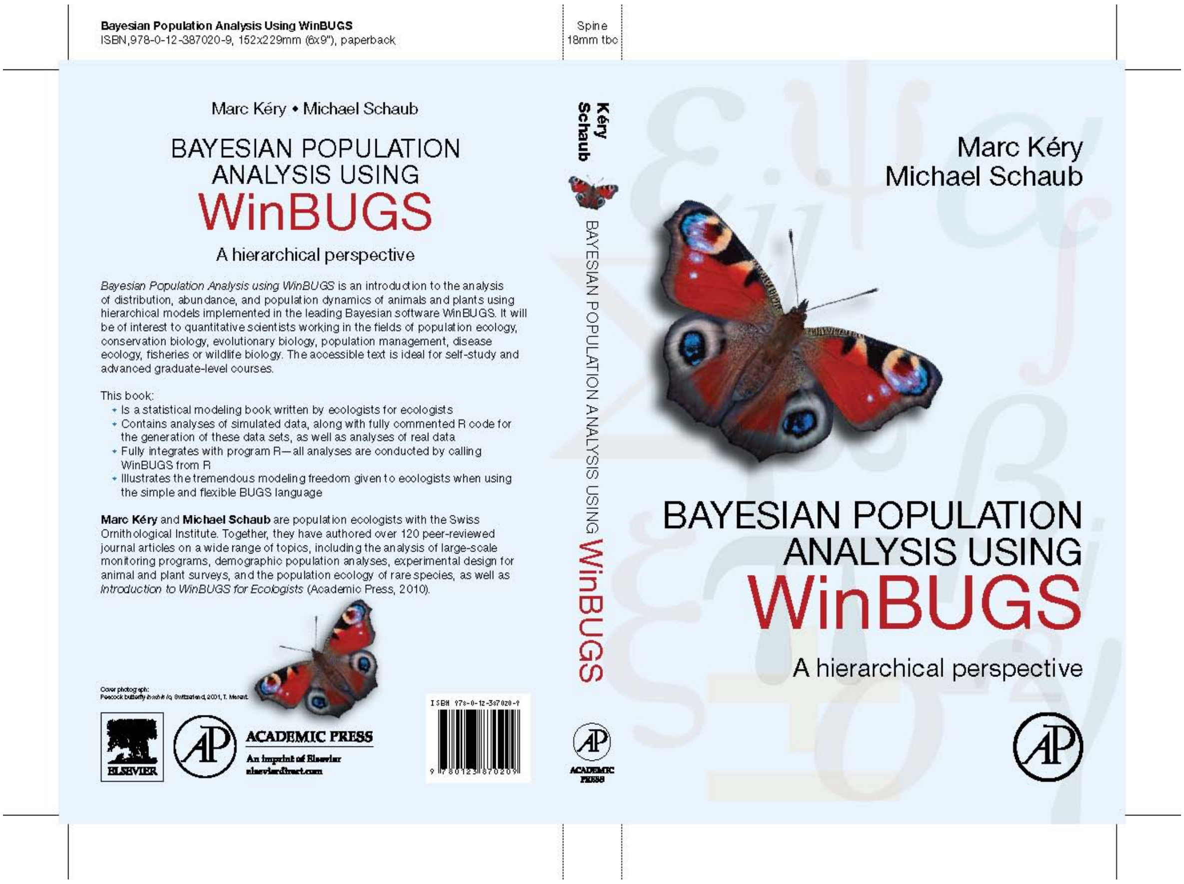
Solutions to the exercises in the book *Bayesian Population Analysis using WinBUGS* (Kéry & Schaub, Academic Press, 2012)



Latest changes: 6 February 2012

This document contains the full and commented solutions of the exercises in the BPA book. We have strived to retain the same layout and programming conventions as in the book itself. At the start of each exercise, we repeat the actual task and then afterwards give the (or rather, a) solution. For some solutions, utility functions described in the book will be necessary. They can be downloaded on www.vogelwarte.ch/web-appendices.html, where you can also download a text version of this document.

Marc & Michael

Contents

Chapter 1

<u>Exercise 1</u>
Task: Detection probability: Convince yourself that very few quantities in nature are ever perfectly detectable. Stand at the window for 1 minute and make a list of all the bird species that you see. Repeat this once or twice, then compare the lists among times and observers. If detection were really perfect (for all species, at all times, for all observers, etc.), then all the lists would be the same for all observers and it would not matter for how long you watched. Essentially you would detect all species instantaneously.

You may conduct that exercise with a quantity of your choice: e.g., the number (or identity) of people in your office hall, the number of people in your bus. Alternatively, you could also count the brands of cars that pass in front of you.

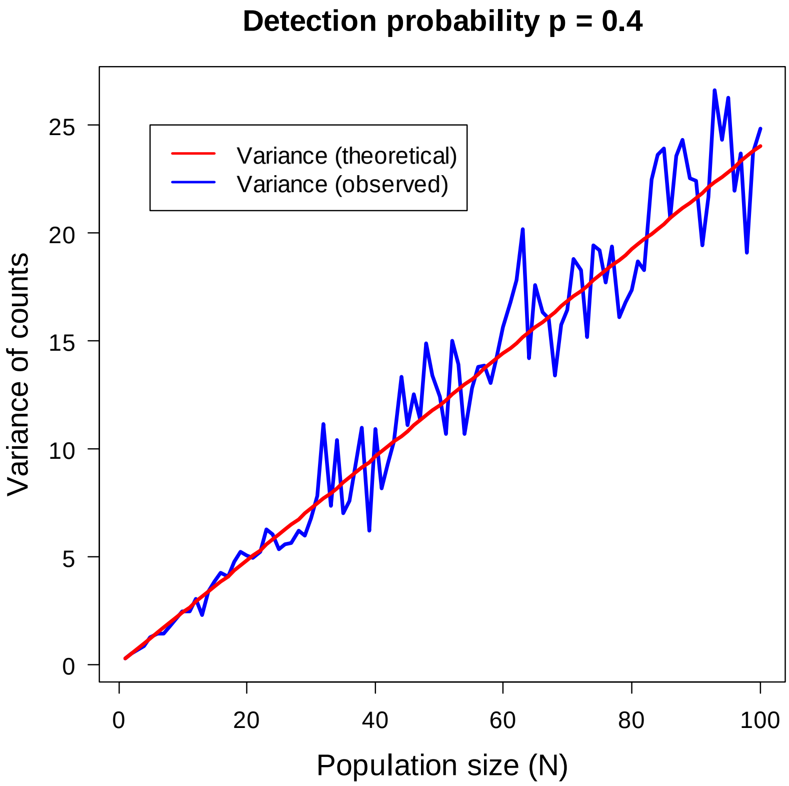
Solution: Just do it.

<u>Exercise 2</u>
Task: Intrinsic variability of counts: It is our experience that ecologists often don't realize that counts vary intrinsically as soon as detection is imperfect. Moreover, counts that vary more are sometimes viewed as being of inferior quality than counts that vary less, or, that the observers producing these counts are better or worse. It is true that, everything else equal, sloppy counts will usually be more variable than those made by a more dedicated observer. However, owing to the mean-variance relationship in binomial counts, two of the most important factors affecting the variability of counts is (a) the number of things available for counting (that is, N) and (b) detection probability p. Produce a plot or a table that makes you better understand these relationships.

Solution: From statistical theory, the variance of a binomial random variable (count) C with index (population size) N and success (detection) probability p is $Np(1-p)$. Hence, counts will vary more when N is big or the product of $p(1-p)$ is big. We convince ourselves of this fact by conducting two little simulations. In the first, we vary N while holding p constant and in the second, we vary p while holding constant N.

First, we assume that we made counts in 100 populations that contain between 1 and 100 sparrows and a constant detection probability of $p = 0.4$. We conduct 100 counts in each population and then plot the variance of those counts against population size.

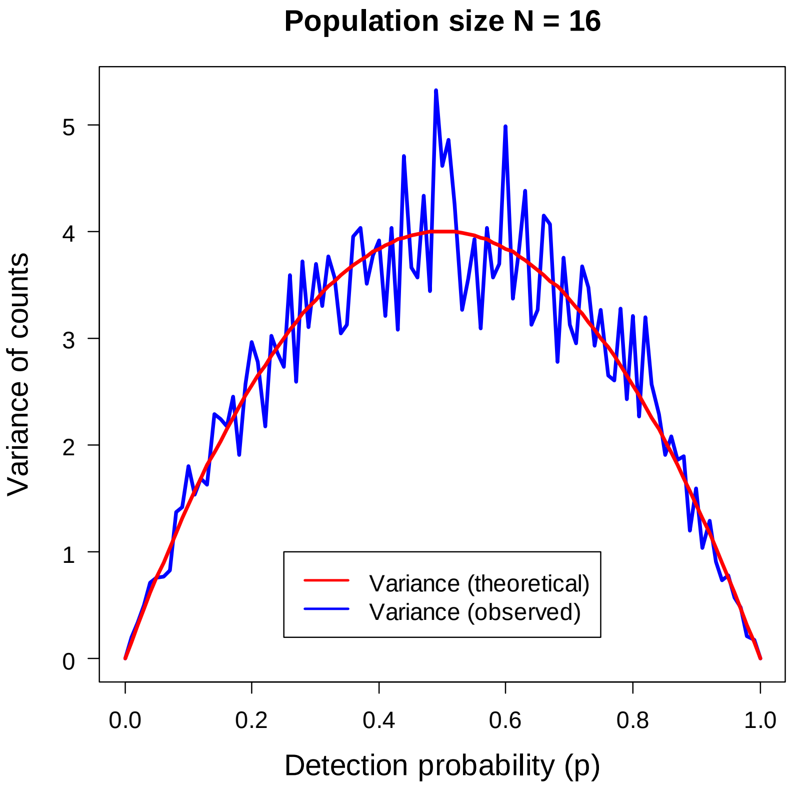
```
p <- 0.4
N <- 1:100
counts <- array(NA, dim = c(100, 100))
for (i in 1:100){
   counts[i,] <- rbinom(n = 100, size = N[i], prob = p)
   }
obs.variance <- apply(counts, 1, var)
theo.variance <- N*p*(1-p)
plot(N, obs.variance, col = "blue", type = "l", xlab = "Population size
(N)", ylab = "Variance of counts", las = 1, lwd = 3, cex.main = 1.5,
cex.lab = 1.5, cex.axis = 1.2, main = "Detection probability p = 0.4")
lines(N, theo.variance, col = "red", type = "l", lwd = 3)
legend(5, 25, c('Variance (theoretical)', 'Variance (observed)'),
col=c("red", "blue"), lty = c(1,1), lwd = 2, cex = 1.2)
```



This shows the relationship between population size *N* and the variance of the counts in those populations when detection probability is *p* = 0.4. We see that the observed variability in the binomial counts (in blue) closely follows what we expect it to be based on statistical theory (red line; this is *Np*(1-*p*)). Hence, everything else equal, nothing can save us from getting more variable counts in larger populations. And, small populations always produce more 'repeatable' counts than larger populations.

Next, we inspect the relationship between detection probability and the variance of counts. Here we assume that the population size is 16 individuals and that detection probability varies from 0 to 1.

```
p <- seq(0,1,0.01)
N <- 16
counts <- array(NA, dim = c(101, 100))
for (i in 1:101){
   counts[i,] <- rbinom(n = 100, size = N, prob = p[i])
   }
obs.variance <- apply(counts, 1, var)
theo.variance <- N*p*(1-p)
plot(p, obs.variance, col = "blue", type = "l", xlab = "Detection
probability (p)", ylab = "Variance of counts", las = 1, lwd = 3, cex.main =
1.5, cex.lab = 1.5, cex.axis = 1.2, main = "Population size N = 16")
lines(p, theo.variance, col = "red", type = "l", lwd = 3)
legend(0.25, 1, c('Variance (theoretical)', 'Variance (observed)'),
col=c("red", "blue"), lty = c(1,1), lwd = 2, cex = 1.2)
```



This shows the relationship between detection probability *p* and the variance of the counts in populations with *N* = 16. The actual variability in the binomial counts (in blue) closely follows what we expect it to be from statistical theory (red line; this is again *Np*(1-*p*)). Again, everything else equal, there is nothing that can save us from getting more variable counts when detection probability is close to 0.5. Hence, we can make the counts more 'repeatable' (i.e., reduce their variance) by either pushing up detection probability towards 1 or else by doing an extremely bad job at counting with a resulting detection probability close to (or equal to) zero. Hence, the variability of counts alone does not by itself say anything about how good the counts (or the people doing the counting) are.

Chapter 2

There are no exercises in this chapter.

Chapter 3

Exercise 1

Task: Adapt the first data generation function in this chapter to generate the data using coefficients that refer to the values of standardized covariate values and repeat the analysis in R and in WinBUGS.

