    # Below main diagonal
    for (j in 1:(t-1)){
        pr.j[t,j] <- 0
        pr.a[t,j] <- 0
        } #j
    } #t
# Last column: probability of non-recapture
for (t in 1:(n.occasions-1)){
    pr.j[t,n.occasions] <- 1-sum(pr.j[t,1:(n.occasions-1)])
    pr.a[t,n.occasions] <- 1-sum(pr.a[t,1:(n.occasions-1)])
    } #t
}
",fill = TRUE)
sink()

# Bundle data
bugs.data <- list(marr.j = CH.J.marray, marr.a = CH.A.marray,
   n.occasions = dim(CH.J.marray)[2])

# Initial values
inits <- function(){list(mean.phijuv = runif(1, 0, 1), mean.phiad =
   runif(1, 0, 1), mean.p = runif(1, 0, 1))}

# Parameters monitored
parameters <- c("mean.phijuv", "mean.phiad", "mean.p")

# MCMC settings
ni <- 3000
nt <- 3
nb <- 1000
nc <- 3

# Call WinBUGS from R (BRT <1 min)
cjs.2 <- bugs(bugs.data, inits, parameters, "cjs-mnl-age.bug",
   n.chains = nc, n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE,
   bugs.directory = bugs.dir, working.directory = getwd())
```

Convergence is achieved quickly; 3000 iterations with a burnin of 1000 are sufficient. Plotting the posterior distributions shows parameter estimates that resemble well the values used to simulate the data (Fig. 7.10).

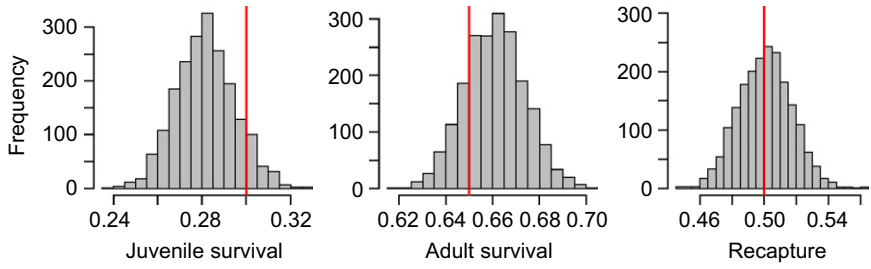

FIGURE 7.10 Posterior distributions of juvenile and adult survival and of recapture probability. Red lines indicate the values used to generate the data set.

```
par(mfrow = c(1, 3), las = 1)
hist(cjs.2$sims.list$mean.phijuv, nclass = 30, col = "gray", main = "",
   xlab = "Juvenile survival", ylab = "Frequency")
abline(v = phi.juv, col = "red", lwd = 2)
hist(cjs.2$sims.list$mean.phiad, nclass = 30, col = "gray", main = "",
   xlab = "Adult survival", ylab = "")
abline(v = phi.ad, col = "red", lwd = 2)
hist(cjs.2$sims.list$mean.p, nclass = 30, col = "gray", main = "",
   xlab = "Recapture", ylab = "")
abline(v = p[1], col = "red", lwd = 2)
```

We assumed that recapture probability was not dependent on age because all birds are >1 year old when they are first recaptured. Sometimes, however, it may be useful to fit age effects for recapture probability. Often, young individuals do not reproduce as successfully as adults. If individuals can only be captured when reproducing, this can result in a lower recapture probability of young individuals.

The model could also be extended to include more age classes. In principle, the number of m-arrays is equal to the number of age classes in the model. However, careful bookkeeping is required to fit these models. The age of each individual at each recapture has to be evaluated, and afterward the individual is "released" in the m-array in the corresponding age class. M-arrays are specified for each age class in the BUGS model code, and all of them have an age structure with the exception of the m-array for the oldest age class. Cell probabilities of the m-array of the second oldest age class have an age structure with two classes, that of the third-oldest age class an age structure with three classes, and so forth.

7.11 ANALYSIS OF A REAL DATA SET: SURVIVAL OF FEMALE LEISLER'S BATS

Leisler's bat (Fig. 7.11) is a medium-sized bat species that forms nursery colonies in cavities in woodlands and is widespread throughout Europe. Northern populations migrate to the Mediterranean in winter. Wigbert Schorcht and his colleagues studied a population of Leisler's bat in Thuringia (Germany) from 1989 to 2008. They placed bat boxes in a forest and regularly captured individuals in them. The capture–recapture data have been extensively analyzed using CJS models fitted in a frequentist framework (Schorcht et al., 2009). Here, we analyze a subset of these data consisting of 181 adult females that were born in the study area. Females are highly philopatric and thus our estimate of apparent survival is likely close to true survival. Some initial modeling suggested that adult survival was subject to strong temporal variation, whereas recapture probabilities were constant over time (Schorcht et al., 2009). Our interest here is to estimate mean annual survival as well as its temporal variance.

FIGURE 7.11 Leisler's bat (*Nyctalus leisleri*) (Photograph D. Nill).

We will therefore fit a model denoted by (ϕ_t, p.). We first performed a frequentist GOF test with program U-CARE (Choquet et al., 2001). The test assesses the fit of the time-dependent CJS model (ϕ_t, p_t) and didn't show any indication of lack of fit ($\chi^2_{49} = 41.38$, $P = 0.77$). Yet, because we fit a model with random year effects, we want to perform also a posterior predictive check to evaluate the goodness-of-fit and estimate a Bayesian p-value for that model. The data are already summarized in the m-array format, thus we will use the multinomial likelihood to fit the model.