Solution: We choose the coefficients of the cubic polynomial such that they result in a data set that is comparable to the one in the book.

```
data.fn <- function(n = 40, alpha = 4.32671, beta1 = 1.22677, beta2 =
     0.05552, beta3 = -0.23758){
     # n: Number of years
     # alpha, beta1, beta2, beta3: coefficients of a
     #    cubic polynomial of count on year (standardised to have mean 0
     #    and sd 1)

     # Generate values of time covariate and scale it
     year <- 1:n
     year <- as.numeric(scale(year))

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

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

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

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

We obtain one data set and analyse it using the ML routine for GLMs in R.

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

And now the Bayesian analysis in WinBUGS. Since the covariate year is now standardized, we needn't really make any changes to the existing code at all.We first define the place where the WinBUGS executable is sitting on our computer.

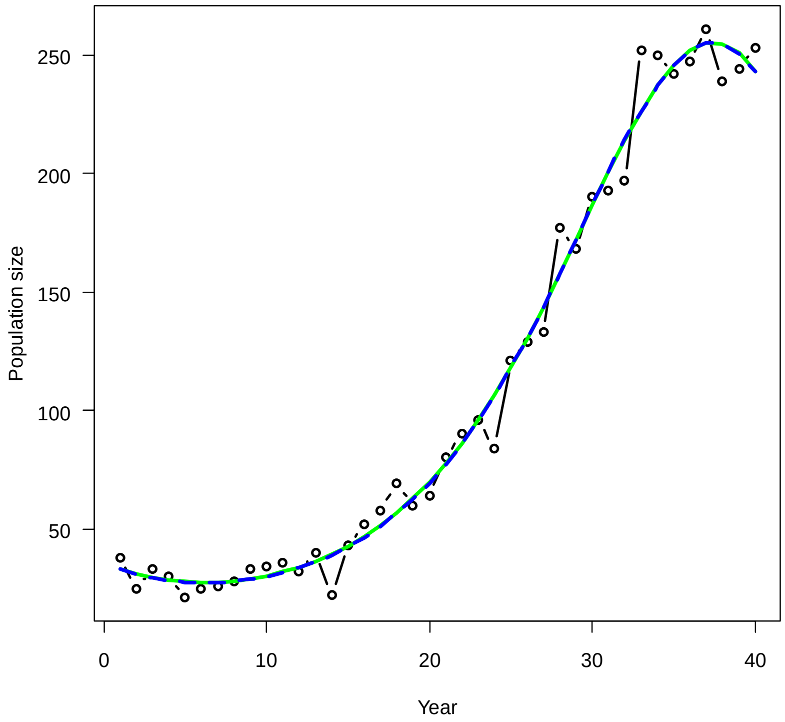
```
bugs.dir <- "c:/Program files/WinBUGS14/"

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

```
# Priors
alpha ~ dunif(-20, 20)
beta1 ~ dunif(-10, 10)
beta2 ~ dunif(-10, 10)
beta3 ~ dunif(-10, 10)

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

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

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

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

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

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

# Summarize posteriors
print(out$summary[1:4,], dig = 3)
         mean     sd    2.5%    25%     50%     75%  97.5% Rhat n.eff
alpha  4.2929 0.0305  4.2315  4.272  4.2930  4.3140  4.352    1   790
beta1  1.2430 0.0443  1.1540  1.214  1.2420  1.2700  1.336    1   610
beta2  0.0737 0.0238  0.0278  0.058  0.0734  0.0891  0.121    1  1000
beta3 -0.2326 0.0228 -0.2813 -0.247 -0.2314 -0.2183 -0.188    1   640

print(fm$coef, dig = 4)
(Intercept)        year   I(year^2)   I(year^3)
    4.29611     1.23851     0.07303    -0.23081
```

We see that the ML and Bayesian parameter estimates match quite well, as we expect when using vague prior distributions in the latter. We can also draw a plot that shows the observed data (in black) and the expected number of pairs under both analyses (ML in green, Bayesian posterior means in blue).

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

```
WinBUGS.predictions <- out$mean$lambda
lines(1:40, WinBUGS.predictions, type = "l", lwd = 3, col = "blue", lty =
2)
```



Exercise 2

Task: Take the following toy data set and fit a logistic regression of the number of successes *r* among *n* trials as a function of covariate *X*. Also write out the GLM for this data set.

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

Solution: We start by plotting the data. Actually, we plot directly our estimate of the binomial parameter *p*, which is *r*/*n*. Note that this is NOT the observed reponse that is modeled in the binomial GLM, rather, the observed response is *r*. This may be confusing at first.

```
plot(X, r/n, type = "p", xlab = "Covariate X", ylab = "Ratio r/n", las = 1,
lwd = 3, cex.main = 1.5, cex.lab = 1.5, cex.axis = 1.2, main = "")
```

So there appears to be a declining relationship between the success parameter *p* and the covariate *X*. To formally examine the relationship, we fit the following GLM to the observed counts r:

1. The assumed statistical distribution to capture the random variability in the response is the binomial, which also requires specification of the binomial total, or index, *N* (which is n in our case). Since the binomial is the sum of *N* Bernoullis (single coin flips), this is the number of coin flips that nature made to arrive at a count of *r*. Hence, we can write $r_i \sim Binomial(N_i, p_i)$

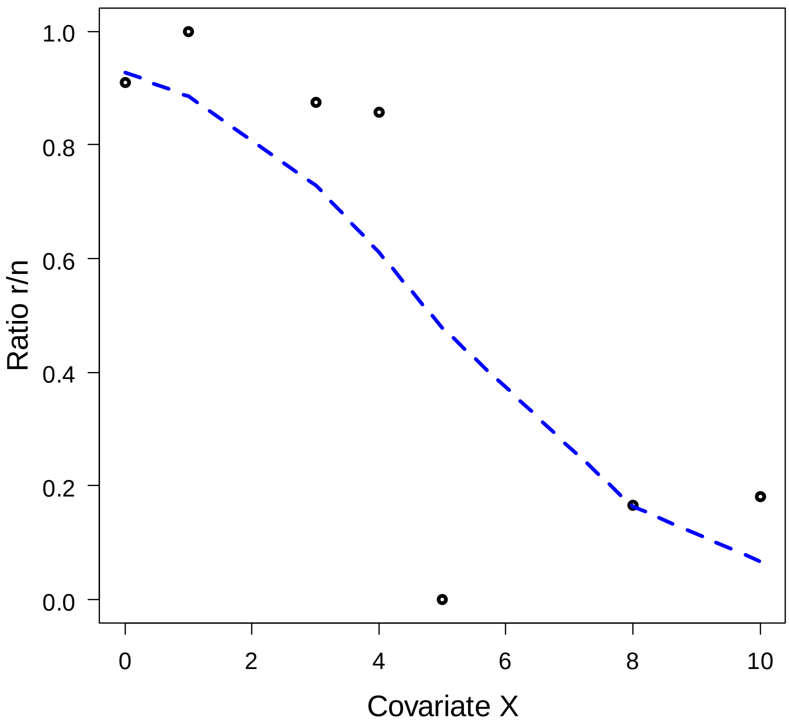
2. The link function is the transformation that we apply to the expected response (which is *p* for the Bernoulli, or *N* times *p* for the Binomial with *N*>1) to be able to model it as a linear function of covariates. We choose the customary logit link, i.e., we can write $\eta_i = g(\mu_i) = \text{logit}(p_i)$.
3. The linear predictor is the way how we imagine that the expected response, on the link scale, is affected by the covariates of our choice. Since we want to fit a simple linear regression model on the logit scale, we can write $\eta_i = \text{logit}(p_i) = \alpha + \beta * X_i$.

To fit the binomial GLM, we can take code from section 3.5 and slightly adapt it for the variable names (also, we don't fit a quadratic term of the covariate).

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

# Priors
alpha ~ dnorm(0, 0.001)
beta ~ dnorm(0, 0.001)

# Likelihood
for (i in 1:sample.size){
   r[i] ~ dbin(p[i], n[i])
   logit(p[i]) <- alpha + beta * X[i]
   }
}
",fill = TRUE)
sink()

# Bundle data
win.data <- list(r = r, n = n, sample.size = length(X), X = X)

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

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

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

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

We can plot the predictions into the plot made earlier.

```
# Plot predictions (don't get confused with order !)
predictions <- out$mean$p
lines(sort(X), predictions[order(X)], lwd = 3, col = "blue", lty = 2)
```



Exercise 3

Task: The Bernoulli distribution is a special case of the binomial with trial size equal to 1. It has only one parameter, the success probability *p*. The Bernoulli distribution is a conventional model for species distributions, where observed detection/nondetection data are related to explanatory (e.g., habitat) variables in a linear or other fashion with a logit link. Write an R function to assemble "presence/absence" data collected at 200 sites, where the success probability (i.e., occurrence probability) is related to habitat variable *X* (ranging from -1 to 1) on the logit-linear scale with intercept -2 and slope 5. Then write a WinBUGS program to 'break down' the simulated data (i.e., analyze them) and thus recover these parameter values.

Solution: Here is the R function to generate and plot the data.

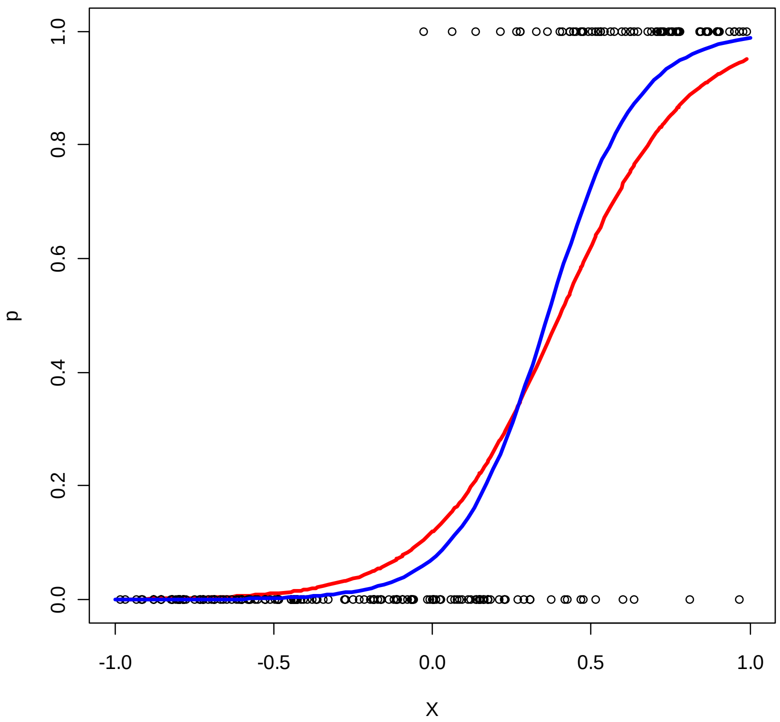
```
data.generation <- function(){
n <- 200                # Number of sites
X <- sort(runif(n, -1, 1))   # Random uniform on range -1 to 1
a <- -2                 # Intercept ...
b <- 5                  # ... and slope of logit-linear relationship
p <- plogis(a + b * X)  # Success probability
y <- rbinom(n = n, size = 1, prob = p) # Observed binary data

plot(X, p, xlim = c(-1, 1), ylim = c(0,1), type = "l", lwd = 3, col =
"red")
points(X, y)

return(list(X = X, y = y))
}
```

We generate one data set and attach it.

```
data <- data.generation()
attach(data)
```

And here is code to fit a logistic regression model to a data set generated with this function in WinBUGS.

```
# Define the model in the BUGS language and write a text file
sink("model.txt")
cat("
model {

# Priors
alpha ~ dunif(-20, 20)
beta ~ dunif(-20, 20)

# Likelihood: Note key components of a GLM in one line each
for (i in 1:nobs){
   y[i] ~ dbern(p[i])              # 1. Distribution for random part
   logit(p[i]) <- lp[i]            # 2. Link function
   lp[i] <- alpha + beta * X[i]    # 3. Linear predictor
   }

# Form predictions using pred.x
for (i in 1:nobs.pred){
   logit(pred.p[i]) <- alpha + beta * pred.x[i]
   }
}  # end model
",fill=TRUE)
sink()

# Bundle data
pred.x = seq(-1, 1, length.out = 100)
win.data <- list(y = y, X = X, nobs = length(y), pred.x = pred.x, nobs.pred
= length(pred.x))

# Initial values (not required for all)
inits <- function() list(alpha = runif(1, -2, 2))

# Define parameters to be monitored
params <- c("alpha", "beta", "pred.p")

# MCMC settings
ni <- 600   ;   nt <- 2   ;   nb <- 100   ;   nc <- 3

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

# Summarize posterior distributions
print(out, dig = 3)

# Plot model predictions into the same plot
points(pred.x, out$mean$pred.p, col = "blue", lwd = 3, type = "l")
```



Remember that red is the truth and blue our estimate.

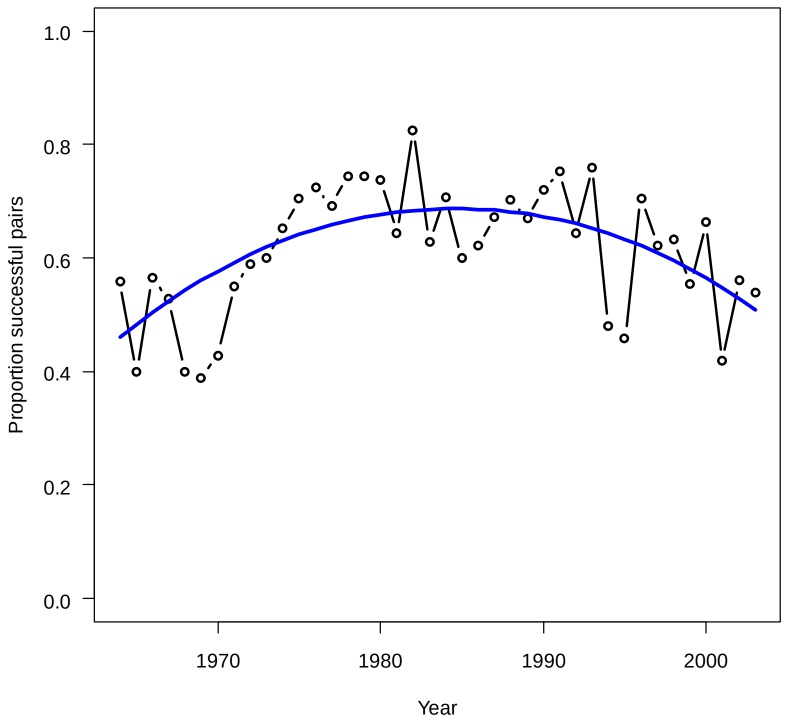
Exercise 4

Task: In section 3.5.2 we used a binomial GLM to describe the proportion of successful peregrine pairs per year in the French Jura mountains. To see the connections between three important types of GLMs, first use a Poisson GLM to model the number of successful pairs (thus disregarding the fact that the binomial total varies by year) and second, use a normal GLM to do the same. In the same graph compare the predicted numbers of successful pairs for every year under all three models (binomial, Poisson and normal GLM). Do this both in R and in WinBUGS.