```
m.leisleri <- matrix(c(4,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,
0,5,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,3,
0,0,9,2,0,0,0,0,0,0,0,0,0,0,0,0,0,3,
0,0,0,10,2,0,0,0,0,0,0,0,0,0,0,0,0,5,
0,0,0,0,10,2,1,0,0,0,0,0,0,0,0,0,0,6,
0,0,0,0,0,0,15,0,0,0,0,0,0,0,0,0,0,0,6,
0,0,0,0,0,0,0,11,2,0,1,0,0,0,0,0,0,0,19,
0,0,0,0,0,0,0,0,12,1,1,0,0,0,0,0,0,0,6,
0,0,0,0,0,0,0,0,0,13,2,0,0,0,0,0,0,0,4,
0,0,0,0,0,0,0,0,0,0,14,0,0,0,0,0,0,0,6,
0,0,0,0,0,0,0,0,0,0,0,13,1,0,0,0,1,0,0,8,
0,0,0,0,0,0,0,0,0,0,0,0,15,3,1,0,0,0,0,12,
0,0,0,0,0,0,0,0,0,0,0,0,0,12,4,0,1,0,0,7,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,19,2,0,0,0,3,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,28,1,0,0,4,
```

```
0,0,0,0,0,0,0,0,0,0,0,0,0,0,22,7,2,21,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,12,2,21,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,14,18), ncol = 19, nrow = 18,
   byrow = TRUE)
```

The BUGS code poses no additional difficulties; we merely have to add the hierarchical extension to the multinomial model to account for random year effects. This extension assumes that the annual survival probabilities are random draws from a normal distribution whose mean is the logit of mean survival and a variance. This variance (`sigma2` in the code below) is the temporal variance of survival on the logit scale. In case we prefer to express the temporal variance on the probability scale, we also have a parameter called `sigma2.real`.

```
# Specify model in BUGS language
sink("cjs-mnl-ran.bug")
cat("
model {

# Priors and constraints
for (t in 1:(n.occasions-1)){
    logit(phi[t]) <- mu + epsilon[t]
    epsilon[t] ~ dnorm(0, tau)
    p[t] <- mean.p
    }
mean.phi ~ dunif(0, 1)                # Prior for mean survival
mu <- log(mean.phi / (1-mean.phi))    # Logit transformation
sigma ~ dunif(0, 5)                   # Prior for standard deviation
tau <- pow(sigma, -2)
sigma2 <- pow(sigma, 2)
# Temporal variance on real scale
sigma2.real <- sigma2 * pow(mean.phi, 2) * pow((1-mean.phi), 2)
mean.p ~ dunif(0, 1)                  # Prior for mean recapture

# Define the multinomial likelihood
for (t in 1:(n.occasions-1)){
    marr[t,1:n.occasions] ~ dmulti(pr[t,], r[t])
    }

# Calculate the number of birds released each year
for (t in 1:(n.occasions-1)){
    r[t] <- sum(marr[t,])
    }

# Define the cell probabilities of the m-array:
# Main diagonal
for (t in 1:(n.occasions-1)){
    q[t] <- 1-p[t]
    pr[t,t] <- phi[t]*p[t]
    # Above main diagonal
    for (j in (t+1):(n.occasions-1)){
        pr[t,j] <- prod(phi[t:j])*prod(q[t:(j-1)])*p[j]
        } #j
```