Solution: We will start by fitting the Binomial model again and then fit the Poisson and the normal model in R and in WinBUGS. We assume that you have loaded the peregrine data already and assigned them to an object 'peregrine', which you've attached. So first, the binomial model again.

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

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

# Likelihood
for (i in 1:nyears){
   C[i] ~ dbin(p[i], N[i])          # 1. Distribution for random part
```

```
   logit(p[i]) <- alpha + beta1 * year[i] + beta2 * pow(year[i],2) # link
function and linear predictor
   }
}
",fill = TRUE)
sink()

# Bundle data
win.data <- list(C = R.Pairs, N = Pairs, nyears = length(Year), year =
(Year-1984)/20)

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

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

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

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

# Summarize posteriors and plot estimates of proportion of successful pairs
print(out1, dig = 3)
            mean    sd    2.5%     25%     50%     75%   97.5%  Rhat n.eff
alpha      0.786 0.056   0.676   0.748   0.786   0.823   0.896 1.001  2500
beta1      0.054 0.074  -0.090   0.003   0.055   0.104   0.195 1.001  3000
beta2     -0.890 0.122  -1.128  -0.974  -0.888  -0.808  -0.648 1.001  3000
p[1]       0.461 0.037   0.390   0.436   0.460   0.485   0.533 1.001  3000
p[2]       0.483 0.034   0.420   0.460   0.482   0.506   0.550 1.001  3000
[ ... ]

plot(Year, R.Pairs/Pairs, type = "b", lwd = 2, col = "black", main = "",
las = 1, ylab = "Proportion successful pairs", xlab = "Year", ylim =
c(0,1))
lines(Year, out1$mean$p, type = "l", lwd = 3, col = "blue")
```



Here's the logistic regression fit using ML in R.

```
C <- R.Pairs
N <- Pairs
year <- (Year-1984)/20
fm1 <- glm(cbind(C, N-C) ~ year + I(year^2), family = binomial)
summary(fm1)
```

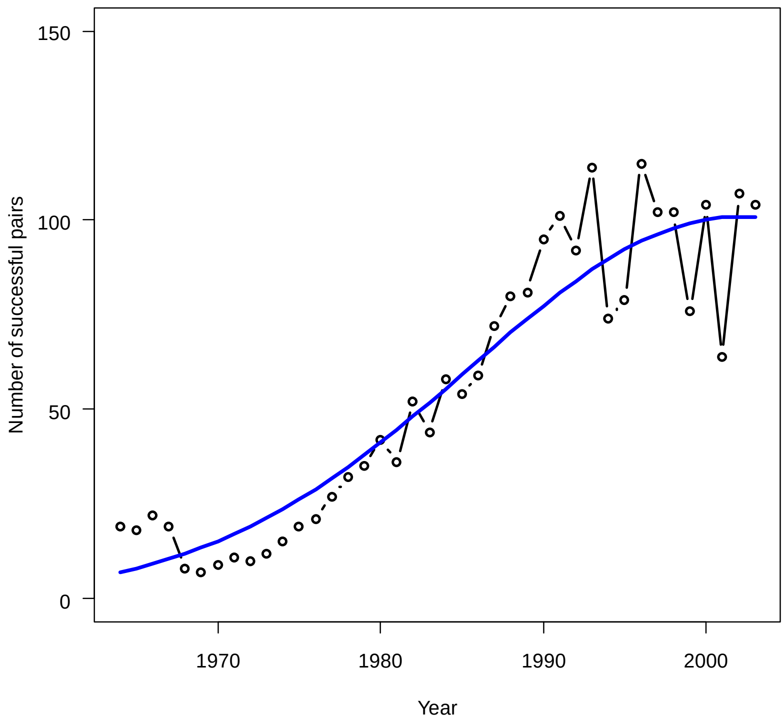
We compare the MLEs and the Bayesian posterior means.

```
print(out1$summary[1:3,], dig = 3)
          mean     sd    2.5%      25%    50%    75%  97.5% Rhat n.eff
alpha   0.7855 0.0563  0.6760  0.74767  0.786  0.823  0.896    1  2500
beta1   0.0541 0.0737 -0.0896  0.00332  0.055  0.104  0.195    1  3000
beta2  -0.8901 0.1221 -1.1280 -0.97380 -0.888 -0.808 -0.648    1  3000

print(summary(fm1)$coef[,1:2])
               Estimate Std. Error
(Intercept)  0.78578050 0.05568340
year         0.05639723 0.07433135
I(year^2)   -0.89203218 0.12247088
```

Second, we fit a Poisson GLM to the counts of successful pairs, thereby ignoring the variable number of pairs surveyed during the 40 years. The (single) Poisson parameter is frequently called lambda, but to make the correspondence between the different parametets in the binomial, the Poisson and the normal crystal clear, we keep p as the name for the mean parameter throughout.

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

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

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

# Bundle data
win.data <- list(C = R.Pairs, nyears = length(Year), year = (Year-1984)/20)

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

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

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

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

# Summarize posteriors and plot estimates of counts of successful pairs
print(out2, dig = 3)
            mean    sd   2.5%    25%    50%    75%  97.5%  Rhat n.eff
alpha      4.015 0.033  3.951  3.993  4.015  4.037  4.079 1.001  3000
beta1      1.334 0.054  1.231  1.296  1.333  1.370  1.440 1.001  3000
beta2     -0.745 0.088 -0.917 -0.804 -0.744 -0.684 -0.573 1.001  3000
p[1]       6.981 0.811  5.503  6.413  6.926  7.515  8.634 1.001  3000
p[2]       8.017 0.856  6.460  7.419  7.969  8.589  9.762 1.001  3000
[ ... ]

plot(Year, R.Pairs, type = "b", lwd = 2, col = "black", main = "", las = 1,
ylab = "Number of successful pairs", xlab = "Year", ylim = c(0,150))
lines(Year, out2$mean$p, type = "l", lwd = 3, col = "blue")
```



Here's the Poisson regression fit using ML in R.

```
C <- R.Pairs
year <- (Year-1984)/20
fm2 <- glm(C ~ year + I(year^2), family = poisson)
summary(fm2)
```

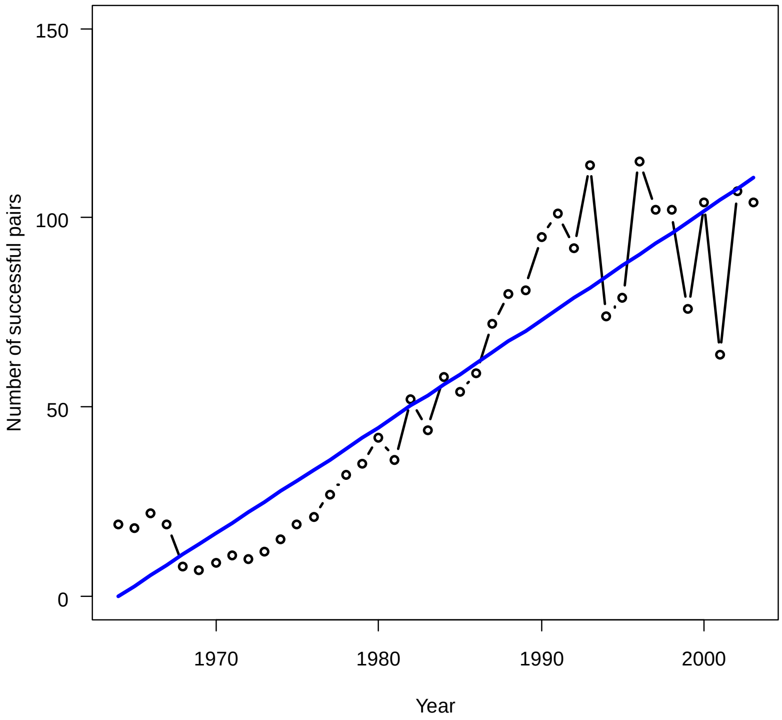
We compare the MLEs and the Bayesian posterior means.

```
print(out2$summary[1:3,], dig = 3)
        mean     sd   2.5%    25%    50%    75%  97.5% Rhat n.eff
alpha   4.015 0.0327  3.951  3.993  4.015  4.037  4.079    1  3000
beta1   1.334 0.0537  1.231  1.296  1.333  1.370  1.440    1  3000
beta2  -0.745 0.0881 -0.917 -0.804 -0.744 -0.684 -0.573    1  3000

print(summary(fm2)$coef[,1:2])
              Estimate Std. Error
(Intercept)  4.0150415 0.03219114
year         1.3320664 0.05425388
I(year^2)   -0.7400287 0.08883842
```

Third, we fit a Normal GLM to the counts of successful pairs, again ignoring the variable number of pairs surveyed during the 40 years. We call the mean parameter of the normal distribution p as before, and the variance sigma2. We also need to modify the priors a little from before to avoid to constrain the posterior distributions.

```
# Specify Normal GLM in BUGS language
sink("GLM_Normal.txt")
cat("
model {

# Priors
```

```
alpha ~ dnorm(0, 0.00001)
beta1 ~ dnorm(0, 0.00001)
beta2 ~ dnorm(0, 0.00001)
tau <- pow(sigma, -2)
sigma ~ dunif(0, 100)

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

# Bundle data
win.data <- list(C = R.Pairs, nyears = length(Year), year = (Year-1984)/20)

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

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

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

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

# Summarize posteriors and plot estimates of counts of successful pairs
print(out3, dig = 3)
            mean    sd    2.5%     25%     50%     75%   97.5%  Rhat n.eff
alpha     55.942 3.609  48.850  53.620  55.895  58.330  62.981 1.002  1500
beta1     56.686 4.198  48.550  53.910  56.705  59.510  65.000 1.001  2600
beta2      0.782 8.079 -14.782  -4.772   0.669   6.188  16.582 1.001  3000
p[1]       0.039 6.859 -13.441  -4.437  -0.010   4.646  13.671 1.001  2800
p[2]       2.797 6.196  -9.427  -1.272   2.824   6.945  15.172 1.001  3000
[ ... ]

plot(Year, R.Pairs, type = "b", lwd = 2, col = "black", main = "", las = 1,
ylab = "Number of successful pairs", xlab = "Year", ylim = c(0,150))
lines(Year, out3$mean$p, type = "l", lwd = 3, col = "blue")
```



Then, we also use `glm()` to fit the Normal regression using ML in R (we could also use `lm()`).

```
C <- R.Pairs
year <- (Year-1984)/20
fm3 <- glm(C ~ year + I(year^2), family = gaussian)
summary(fm3)
```

We compare the MLEs and the Bayesian posterior means.

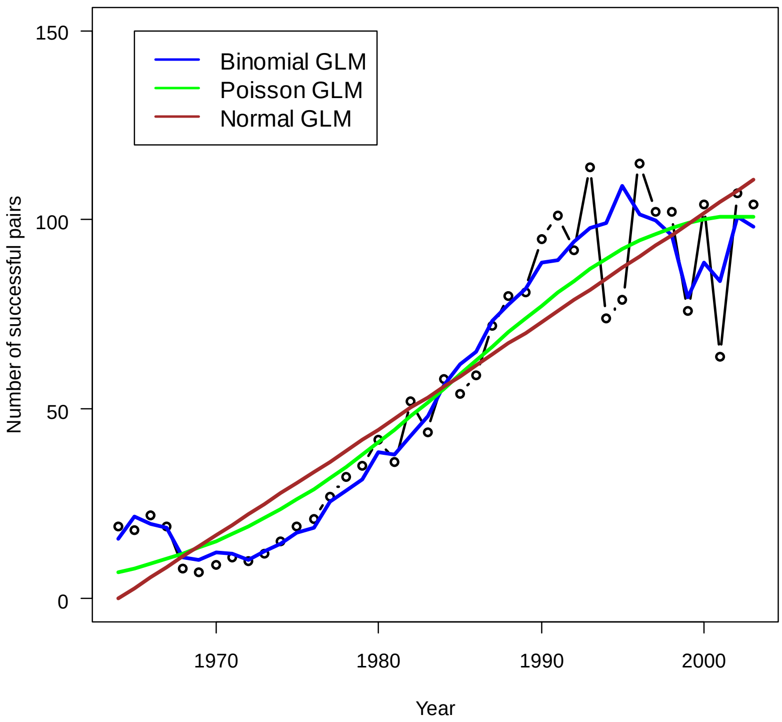
```
print(out3$summary[1:3,], dig = 3)
        mean   sd  2.5%   25%    50%   75% 97.5% Rhat n.eff
alpha 55.942 3.61  48.8 53.62 55.895 58.33  63.0    1  1500
beta1 56.686 4.20  48.5 53.91 56.705 59.51  65.0    1  2600
beta2  0.782 8.08 -14.8 -4.77  0.669  6.19  16.6    1  3000

print(summary(fm3)$coef[,1:2])
              Estimate Std. Error
(Intercept) 55.8714353   3.525820
year        56.7857566   4.092058
I(year^2)    0.9684154   7.897137
```

Noting how imprecise the estimate of the quadratic effect of year is, we acknowledge the usual similarity between MLEs and Bayesian posterior means from an analysis with vague priors.

Finally, we plot the predictions of the number of successful pairs under the three different models into a single diagram.

```
plot(Year, R.Pairs, type = "b", lwd = 2, col = "black", main = "", las = 1,
ylab = "Number of successful pairs", xlab = "Year", ylim = c(0,150))
```

```
lines(Year, Pairs*out1$mean$p, type = "l", lwd = 3, col = "blue")
lines(Year, out2$mean$p, type = "l", lwd = 3, col = "green")
lines(Year, out3$mean$p, type = "l", lwd = 3, col = "brown")
legend(1965, 150, c('Binomial GLM', 'Poisson GLM', 'Normal GLM'), col =
c("blue", "green", "brown"), lty = 1, lwd = 2, cex = 1.2)
```



Though it it clear from this plot that the binomial model provides the best fit to the data, we don't discuss any further the differences among the three GLMs. What we do hope is that this exercise has provided a nice demonstration of how easy one can 'jump' from one to the other when using the BUGS language and how transparent the differences between the models are. In R, using `glm()`, it is also trivial to go from one to the other, but the actual meaning of these variants of a GLM is more elusive.

Chapter 4

Exercise 1

Task: Overdispersion: Generate a data set using the function in section 4.2.1 and use WinBUGS to compare the regression estimates under the Poisson GLM and those under the Poisson GLMM. The Bayesian analysis yields better estimates of the uncertainty in the estimates of a random-effects model and lets you see more clearly how the regression estimates have an increased posterior standard deviation when estimated under the model with random year effects.

Solution: We first use the data-generating function in section 4.2.1 to obtain one data set.

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

Then, we fit the two Poisson models. Here is code for fitting the simple Poisson GLM, adapted from the one in section 3.3.1.

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

# Priors
alpha ~ dunif(-20, 20)
beta1 ~ dunif(-10, 10)
beta2 ~ dunif(-10, 10)
beta3 ~ dunif(-10, 10)

# Likelihood
for (i in 1:n){
   C[i] ~ dpois(lambda[i])
   log(lambda[i]) <- log.lambda[i]
   log.lambda[i] <- alpha + beta1 * year[i] + beta2 * pow(year[i],2) +
beta3 * pow(year[i],3)
   }
}
",fill = TRUE)
sink()

# Bundle data (note standardization of covariate)
win.data <- list(C = data$C, n = length(data$C), year = (data$year-20)/20)

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

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

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

# Call WinBUGS from R
```

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

# Summarize posteriors
print(out1, dig = 3)
```

And here is the code for fitting the Poisson GLMM, which allows for random annual deviations of the Poisson mean around the cubic regression line and thereby accounts for such a form of overdispersion.

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

# Priors
alpha ~ dunif(-20, 20)
beta1 ~ dunif(-10, 10)
beta2 ~ dunif(-10, 10)
beta3 ~ dunif(-10, 10)
tau <- 1 / (sd*sd)
sd ~ dunif(0, 5)

# Likelihood
for (i in 1:n){
   C[i] ~ dpois(lambda[i])
   log(lambda[i]) <- log.lambda[i]
   log.lambda[i] <- alpha + beta1 * year[i] + beta2 * pow(year[i],2) +
beta3 * pow(year[i],3) + eps[i]
   eps[i] ~ dnorm(0, tau)
   }
}
",fill = TRUE)
sink()

# Bundle data (note standardization of covariate)
win.data <- list(C = data$C, n = length(data$C), year = (data$year-20)/20)

# Initial values
inits <- function() list(alpha = runif(1, -2, 2), beta1 = runif(1, -3, 3),
sd = runif(1, 0,1))

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

# MCMC settings
ni <- 30000
nt <- 10
nb <- 20000
nc <- 3

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

# Summarize posteriors
print(out2, dig = 2)
```

We compare the estimates of the regression coefficients under the simple GLM and the more complex GLMM. Note that we can't directly compare them to the values used in the data-generating function, because now we scaled the covariate, while in the data-generation we didn't. Don't forget that the exact numbers will depend on the actual data set that you drew from the stochastic system described by the data-generating function.

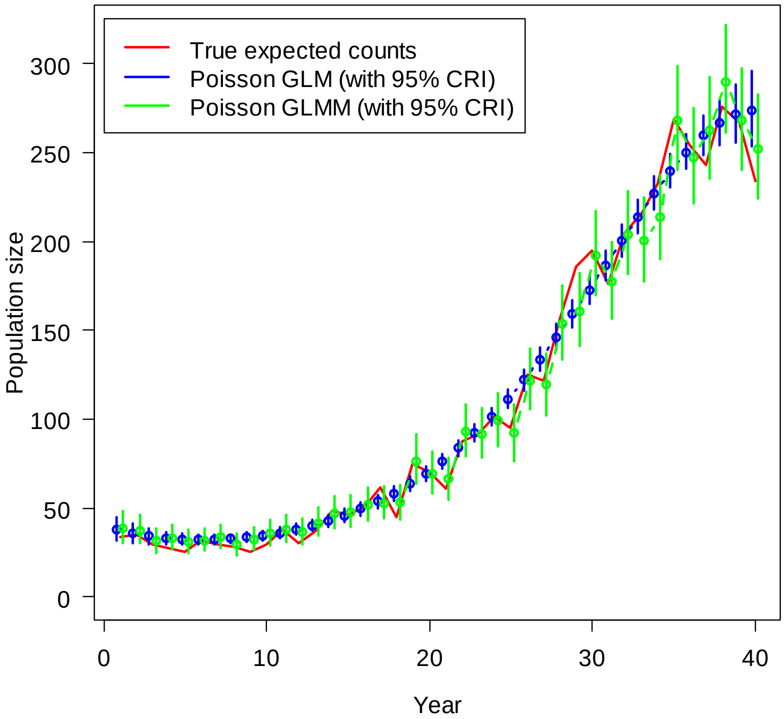
```
GLM.estimate <- out1$summary[1:4,1:2]
GLMM.estimate <- out2$summary[1:4,1:2]
cbind(GLM.estimate, GLMM.estimate)
            mean         sd       mean         sd
alpha  4.2414047 0.03098390  4.2381917 0.04006138
beta1  1.8264920 0.07799633  1.8091873 0.11744985
beta2  0.4176249 0.07307508  0.4100419 0.09280864
beta3 -0.8741873 0.11851932 -0.8427971 0.17849864
```

We see that while the point estimates are very similar, the uncertainty estimate (posterior standard deviation) for this data set is about 28% larger on average for the GLMM, which accounts for the added uncertainty coming from the random year effects.

```
mean(1 - (GLM.estimate[,2] / GLMM.estimate[,2]))
```

We can also plot the expected counts in each year with their uncertainty intervals (95% CRI). The latter are again wider under the GLMM.

```
plot(data$year, data$expected.count, type = "l", col = "red", lwd = 2, main
= "", las = 1, ylab = "Population size", xlab = "Year", ylim = c(0, 320),
cex.axis = 1.2, cex.lab = 1.2, las = 1)
lines(1:40-0.2, out1$mean$lambda, type = "b", col = "blue", lwd = 2, lty =
2)
segments(1:40-0.2, out1$summary[5:44,3], 1:40-0.2, out1$summary[5:44,7],
col = "blue", lwd = 2)
lines(1:40+0.2, out2$mean$lambda, type = "b", col = "green", lwd = 2, lty =
2)
segments(1:40+0.2, out2$summary[5:44,3], 1:40+0.2, out2$summary[5:44,7],
col = "green", lwd = 2)
legend(0, 325, c('True expected counts', 'Poisson GLM (with 95% CRI)',
'Poisson GLMM (with 95% CRI)'), col=c("red", "blue", "green"), lty = 1, lwd
= 2, cex = 1.2)
```



We see that the 95% CRI from the simple GLM overstate the precision in the estimates of the expected counts. In contrast, the CRI from the GLMM have much better coverage (relative to the red line).

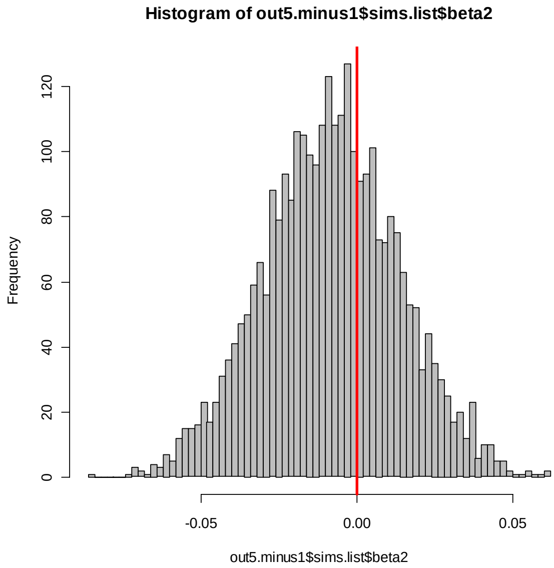
Exercise 2

Task: First-year observer effect: We have seen that in the tit data any first-year observer effect is confounded with the effect of the first year. Repeat the last analysis for a restricted data set without year 1.

Solution: We will repeat the fitting of model GLMM3 with the subsetted data. You will have to execute part of the R code at the beginning of section 4.3.2 to read in the data set and pre-process it for the analysis. There is nothing that needs to be changed in the model code, so we can directly use the model file GLMM3.txt again. All we have to do is to adapt the data statement by cutting off the first year of data (plus the number of inits given for `eps`).

```
# Bundle new data without first year
win.data <- list(C = t(C[,-1]), nsite = nrow(C), nyear = ncol(C)-1, first =
t(first[,-1]))

# Specify model in BUGS language
sink("GLMM3.txt")
cat("
model {

# Priors
mu ~ dnorm(0, 0.01)                    # Overall mean
beta2 ~ dnorm(0, 0.01)                 # First-year observer effect
```

```
for (j in 1:nsite){
   alpha[j] ~ dnorm(0, tau.alpha)   # Random site effects
   }
tau.alpha <- 1/ (sd.alpha * sd.alpha)
sd.alpha ~ dunif(0, 5)

for (i in 1:nyear){
   eps[i] ~ dnorm(0, tau.eps)      # Random year effects
   }
tau.eps <- 1/ (sd.eps * sd.eps)
sd.eps ~ dunif(0, 5)

# Likelihood
for (i in 1:nyear){
   for (j in 1:nsite){
      C[i,j] ~ dpois(lambda[i,j])
      lambda[i,j] <- exp(log.lambda[i,j])
      log.lambda[i,j] <- mu + beta2 * first[i,j] + alpha[j] + eps[i]
      }  #j
   }  #i
}
",fill = TRUE)
sink()

# Initial values
inits <- function() list(mu = runif(1, 0, 4), beta2 = runif(1, -1, 1),
alpha = runif(235, -2, 2), eps = runif(8, -1, 1))

# Parameters monitored
params <- c("mu", "beta2", "alpha", "eps", "sd.alpha", "sd.eps")

# MCMC settings
ni <- 6000
nt <- 5
nb <- 1000
nc <- 3

# Call WinBUGS from R (BRT 3 min)
out5.minus1 <- bugs(win.data, inits, params, "GLMM3.txt", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory =
bugs.dir, working.directory = getwd())

# Summarize posterior of first-year observer effect in table and figure
print(out5.minus1$summary[1:2,], dig = 3)
hist(out5.minus1$sims.list$beta2, breaks = 100, col = "grey")
abline(v = 0, col = "red", lwd = 3)
```



So even when discarding the data from the first year, which were confounding our estimate of the first-year observer effect in the analysis in the book, we still can't detect any such effect.

Exercise 3

Task: Reparameterizations: In GLMM 2, put the grand mean of the double random effects model, mu, into the hyperdistribution of one of the random effects. That is, fit the model like this:

```
for (j in 1:nsite){
   alpha[j] ~ dnorm(mu, tau.alpha)
   }
```

You will see that convergence is worse. This is an example of where WinBUGS is very sensitive to how a model is parameterized.

Solution: We again assume that you have read in the tit data set and pre-processed it so that you can run the R and WinBUGS commands below. We run the parameterization of the model as it is shown in the BPA book first and call it A and then using the other parameterization, which we call B.

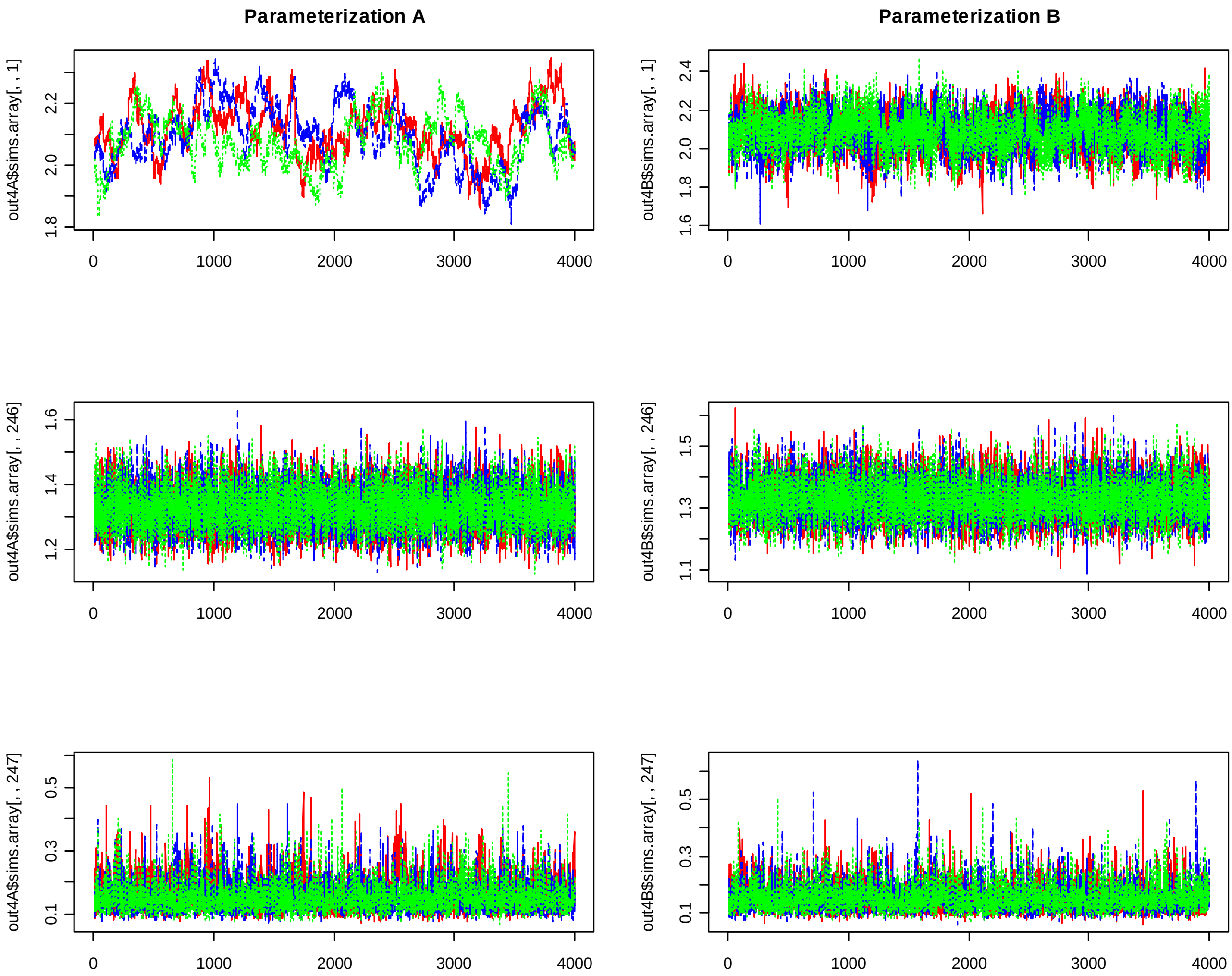
```
# Specify model in BUGS language: this is the model as in the book
sink("GLMM2A.txt")
cat("
model {

# Priors
mu ~ dnorm(0, 0.01)                  # Grand mean

for (j in 1:nsite){
   alpha[j] ~ dnorm(0, tau.alpha)    # Random site effects
   }
tau.alpha <- 1/ (sd.alpha * sd.alpha)
sd.alpha ~ dunif(0, 5)
```

```
for (i in 1:nyear){
   eps[i] ~ dnorm(0, tau.eps)         # Random year effects
   }
tau.eps <- 1/ (sd.eps * sd.eps)
sd.eps ~ dunif(0, 3)

# Likelihood
for (i in 1:nyear){
   for (j in 1:nsite){
       C[i,j] ~ dpois(lambda[i,j])
      lambda[i,j] <- exp(log.lambda[i,j])
      log.lambda[i,j] <- mu + alpha[j] + eps[i]
      }  #j
   }  #i
}
",fill = TRUE)
sink()

# Bundle data
win.data <- list(C = t(C), nsite = nrow(C), nyear = ncol(C))

# Initial values (not required for all)
inits <- function() list(mu = runif(1, 0, 4), alpha = runif(235, -2, 2),
eps = runif(9, -1, 1))

# Parameters monitored
params <- c("mu", "sd.alpha", "sd.eps", "alpha", "eps")

# MCMC settings
ni <- 10000
nt <- 2
nb <- 2000
nc <- 3

# Call WinBUGS from R (BRT 3 min)
out4A <- bugs(win.data, inits, params, "GLMM2A.txt", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory =
bugs.dir, working.directory = getwd())

# Summarize posteriors of hyperparameters
print(out4A$summary[c(1, 246:247),], dig = 4)
            mean      sd    2.5%    25%    50%    75%  97.5%  Rhat n.eff
mu        2.0970 0.09293 1.91597 2.0340 2.0970 2.1610 2.2730 1.039    61
sd.alpha 1.3287 0.06615 1.20700 1.2830 1.3260 1.3710 1.4670 1.001  3400
sd.eps    0.1543 0.04928 0.08992 0.1204 0.1445 0.1766 0.2787 1.006   420
```

This is the parameterization from the book. And now the other parameterization.

```
# Specify model in BUGS language
sink("GLMM2B.txt")
cat("
model {

# Priors
for (j in 1:nsite){
   alpha[j] ~ dnorm(mu, tau.alpha)    # Random site effects with grand mean
   }
mu ~ dnorm(0, 0.01)
tau.alpha <- 1/ (sd.alpha * sd.alpha)
sd.alpha ~ dunif(0, 5)

for (i in 1:nyear){
```

```
   eps[i] ~ dnorm(0, tau.eps)        # Random year effects
   }
tau.eps <- 1/ (sd.eps * sd.eps)
sd.eps ~ dunif(0, 3)

# Likelihood
for (i in 1:nyear){
   for (j in 1:nsite){
      C[i,j] ~ dpois(lambda[i,j])
      lambda[i,j] <- exp(log.lambda[i,j])
      log.lambda[i,j] <- alpha[j] + eps[i]
      }  #j
   }  #i
}
",fill = TRUE)
sink()

# Bundle data
win.data <- list(C = t(C), nsite = nrow(C), nyear = ncol(C))

# Initial values (not required for all)
inits <- function() list(mu = runif(1, 0, 4), alpha = runif(235, -2, 2),
eps = runif(9, -1, 1))

# Parameters monitored
params <- c("mu", "sd.alpha", "sd.eps", "alpha", "eps")

# MCMC settings
ni <- 10000
nt <- 2
nb <- 2000
nc <- 3

# Call WinBUGS from R (BRT 3 min)
out4B <- bugs(win.data, inits, params, "GLMM2B.txt", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory =
bugs.dir, working.directory = getwd())

# Summarize posteriors of hyperparameters
print(out4B$summary[c(1, 246:247),], dig = 4)
           mean      sd    2.5%    25%    50%   75%  97.5%  Rhat n.eff
mu       2.0871 0.10036 1.88697 2.0210 2.0890 2.155 2.2810 1.001 12000
sd.alpha 1.3294 0.06647 1.20600 1.2830 1.3270 1.372 1.4690 1.001 12000
sd.eps   0.1543 0.04792 0.09051 0.1207 0.1447 0.177 0.2751 1.002  1600

# Produce traceplots for hyperparameters from both analyses
par(mfrow = c(3,2))
matplot(out4A$sims.array[,,1], type = "l", col = c("red", "blue", "green"),
main = "Parameterization A")
matplot(out4B$sims.array[,,1], type = "l", col = c("red", "blue", "green"),
main = "Parameterization B")

matplot(out4A$sims.array[,,246], col = c("red", "blue", "green"), type =
"l")
matplot(out4B$sims.array[,,246], col = c("red", "blue", "green"), type =
"l")

matplot(out4A$sims.array[,,247], col = c("red", "blue", "green"), type =
"l")
matplot(out4B$sims.array[,,247], col = c("red", "blue", "green"), type =
"l")
```



So actually, it is the other way round from what is suggested in the task! Convergence and mixing is better for parameterization B. In this example, parameterization A also produced satisfactory results within reasonable time, but in other, more complex models, switching from one parameterization of a model to another may be decisive for getting an analysis to work.

Exercise 4

Task: Interpretation of random effects: Fit a series of models to the tit data with different random effects:

- a site random effect: random contributions from each site
- a year random effect: random contributions from each year
- a site plus a year random effect
- a site-by-year random effect: random contributions from each site-year combination

Compare parameter estimates, explain the difference in the interpretation of those models and try to make sense of the differences.

Solution: We will number these models for ease of presentation. This numbering will be different from the one in chapter 4 in the book. We will also adapt the notation of parameters to make the comparisons easier in this exercise.

Model 1: site random effects: random contributions from each site
Model 2: year random effects: random contributions from each year
Model 3: random site plus year random effects: independent random contributions of site and year
Model 4: site-by-year random effects: random contributions from each site-year combination

As before, we assume that you have the data prepared for analysis with WinBUGS.

First, we fit model 1, with a mean and random site effects only. This is the model on p. 102 in the BPA book (GLMM1).

```
# Specify model 1 in BUGS language
sink("model1.txt")
cat("
model {

# Priors
for (j in 1:nsite){
   alpha[j] ~ dnorm(mu, tau.site)   # Random site effects
   }
mu ~ dnorm(0, 0.01)
tau.site <- pow(sd.site, -2)
sd.site ~ dunif(0, 5)

# Likelihood
for (i in 1:nyear){
   for (j in 1:nsite){
      C[i,j] ~ dpois(lambda[i,j])
      lambda[i,j] <- exp(log.lambda[i,j])
      log.lambda[i,j] <- alpha[j]
      }  #j
   }  #i
}
",fill = TRUE)
sink()


# Bundle data
win.data <- list(C = t(C), nsite = nrow(C), nyear = ncol(C))

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

# Parameters monitored
params <- c("mu", "sd.site", "alpha")

# MCMC settings
ni <- 1200
nt <- 2
nb <- 200
nc <- 3

# Call WinBUGS from R (BRT 1 min)
out.model1 <- bugs(win.data, inits, params, "model1.txt", n.chains = nc,
```

```
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory =
bugs.dir, working.directory = getwd())

# Summarize posteriors for all quantities
print(out.model1, dig = 2)

# Summarize posteriors for hyperparams only
print(out.model1$summary[236:237,], dig = 4)
          mean      sd  2.5%   25%   50%   75% 97.5%  Rhat n.eff
mu       2.093 0.08715 1.921 2.037 2.090 2.149 2.264 1.004   450
sd.site 1.327 0.06546 1.207 1.282 1.326 1.369 1.465 1.000  1500
```

Second, we fit model 2, with a mean and random year effects only.

```
# Specify model 2 in BUGS language
sink("model2.txt")
cat("
model {

# Priors
for (i in 1:nyear){
   beta[i] ~ dnorm(mu, tau.year)   # Random year effects
   }
mu ~ dnorm(0, 0.01)
tau.year <- pow(sd.year, -2)
sd.year ~ dunif(0, 3)

# Likelihood
for (i in 1:nyear){
   for (j in 1:nsite){
      C[i,j] ~ dpois(lambda[i,j])
      lambda[i,j] <- exp(log.lambda[i,j])
      log.lambda[i,j] <- beta[i]
      }  #j
   }  #i
}
",fill = TRUE)
sink()


# Bundle data
win.data <- list(C = t(C), nsite = nrow(C), nyear = ncol(C))

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

# Parameters monitored
params <- c("mu", "sd.year", "beta")

# MCMC settings
ni <- 1200
nt <- 2
nb <- 200
nc <- 3

# Call WinBUGS from R (BRT 1 min)
out.model2 <- bugs(win.data, inits, params, "model2.txt", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory =
bugs.dir, working.directory = getwd())

# Summarize posteriors for all quantities
```

```
print(out.model2, dig = 2)

# Summarize posteriors for hyperparams only
print(out.model2$summary[10:11,], dig = 4)
         mean      sd     2.5%    25%    50%    75%  97.5%  Rhat n.eff
mu      2.659 0.05154 2.55047 2.6270 2.6605 2.6910 2.7570 1.001  1500
sd.year 0.150 0.04685 0.08888 0.1191 0.1396 0.1693 0.2669 1.002  1200
```

Third, we fit model 3, which has a mean, and *both* random site and random year effects, which are additive. This is GLMM2 on p. 103 of the BPA book and also the model in the previous exercise.