```
    # Below main diagonal
    for (j in 1:(t-1)){
        pr[t,j]<-0
        } #j
    } #t
# Last column: probability of non-recapture
for (t in 1:(n.occasions-1)){
    pr[t,n.occasions] <- 1-sum(pr[t,1:(n.occasions-1)])
    } # t

# Assess model fit using Freeman-Tukey statistic

# Compute fit statistics for observed data
for (t in 1:(n.occasions-1)){
    for (j in 1:n.occasions){
        expmarr[t,j] <- r[t]*pr[t,j]
        E.org[t,j] <- pow((pow(marr[t,j], 0.5)-pow(expmarr[t,j],
           0.5)), 2)
        }
    }

# Generate replicate data and compute fit stats from them
for (t in 1:(n.occasions-1)){
    marr.new[t,1:n.occasions] ~ dmulti(pr[t,], r[t])
    for (j in 1:n.occasions){
        E.new[t,j] <- pow((pow(marr.new[t,j], 0.5)-pow(expmarr[t,j],
           0.5)), 2)
        }
    }
fit <- sum(E.org[,])
fit.new <- sum(E.new[,])
}
",fill = TRUE)
sink()

# Bundle data
bugs.data <- list(marr = m.leisleri, n.occasions = dim(m.leisleri)[2])

# Initial values
inits <- function(){list(mean.phi = runif(1, 0, 1), sigma = runif(1, 0,
   5), mean.p = runif(1, 0, 1))}

# Parameters monitored
parameters <- c("phi", "mean.p", "mean.phi", "sigma2", "sigma2.real",
   "fit", "fit.new")

# MCMC settings
ni <- 5000
nt <- 3
nb <- 1000
nc <- 3

# Call WinBUGS from R (BRT 3 min)
leis.result <- bugs(bugs.data, inits, parameters, "cjs-mnl-ran.bug",
   n.chains = nc, n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE,
   bugs.directory = bugs.dir, working.directory = getwd())
```

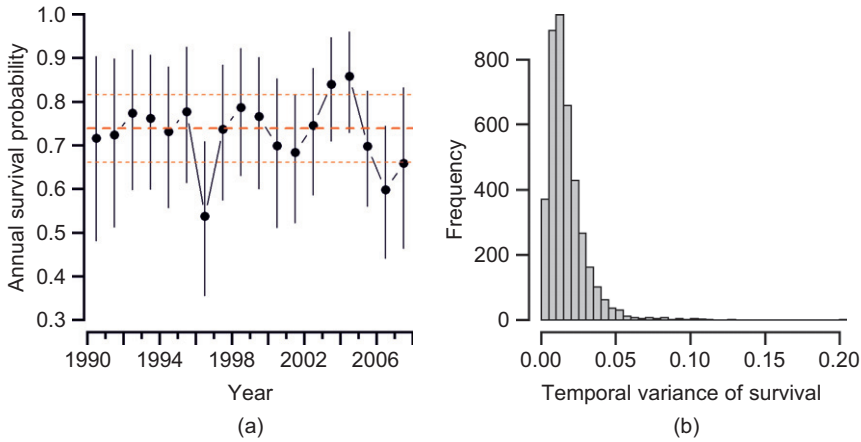

FIGURE 7.12 (a) Annual survival probability of adult female Leisler's bats (closed symbols, with 95% CRIs) and mean survival (red line; with 95% CRI dotted). (b) Posterior distribution of the temporal variance of adult survival.

The Markov chains converge quickly; with just 5000 iterations, we obtain satisfactory Rhat values (all <1.01). Mean annual survival is about 74%. Interestingly, and by chance, the recapture probability is numerically almost identical. Figure 7.12 plots the posterior distributions of the annual and mean survival probabilities as well as of the temporal variance. Annual survival probabilities were similar in most years, but in some, they were unusually low (1996–1997, 2006–2007) or unusually high (2003–2005). A next step in the demographic analysis of this population might be to find out which environmental factor is correlated with the temporal variation in survival, as shown in Section 7.4.3.