```
# Specify model 3 in BUGS language
sink("model3.txt")
cat("
model {

# Priors
for (j in 1:nsite){
   alpha[j] ~ dnorm(mu, tau.site)   # Random site effects
   }
mu ~ dnorm(0, 0.01)
tau.site <- pow(sd.site, -2)
sd.site ~ dunif(0, 5)

for (i in 1:nyear){
   beta[i] ~ dnorm(0, tau.year)   # Random year effects
   }
tau.year <- pow(sd.year, -2)
sd.year ~ dunif(0, 3)

# Likelihood
for (i in 1:nyear){
   for (j in 1:nsite){
      C[i,j] ~ dpois(lambda[i,j])
      lambda[i,j] <- exp(log.lambda[i,j])
      log.lambda[i,j] <- alpha[j] + beta[i]
      }  #j
   }  #i
}
",fill = TRUE)
sink()


# Bundle data
win.data <- list(C = t(C), nsite = nrow(C), nyear = ncol(C))

# Initial values (A)
inits <- function() list(mu = runif(1, 0, 4))

# Initial values (B)
inits <- function() list(mu = runif(1, 0, 4), alpha = runif(235, -2, 2),
beta = runif(9, -1, 1))

# Parameters monitored
params <- c("mu", "sd.year", "sd.site", "alpha", "beta")

# MCMC settings
ni <- 10000
```

```
nt <- 2
nb <- 2000
nc <- 3

# Call WinBUGS from R (BRT 6 min)
out.model3 <- bugs(win.data, inits, params, "model3.txt", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory =
bugs.dir, working.directory = getwd())


# Summarize posteriors for all quantities
print(out.model3, dig = 2)

# Summarize posteriors for hyperparams only
print(out.model3$summary[1:3,], dig = 4)
           mean      sd    2.5%    25%    50%    75%  97.5%  Rhat n.eff
mu       2.0786 0.10141 1.87500 2.0130 2.0780 2.1480 2.2730 1.005   520
sd.year  0.1541 0.04904 0.08988 0.1209 0.1442 0.1763 0.2762 1.001  7400
sd.site  1.3298 0.06613 1.20400 1.2840 1.3280 1.3740 1.4660 1.001  7900
```

Strikingly, without initial values for the random effects alpha and beta (i.e., with inits function A), convergence is very, VERY bad, indeed. In contrast, with inits function B (i.e., providing starting values for the random effects that are somewhere near their solutions), convergence is achieved without a problem with the specified chain and burnin lengths.

Fourth, we fit model 4, which has a mean plus random site-by-year effects. If we want to monitor the random effects, we have 9*235+2 quantities to estimate. To avoid R choking on very large vectors, we have to be frugal with our choice of MCMC settings.

```
# Specify model 4 in BUGS language
sink("model4.txt")
cat("
model {

# Priors
for (i in 1:nyear){
   for (j in 1:nsite){
      alpha[i,j] ~ dnorm(mu, tau)   # Random site-year effects
      } #j
   } #i
mu ~ dnorm(0, 0.01)
tau <- pow(sd, -2)
sd ~ dunif(0, 5)

# Likelihood
for (i in 1:nyear){
   for (j in 1:nsite){
      C[i,j] ~ dpois(lambda[i,j])
      lambda[i,j] <- exp(log.lambda[i,j])
      log.lambda[i,j] <- alpha[i, j]
      }  #j
   }  #i
}
",fill = TRUE)
sink()


# Bundle data
win.data <- list(C = t(C), nsite = nrow(C), nyear = ncol(C))
```

```
# Initial values (A)
inits <- function() list(mu = runif(1, 0, 4))

# Initial values (B)
inits <- function() list(mu = runif(1, 0, 4), alpha = runif(9*235, -2, 2))

# Parameters monitored
params <- c("mu", "sd", "alpha")

# MCMC settings (so chosen that we don't have to save too many samples)
ni <- 10000
nt <- 16
nb <- 2000
nc <- 3

# Call WinBUGS from R (BRT 9 min)
out.model4 <- bugs(win.data, inits, params, "model4.txt", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory =
bugs.dir, working.directory = getwd())

# Summarize posteriors for all quantities
print(out.model4, dig = 2)

# Summarize posteriors for hyperparams only
print(out.model4$summary[1:2,], dig = 4)
    mean      sd  2.5%   25%   50%   75% 97.5% Rhat n.eff
mu 2.144 0.02818 2.087 2.126 2.144 2.162 2.199    1  1500
sd 1.184 0.02444 1.138 1.168 1.184 1.200 1.232    1  1500
```

So what is the meaning of these random effects in the four models ? In model 1, we assume that the expected count, on the log scale, for each site is a random draw from a normal distribution with mean = 2.093 and standard deviation = 1.327. This model assumes that years don't differ in their expected counts. In model 2, we assume that the expected count, on the log scale, for each year is a random draw from a normal distribution with mean 2.659 and standard deviation 0.150. This model assumes that sites don't differ in the expected counts. In model 3, the expected count, on the log scale, varies around the grand mean of 2.079 and is affected by additive site and year effects, which are both random draws from zero-mean normal distributions with standard deviations of 1.330 and 0.1541, respectively. Thus, this model assumes that both sites and years differ in their expected counts, and that sites and years affect the expected count in different ways. In model 4, the expected count, on the log scale, is a random draw from a single normal distribution with mean 2.144 and standard deviation 1.184. In this model also, both sites and years are allowed to affect the expected counts, but there is no longer a separate effect of site and year. Instead, whenever site and/or year changes, we get a different draw from that single normal distribution.

Exercise 5

Task: Fixed and random: Convert these models into fixed-effects models, i.e., specify each of the following models without making the assumption that a set of effects come from a common distribution: site, year, site + year, observer, site + observer, year + observer, site + year + observer, first-year indicator, … Comparing the fixed- and the random-effects version of a model as specified in the BUGS language will be very helpful for your understanding of mixed models!

Solution: We here show the solutions for two of the mentioned models: year + observer and site + year + observer. We will first fit the model with all sets of effects random and then, with all of them fixed. We will choose yet another model nomenclature and call the first model 1 and the second model 2. To denote the fixed and the random-effects variants of these models, we will add a suffix F and R to the model names.

First, the model with random year and random observer effects; model 1R. Remember that the variable 'newobs' indexes the 271 plus 1 distinct observers (level 272 of the factor is a catch-all for when the observer ID was not known; see p. 98 in the book)

```
# Specify model 1R in BUGS language
sink("model1R.txt")
cat("
model {

# Priors
for (i in 1:nyear){
   beta[i] ~ dnorm(mu, tau.year)   # Random year effects
   }
mu ~ dnorm(0, 0.01)
tau.year <- pow(sd.year, -2)
sd.year ~ dunif(0, 3)

for (k in 1:nobs){
   gamma[k] ~ dnorm(0, tau.obs)   # Random observer effects
   }
tau.obs <- pow(sd.obs, -2)
sd.obs ~ dunif(0, 3)

# Likelihood
for (i in 1:nyear){
   for (j in 1:nsite){
      C[i,j] ~ dpois(lambda[i,j])
      lambda[i,j] <- exp(log.lambda[i,j])
      log.lambda[i,j] <- beta[i] + gamma[newobs[i,j]]
      }  #j
   }  #i
}
",fill = TRUE)
sink()


# Bundle data
win.data <- list(C = t(C), newobs = t(newobs), nsite = nrow(C), nyear =
ncol(C), nobs = length(unique(c(newobs))))

# Initial values (A)
inits <- function() list(mu = runif(1, 1, 3))

# Initial values (B)
inits <- function() list(mu = runif(1, 1, 3), beta = runif(9, -1, 1), gamma
= runif(272, -1, 1))

# Parameters monitored
params <- c("mu", "sd.year", "sd.obs", "beta", "gamma")

# MCMC settings
ni <- 10000
nt <- 2
```

```
nb <- 2000
nc <- 3
# MCMC test settings
ni <- 120
nt <- 2
nb <- 20
nc <- 3

# Call WinBUGS from R (BRT 6 min)
out.model1R <- bugs(win.data, inits, params, "model1R.txt", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory =
bugs.dir, working.directory = getwd())

# Summarize posteriors for all quantities
print(out.model1R, dig = 2)

# Summarize posteriors for hyperparams only
print(out.model1R$summary[1:3,], dig = 4)
          mean      sd   2.5%    25%    50%    75%  97.5%  Rhat n.eff
mu      2.1400 0.08512 1.9680 2.0840 2.1420 2.1970 2.3020 1.012   300
sd.year 0.1489 0.04910 0.0863 0.1158 0.1384 0.1699 0.2727 1.001 12000
sd.obs  1.1344 0.05826 1.0270 1.0940 1.1320 1.1720 1.2540 1.001  3700
```

This model converges reasonably rapidly again only with inits function B. Interestingly, the variability among observers is estimated at about the same magnitude as the variability among sites in model 3 in exercise 4 in this chapter. It is likely that much of the variability among sites is soaken up by the observer effects, which are confounded to some degree with the site effects. It will be interesting to see how the observer and the site effects are separated in model 2 afterwards.

Next, we change the effects of both the year and the observer factors to fixed and fit model 1F. We have to do two things for this: first, chose independent, vague priors for the year and the observer effects (i.e., without shared hyperparameters) and second, drop the intercept mu and introduce some constraints on the fixed effects to avoid parameter redundancy. The linear predictor of this model has the form of a two-way ANOVA with main effects. We will introduce a corner constraint on one of the set of effects and set the last observer effect to zero. This makes sense, because the last 'observer' is the catch-all group for those surveys for which we don't actually know how made them. The beta terms will then be the year effects for those unknown observers and the gamma terms will measure the difference in the log of the expected counts between the other observers and that level of 'observer 272'.

Note the inits function, which must give an NA as initial value for the effect of observer 272.

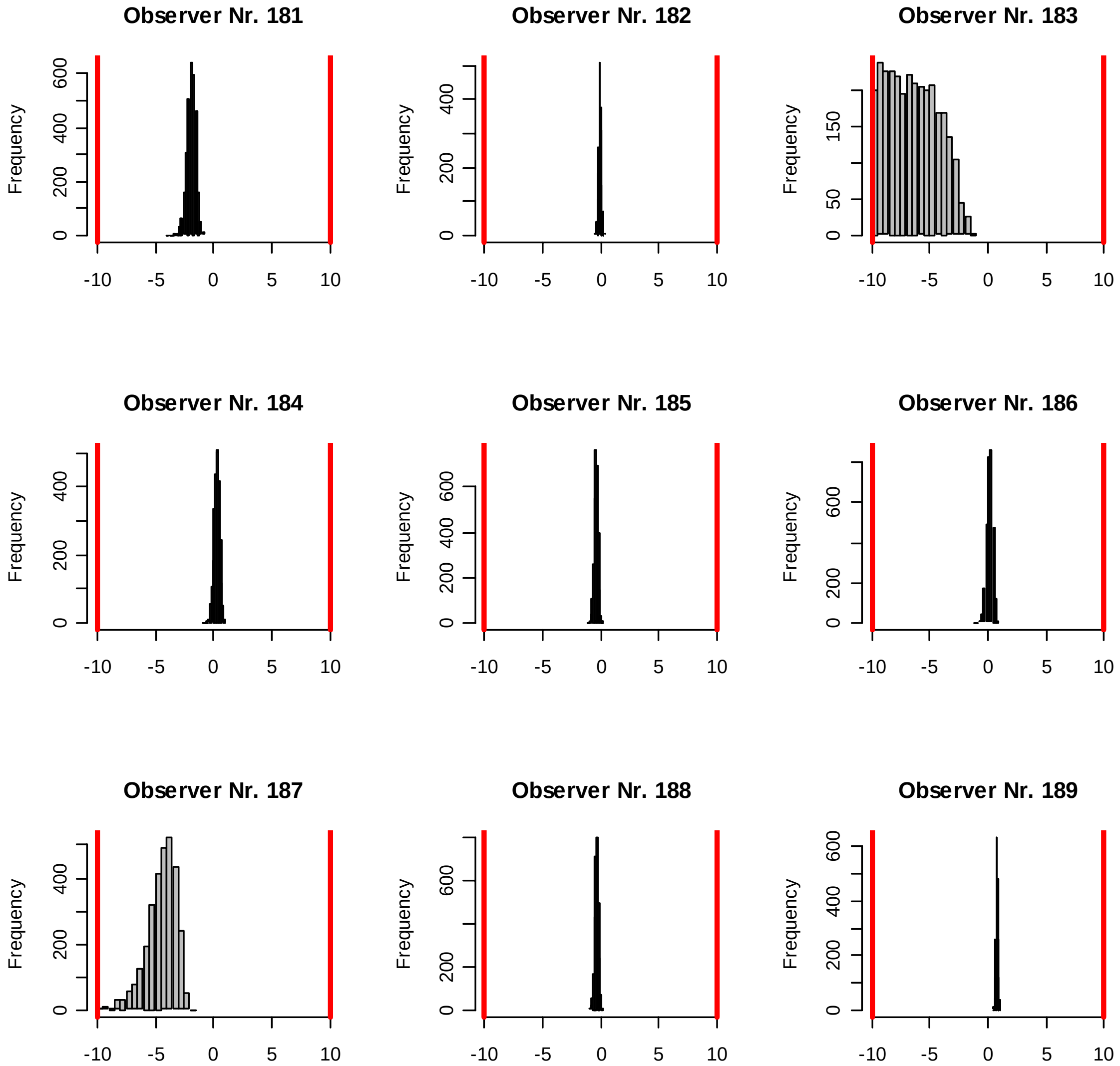
```
# Specify model 1R in BUGS language
sink("model1F.txt")
cat("
model {

# Priors
for (i in 1:nyear){
   beta[i] ~ dnorm(0, 0.01)   # Fixed year effects
   }

for (k in 1:(nobs-1)){
#  gamma[k] ~ dnorm(0, 0.1)   # Fixed observer effects
   gamma[k] ~ dunif(-10, 10)   # Fixed observer effects
```