```
# Summarize posteriors
print(leis.result, digits = 3)

               mean     sd    2.5%     25%     50%     75%   97.5%  Rhat n.eff
phi[1]        0.716  0.106   0.481   0.653   0.726   0.789   0.904 1.001  4000
[ ... ]
phi[18]       0.658  0.093   0.464   0.600   0.661   0.723   0.832 1.004   590
mean.p        0.747  0.029   0.689   0.728   0.748   0.766   0.800 1.001  4000
mean.phi      0.739  0.038   0.661   0.715   0.739   0.763   0.815 1.003  2700
sigma2        0.467  0.340   0.062   0.235   0.386   0.610   1.341 1.012   390
sigma2.real   0.017  0.013   0.002   0.009   0.014   0.022   0.048 1.013   450
fit          21.047  2.279  17.260  19.410  20.850  22.400  26.279 1.003   830
fit.new      18.950  3.458  12.950  16.450  18.705  21.100  26.370 1.001  4000

# Produce figure of female survival probabilities
par(mfrow = c(1, 2), las = 1, mar=c(4, 4, 2, 2), mgp = c(3, 1, 0))
lower <- upper <- numeric()
T <- dim(m.leisleri)[2]-1
```

```
for (t in 1:T){
    lower[t] <- quantile(leis.result$sims.list$phi[,t], 0.025)
    upper[t] <- quantile(leis.result$sims.list$phi[,t], 0.975)
    }
plot(y = leis.result$mean$phi, x = (1:T)+0.5, type = "b", pch = 16, ylim =
   c(0.3, 1), ylab = "Annual survival probability", xlab = "", axes = F)
axis(1, at = seq(1,(T+1),2), labels = seq(1990,2008,2))
axis(1, at = 1:(T+1), labels = rep("", T+1), tcl = -0.25)
axis(2, las = 1)
mtext("Year", 1, line = 2.25)
segments((1:T)+0.5, lower, (1:T)+0.5, upper)
segments(1, leis.result$mean$mean.phi, T+1, leis.result$mean$mean.phi,
   lty = 2, col = "red", lwd = 2)
segments(1, quantile(leis.result$sims.list$mean.phi,0.025), T+1,
   quantile(leis.result$sims.list$mean.phi, 0.025), lty = 2, col = "red")
segments(1, quantile(leis.result$sims.list$mean.phi, 0.975), T+1,
   quantile(leis.result$sims.list$mean.phi, 0.975), lty = 2, col = "red")
hist(leis.result$sims.list$sigma2.real, nclass = 45, col = "gray",
   main = "", las = 1, xlab = "")
mtext("Temporal variance of survival", 1, line = 2.25)
```

The GOF evaluation of the model shows a good fit (Fig. 7.13) with a Bayesian *p*-value of 0.27. The result is thus qualitatively the same as the GOF test performed in the frequentist framework (see above). Yet, the frequentist goodness-of-fit test evaluates the model with fixed year effects

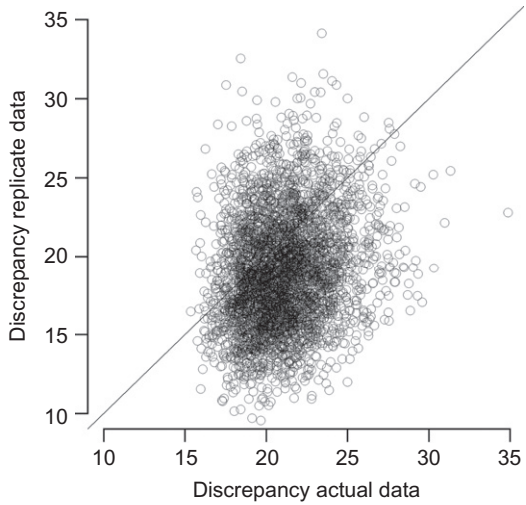

FIGURE 7.13 Scatter plot of replicate (simulated) versus actual (observed) discrepancy measures of model for female Leisler's bats. The Bayesian *p*-value is the proportion of points above the 1:1 equality line.

(ϕ_t, p_t), whereas the Bayesian test evaluates the model actually used for the estimation, that is, a model with random year effects on survival and constant recapture probabilities ($\phi_{\underline{t}}$, $p_.$).