```
   }
gamma[nobs] <- 0                # Set effect of last observer to zero

# Likelihood
for (i in 1:nyear){
   for (j in 1:nsite){
      C[i,j] ~ dpois(lambda[i,j])
      lambda[i,j] <- exp(log.lambda[i,j])
      log.lambda[i,j] <- beta[i] + gamma[newobs[i,j]]
      }  #j
   }  #i
}
",fill = TRUE)
sink()


# Bundle data
win.data <- list(C = t(C), newobs = t(newobs), nsite = nrow(C), nyear =
ncol(C), nobs = length(unique(c(newobs))))

# Initial values
inits <- function() list(beta = runif(9, -1, 1))

# Parameters monitored
params <- c("beta", "gamma")

# MCMC settings
ni <- 6000
nt <- 5
nb <- 1000
nc <- 3

# Call WinBUGS from R (BRT 3 min)
out.model1F <- bugs(win.data, inits, params, "model1F.txt", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory =
bugs.dir, working.directory = getwd())

# Summarize posteriors for all quantities
print(out.model1F, dig = 2)

# Summarize posteriors for year effects only
print(out.model1F$summary[1:9,], dig = 3)
        mean     sd 2.5%  25%  50%  75% 97.5% Rhat n.eff
beta[1] 2.50 0.0234 2.46 2.49 2.50 2.52  2.55    1  2900
beta[2] 2.57 0.0225 2.52 2.55 2.57 2.58  2.61    1  1400
beta[3] 2.33 0.0233 2.28 2.31 2.33 2.34  2.37    1  3000
beta[4] 2.38 0.0233 2.33 2.36 2.38 2.39  2.42    1  3000
beta[5] 2.56 0.0220 2.52 2.55 2.56 2.58  2.60    1  2500
beta[6] 2.61 0.0216 2.57 2.60 2.61 2.63  2.65    1  3000
beta[7] 2.70 0.0211 2.66 2.69 2.71 2.72  2.75    1  3000
beta[8] 2.37 0.0238 2.33 2.36 2.38 2.39  2.42    1  3000
beta[9] 2.47 0.0227 2.43 2.46 2.47 2.49  2.52    1   640
```

We can plot the posterior distributions of all the observer effects to check whether the choice of a U(-10, 10) prior was too informative (given that we intended the choice to be vague).

```
par(mfrow = c(3, 3))
for (i in 1:271){
   hist(out.model1F$sims.list$gamma[,i], xlim = c(-10, 10), col = "grey",
   main = paste("Observer Nr.", i), xlab = "")
```

```
abline(v = c(-10, 10), col = "red", lwd = 3)
browser()
}
```



There only are about 9 observers where the posterior distribution of the effect gamma is seriously hitting one of the bounds of the uniform prior, usually the lower. One example is observer 183 in the above plot. In addition, there are about 11 observers, where the posterior of gamma appears to be a little influenced by our choice of prior for gamma. One example is observer 187 in the plot. Overall, we are therefore not too concerned about the vagueness of the prior. If we were, we could always refit the model with a vaguer prior for the gammas.

Next, we fit a model with random site, random year and random observer effects and call it model 2R. It will be interesting to see how the variability in the log of the expected counts is partitioned among sites and among observers.

```
# Specify model 2R in BUGS language
sink("model2R.txt")
cat("
model {

# Priors
```

```
for (j in 1:nsite){
   alpha[j] ~ dnorm(mu, tau.site)   # Random site effects
   }
mu ~ dnorm(0, 0.01)
tau.site <- pow(sd.site, -2)
sd.site ~ dunif(0, 5)

for (i in 1:nyear){
   beta[i] ~ dnorm(0, tau.year)   # Random year effects
   }
tau.year <- pow(sd.year, -2)
sd.year ~ dunif(0, 3)

for (k in 1:nobs){
   gamma[k] ~ dnorm(0, tau.obs)   # Random observer effects
   }
tau.obs <- pow(sd.obs, -2)
sd.obs ~ dunif(0, 3)

# Likelihood
for (i in 1:nyear){
   for (j in 1:nsite){
      C[i,j] ~ dpois(lambda[i,j])
      lambda[i,j] <- exp(log.lambda[i,j])
      log.lambda[i,j] <- alpha[j] + beta[i] + gamma[newobs[i,j]]
      }  #j
   }  #i
}
",fill = TRUE)
sink()


# Bundle data
win.data <- list(C = t(C), newobs = t(newobs), nsite = nrow(C), nyear =
ncol(C), nobs = length(unique(c(newobs))))

# Initial values (B)
inits <- function() list(mu = runif(1, 0, 4), alpha = runif(235, -2, 2),
beta = runif(9, -1, 1))

# Parameters monitored
params <- c("mu", "sd.year", "sd.site", "sd.obs", "alpha", "beta", "gamma")

# MCMC settings
ni <- 10000
nt <- 2
nb <- 2000
nc <- 3

# Call WinBUGS from R (BRT 11 min)
out.model2R <- bugs(win.data, inits, params, "model2R.txt", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory =
bugs.dir, working.directory = getwd())

# Summarize posteriors for all quantities
print(out.model2R, dig = 2)

# Summarize posteriors for hyperparams only
print(out.model2R$summary[1:4,], dig = 3)
         mean     sd   2.5%   25%   50%   75% 97.5% Rhat n.eff
mu      2.107 0.1112 1.8940 2.033 2.105 2.179 2.337 1.03    77
sd.year 0.152 0.0557 0.0868 0.117 0.140 0.172 0.296 1.02   180
```

```
sd.site 1.303 0.0669 1.1800 1.257 1.301 1.348 1.439 1.00  1700
sd.obs  0.337 0.0321 0.2774 0.314 0.336 0.358 0.403 1.00  1500
```

Interestingly, even though the observer-induced variability is estimated at a much larger value when random site effects are not allowed for, the site variability is not estimated at a smaller value when observer effects are also estimated.

Now we turn this model into a fixed-effects model. Its linear predictor has the form of a three-way, main-effects ANOVA. Again, we introduce 'corner constraints' on parameters of all but factor to avoid fitting an overparameterized model. With these constraints, alpha will be the site effects in year 1 and for observer 272. Convergence with this model is hard to obtain if the priors are chosen too vague.

```
# Specify model 2F in BUGS language
sink("model2F.txt")
cat("
model {

# Priors
for (j in 1:nsite){
   alpha[j] ~ dnorm(0, 0.01)   # Fixed site effects
   }

for (i in 2:nyear){
   beta[i] ~ dnorm(0, 0.1)   # Fixed year effects
   }
beta[1] <- 1                # Effect of first year set to zero

for (k in 1:(nobs-1)){
#   gamma[k] ~ dunif(-10, 10)   # Fixed observer effects
   gamma[k] ~ dnorm(0, 0.1)   # Fixed observer effects
   }
gamma[nobs] <- 0                # Effect of last observer set to zero

# Likelihood
for (i in 1:nyear){
   for (j in 1:nsite){
      C[i,j] ~ dpois(lambda[i,j])
      lambda[i,j] <- exp(log.lambda[i,j])
      log.lambda[i,j] <- alpha[j] + beta[i] + gamma[newobs[i,j]]
      }  #j
   }  #i
}
",fill = TRUE)
sink()


# Bundle data
win.data <- list(C = t(C), newobs = t(newobs), nsite = nrow(C), nyear =
ncol(C), nobs = length(unique(c(newobs))))

# Initial values (NOTE: NA inits for params fixed at zero)
inits <- function() list(alpha = runif(235, 1, 3), beta = c(NA, runif(8, -
1, 1)), gamma = c(runif(271, -1, 1), NA))

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

# MCMC settings
```

```
ni <- 6000
nt <- 5
nb <- 1000
nc <- 3

# Call WinBUGS from R (BRT 7 min)
out.model2F <- bugs(win.data, inits, params, "model2F.txt", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory =
bugs.dir, working.directory = getwd())

# Summarize posteriors for all quantities
print(out.model2F, dig = 2)
```

Convergence is not fantastic. For instance, for 166 parameters, the value of Rhat is greater than 1.2.

```
length(which(out.model2F$summary[,8] > 1.2))
 [1] 166
```

Nevertheless,we leave it at that, because the main aim of this exercise was to show, again, how simple it is, conceptually, to move from a random- to a fixed-effects model.

Exercise 6

Task: Covariates: In the tit model in section 4.3.2, add the log-linear effects of elevation and forest cover in the linear predictor of abundance. Also add in squared effects of these covariates.

Solution: We could take almost any of the of the models in section 4.3.2 and add the effects of these two covariates, with the exception of the models with a fixed site effect. The reason for this is that with 235 fixed site *and* 2 (or even more) site-specific covariate effects we would try to estimate more parameters than what we possibly can with data from 235 sites. Thus, we have to introduce these covariates either in a model with random instead of fixed site effects or with no site effects at all. What we will do here is to fit them within model GLMM2 (see p. 103 in the BPA book and also exercises 3 and 4 in this chapter). Comparing the magnitude of the site variance with and without the covariates will provide a measure of their explanatory power (see section 7.4.3 in the BPA book).

We refit the model from exercise 4 first and then, in a second step, add in the two sets of covariate effects. We will choose yet another numbering for the models and call the former model 0 and the latter models 1 and 2.

```
# Specify model 0 in BUGS language
sink("model0.txt")
cat("
model {

# Priors
for (j in 1:nsite){
   alpha[j] ~ dnorm(mu, tau.site)    # Random site effects with grand mean
   }
mu ~ dnorm(0, 0.01)
tau.site <- 1/ (sd.site * sd.site)
```

```
sd.site ~ dunif(0, 5)

for (i in 1:nyear){
   beta[i] ~ dnorm(0, tau.year)        # Random year effects
   }
tau.year <- 1/ (sd.year * sd.year)
sd.year ~ dunif(0, 3)

# Likelihood
for (i in 1:nyear){
   for (j in 1:nsite){
       C[i,j] ~ dpois(lambda[i,j])
      lambda[i,j] <- exp(log.lambda[i,j])
      log.lambda[i,j] <- alpha[j] + beta[i]
      }  #j
   }  #i
}
",fill = TRUE)
sink()

# Bundle data
win.data <- list(C = t(C), nsite = nrow(C), nyear = ncol(C))

# Initial values
inits <- function() list(mu = runif(1, 0, 4), alpha = runif(235, -2, 2),
beta = runif(9, -1, 1))

# Parameters monitored
params <- c("mu", "sd.site", "sd.year", "alpha", "beta")

# MCMC settings
ni <- 10000
nt <- 2
nb <- 2000
nc <- 3

# Call WinBUGS from R (BRT 6 min)
out0 <- bugs(win.data, inits, params, "model0.txt", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory =
bugs.dir, working.directory = getwd())

# Summarize posteriors of hyperparameters
print(out0$summary[1:3,], dig = 4)
          mean      sd    2.5%    25%    50%    75%  97.5%  Rhat n.eff
mu      2.0879 0.09955 1.89200 2.0210 2.0880 2.1550 2.2810 1.002  2500
sd.site 1.3285 0.06539 1.20700 1.2830 1.3260 1.3710 1.4620 1.001 12000
sd.year 0.1532 0.04840 0.09046 0.1203 0.1435 0.1747 0.2736 1.001  7300
```

Then, we fit two models (model 1 and model 2) with both linear and quadratic terms of the forest cover and the elevation covariates, respectively. We first prepare the data and then write the model. We have to standardise the covariates. Six sites have missing forest covariate values. Missing covariates are not dealt with automatically in WinBUGS. We will mean-impute them, which is equivalent to setting them to zero after standardisation.

Running the model first yielded an “undefined real result” trap. Choosing less dispersed initial values helped.

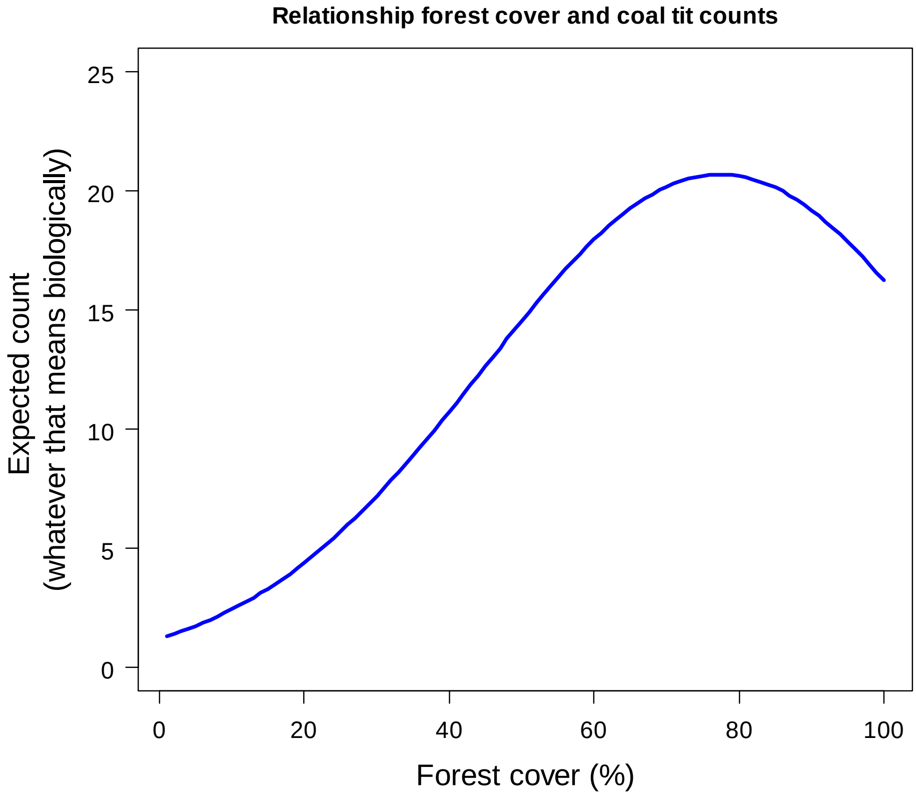
```
# Bundle data (including covariates)
forst <- as.numeric(scale(tits$forest))   # forest_standardised
```