```
# Evaluation of fit
plot(leis.result$sims.list$fit, leis.result$sims.list$fit.new,
   main = "", xlab = "Discrepancy actual data", ylab = "Discrepancy
   replicate data", las = 1, ylim = c(10, 35), xlim = c(10, 35), frame = FALSE)
abline(0, 1, col = "black")
```

7.12 SUMMARY AND OUTLOOK

This chapter presents models of the Cormack–Jolly–Seber (CJS) class for analysis of capture–recapture data in the Bayesian framework to estimate probabilities of survival and recapture. We introduced two different approaches, based on a state-space or a multinomial likelihood. The state-space likelihood has the advantage that it is very flexible and especially enables us to fit models with individual effects, including random effects. The downside is that the Markov chains of these models take much longer per iteration and mix less well, resulting sometimes in a big computational burden. With the multinomial likelihood, we cannot fit models with individual effects, but otherwise the same models are possible as under a state-space likelihood. Use of the multinomial likelihood results in quicker updates and better mixing of the chains. We therefore recommend using the multinomial likelihood unless individual effects need to be fitted.

This chapter contains very important material for the broad class of capture–recapture models because we have introduced several key concepts. We have shown how we can model survival (and recapture) along the "time" as well as along the "individual" axes using GLM formulations (see also Chapter 6 for the analoguous concept to model detection probability). The corresponding models can have fixed or random effects, and there is great flexibility in combining them. In addition, we have introduced age-dependent models, which are a specific combination of effects along the time and the individual axes. We also have introduced goodness-of-fit testing using posterior predictive checks (Bayesian *p*-values). All these key concepts can be applied to the capture–recapture models in later chapters and indeed in an analogous way to all the models in the rest of the book.

Capture–recapture data could also be analyzed with the Jolly–Seber model (JS model; Williams et al., 2002), which is similar to the Cormack–Jolly–Seber model of this chapter. The main difference is that the CJS model conditions on first capture, whereas the JS model describes the complete capture-history. This means that the zeros before the first

capture are not modeled in the CJS model, but they are in the JS model. The latter allows the estimation of additional parameters such as recruitment and population size, at the expense of additional assumptions. We describe the JS model in Chapter 10. Further extensions to this class of model include the robust design model (Kendall et al., 1997; Schofield et al., 2009), reverse-time modeling to estimate population growth rate (Pradel, 1996), or the relative contribution of survival and recruitment to population growth (Nichols et al., 2000). These could be implemented in WinBUGS as well.

This chapter was the first to introduce models for estimation of survival and related demographic parameters. Much of the material (e.g., m-array, state-space likelihood, random and fixed effects in survival) also carries over to similar models in Chapters 8–10. In Chapter 11, we will combine the CJS model with other models into an integrated population model.

7.13 EXERCISES

1. For reasons of greater generality, we always specify CJS models with a likelihood that allows all parameters to potentially vary by individual and time. For a beginner, this may not be the simplest way to fit a CJS model. Consider the constant model in Section 7.3 and adapt the BUGS model code so that we fit that model directly, without constraining the parameter matrices.
2. Simulate capture–recapture data of a species for males and females. The study is conducted for 15 years; the mean survival of males is 0.6 that of females is 0.5, and recapture is 0.4 for both. Assume that each year 30 individuals of each sex are newly marked. Fit the model $\{\phi_{sex}, p\}$ to the data using the multinomial likelihood.
3. Simulate capture–recapture data of a species for males and females. The study is conducted for 10 years, and each year 30 young and 20 adults of each sex are newly marked. The mean survival of young males is 0.3 (0.2 for females) and mean survival of adults of both sexes is 0.7. Further assume that the recapture probability of males is time-dependent [0.5, 0.6, 0.4, 0.4, 0.7, 0.5, 0.8, 0.3, 0.8]. Recapture probability of females varies in parallel to that of the males, it is a bit higher than that of males (difference on the logit scale: 0.3). Analyze these data with the data-generating model.
4. For the model in Section 7.3, do a simulation-based assessment of bias and precision. Generate a data set and then fit the model 500 times (perhaps for smaller sample size to save time) and each time save the estimates. On completion, print out the mean and the standard deviation of the estimates and also plot the distribution of these estimates. Is the estimator from the model biased? Where in the graph

can you see the standard error of the estimates? Are there other methods to check whether a model produces unbiased parameter estimates than simulation?

5. Take the data where survival of young and adult individuals is different (Section 7.7), but where only individuals of exact known age (marked as young) are included. Fit a model, in which survival after the second year changes linearly with increasing age.
6. Simulate data of a study that is running for 15 years, and each year 100 young individuals are marked. Survival in the first year is 0.4 on average with a temporal variability of 0.5 (on the logit scale), survival of older individuals is 0.8 without variability. Recapture probability is 0.6 for all individuals. Analyze these data with the data-generating model using the state-space and the multinomial likelihood.