```
forst[is.na(forst)] <- 0
win.data <- list(C = t(C), nsite = nrow(C), nyear = ncol(C), forst = forst)

# Specify model 1 in BUGS language
sink("model1.txt")
cat("
model {

# Priors
for (j in 1:nsite){
   alpha[j] ~ dnorm(mu, tau.site)    # Random site effects with grand mean
   }
mu ~ dnorm(0, 0.01)
tau.site <- 1/ (sd.site * sd.site)
sd.site ~ dunif(0, 5)

for (i in 1:nyear){
   beta[i] ~ dnorm(0, tau.year)        # Random year effects
   }
tau.year <- 1/ (sd.year * sd.year)
sd.year ~ dunif(0, 3)

gamma1 ~ dnorm(0, 0.01)                # Linear effect of forest
gamma2 ~ dnorm(0, 0.01)                # Quadratic effect of forest

# Likelihood
for (i in 1:nyear){
   for (j in 1:nsite){
       C[i,j] ~ dpois(lambda[i,j])
      lambda[i,j] <- exp(log.lambda[i,j])
      log.lambda[i,j] <- alpha[j] + beta[i] + gamma1 * forst[j] + gamma2 *
pow(forst[j],2)
      }  #j
   }  #i
}
",fill = TRUE)
sink()

# Initial values
inits <- function() list(mu = runif(1, 1, 2), alpha = runif(235, -1, 1),
beta = runif(9, -1, 1), gamma1 = runif(1), gamma2 = runif(1))

# Parameters monitored
params <- c("mu", "sd.site", "sd.year", "alpha", "beta", "gamma1",
"gamma2")

# MCMC settings
ni <- 10000
nt <- 2
nb <- 2000
nc <- 3

# MCMC settings
ni <- 120   ;   nt <- 2   ;   nb <- 20   ;   nc <- 3

# Call WinBUGS from R (BRT 7 min)
out1 <- bugs(win.data, inits, params, "model1.txt", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory =
bugs.dir, working.directory = getwd())

# Summarize posteriors of hyperparameters
print(out1$summary[c(1:3, 248:249),], dig = 3)
```

```
          mean     sd    2.5%    25%    50%    75%  97.5% Rhat n.eff
mu       2.389 0.1118  2.1730  2.314  2.386  2.465  2.611 1.04    54
sd.site  0.991 0.0516  0.8943  0.955  0.990  1.024  1.096 1.00 12000
sd.year  0.155 0.0497  0.0902  0.121  0.145  0.177  0.283 1.00   930
gamma1   0.897 0.0670  0.7705  0.850  0.894  0.942  1.034 1.01   350
gamma2  -0.313 0.0696 -0.4630 -0.356 -0.308 -0.265 -0.184 1.04    72
```

Convergence is not fantastic, but acceptable. Both the linear and the quadratic effect of forest is "significant", in the sense that the 95% CRI does not include zero. To find out how the effect of forest cover on the counts looks like, we produce a plot. This is also an exercise in not getting lost with transformed covariates.

```
forest.pred.original <- 1:100
forest.pred.st <- (forest.pred.original - mean(tits$forest, na.rm = TRUE))
/ sd(tits$forest, na.rm = TRUE)
pred.count <- exp(out1$mean$mu + out1$mean$gamma1 * forest.pred.st +
out1$mean$gamma2 * forest.pred.st^2)
par(mar = c(5,6,3,2), cex.main = 1.2, cex.lab = 1.5, cex.axis = 1.2)
plot(forest.pred.original, pred.count, main = "Relationship forest cover
and coal tit counts", xlab = "Forest cover (%)", ylab = "Expected count
\n(whatever that means biologically)", las = 1, type = "l", col = "blue",
lwd = 3, ylim = c(0,25))
```



Here is the proportion of the variability among sites in counts that is explained by forest cover (see p. 189 in the book).

```
(out0$mean$sd.site^2 - out1$mean$sd.site^2) / out0$mean$sd.site^2
 [1] 0.4435187
```

Thus, almost half of the site-by-site variability in the tit counts is explained by forest cover. This is not surprising perhaps for a typical forest bird. We leave it as a task for the tit biologists to explain the decline in the expected counts beyond values of forest cover of about 80 %.

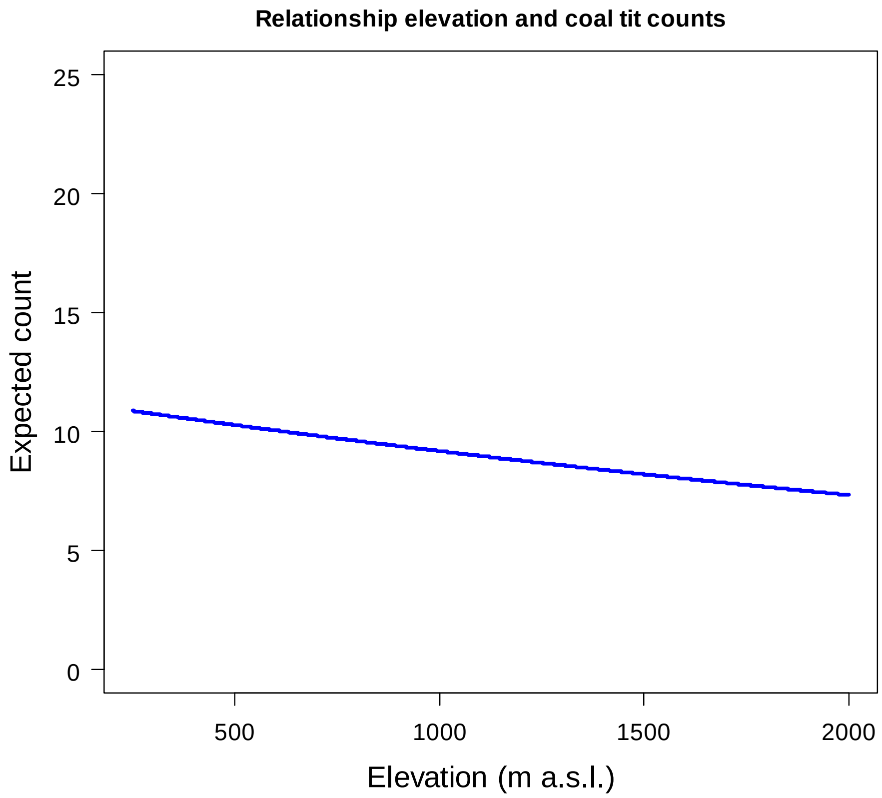
Next, the fit the model with the linear and quadratic effects of elevation.

```
# Bundle data (including covariates)
elest <- as.numeric(scale(tits$elevation))      # elevation_standardised
win.data <- list(C = t(C), nsite = nrow(C), nyear = ncol(C), elest = elest)

# Specify model 2 in BUGS language
sink("model2.txt")
cat("
model {

# Priors
for (j in 1:nsite){
   alpha[j] ~ dnorm(mu, tau.site)    # Random site effects with grand mean
   }
mu ~ dnorm(0, 0.01)
tau.site <- 1/ (sd.site * sd.site)
sd.site ~ dunif(0, 5)

for (i in 1:nyear){
   beta[i] ~ dnorm(0, tau.year)        # Random year effects
   }
tau.year <- 1/ (sd.year * sd.year)
sd.year ~ dunif(0, 3)

gamma3 ~ dnorm(0, 0.01)                # Linear effect of elevation
gamma4 ~ dnorm(0, 0.01)                # Quadratic effect of elevation

# Likelihood
for (i in 1:nyear){
   for (j in 1:nsite){
       C[i,j] ~ dpois(lambda[i,j])
      lambda[i,j] <- exp(log.lambda[i,j])
      log.lambda[i,j] <- alpha[j] + beta[i] + gamma3 * elest[j] + gamma3 *
pow(elest[j],2)
      }  #j
   }  #i
}
",fill = TRUE)
sink()

# Initial values
inits <- function() list(mu = runif(1, 1, 2), alpha = runif(235, -1, 1),
beta = runif(9, -1, 1), gamma3 = runif(1), gamma4 = runif(1))

# Parameters monitored
params <- c("mu", "sd.site", "sd.year", "alpha", "beta", "gamma3",
"gamma4")

# MCMC settings
ni <- 10000
nt <- 2
nb <- 2000
nc <- 3

# Call WinBUGS from R (BRT 7 min)
out2 <- bugs(win.data, inits, params, "model2.txt", n.chains = nc,
```

```
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory =
bugs.dir, working.directory = getwd())

# Summarize posteriors of hyperparameters
print(out2$summary[c(1:3, 248:249),], dig = 3)
            mean     sd     2.5%    25%    50%     75%   97.5% Rhat n.eff
mu        2.2008 0.1149   1.9720  2.124  2.200  2.2770  2.4230 1.00  1800
sd.site   1.3179 0.0669   1.1930  1.271  1.315  1.3610  1.4570 1.00  2400
sd.year   0.1560 0.0518   0.0901  0.121  0.145  0.1782  0.2900 1.00  1000
gamma3   -0.1227 0.0498  -0.2152 -0.158 -0.123 -0.0892 -0.0219 1.01   200
gamma4   -0.0241 9.9051 -19.5600 -6.575 -0.193  6.5643 19.6600 1.00  8000
```

Only the linear effect of elevation has a 95% CRI that does not cover zero. To find out how the effect of elevation on the counts look like, we produce another plot.

```
elevation.pred.original <- 250:2000
elevation.pred.st <- (elevation.pred.original - mean(tits$elevation)) /
sd(tits$elevation)
pred.count <- exp(out2$mean$mu + out2$mean$gamma3 * elevation.pred.st)
par(mar = c(5,6,3,2), cex.main = 1.2, cex.lab = 1.5, cex.axis = 1.2)
plot(elevation.pred.original, pred.count, main = "Relationship elevation
and coal tit counts", xlab = "Elevation (m a.s.l.)", ylab = "Expected
count", las = 1, type = "l", col = "blue", lwd = 3, ylim = c(0, 25))
```



Although the effect of elevation (linear) is “significant” based on a 95% CRI, elevation explains barely 2 % of the variability in counts among sites.

```
(out0$mean$sd.site^2 - out2$mean$sd.site^2) / out0$mean$sd.site^2
```

Exercise 7

Task: Take the model and the data from section 4.3.1.

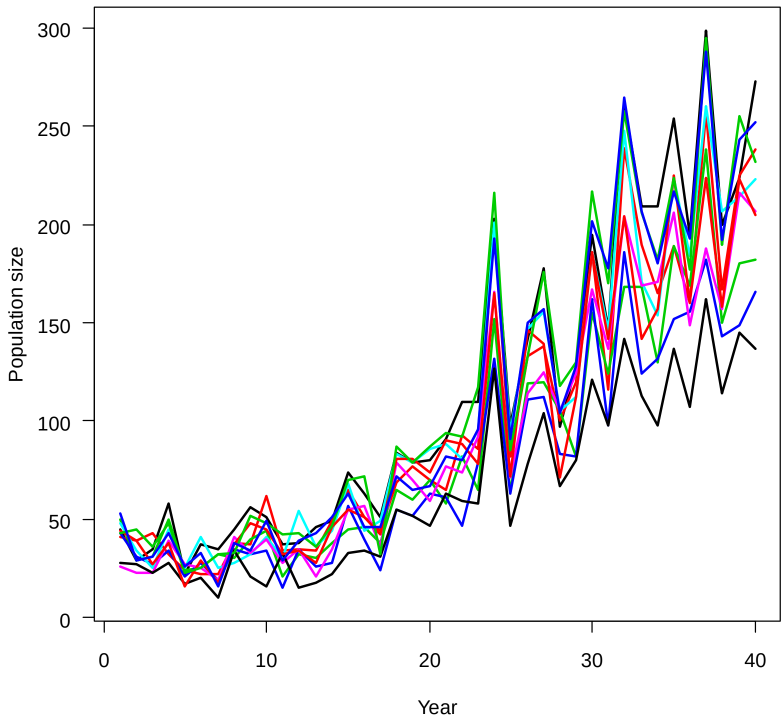
a. Drop the quadratic and the cubic polynomial terms of year.

b. Next, turn the random-effects Poisson GLM into a fixed- (year and site) effects Poisson GLM, i.e., drop the randomness assumption for site and year. Hint: your model will then be a two-way main-effects ANOVA, so you will have to constrain some parameters to make it identifiable.

Solution: We assume that you have a data set for 10 sites in your R workspace, i.e., you have executed this R code,

```
data <- data.fn(nsite = 10, nyear = 40, sd.site = 0.3, sd.year = 0.2),
```

which will result in a neat picture like this:



a. By dropping the two polynomial terms of year we are left with a simple linear regression of year (for the expected counts on the log scale), with additional random site and random year effects. Some initial trial runs suggest that we have to make the uniform priors on the random effects wider.

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

# Priors
for (j in 1:nsite){
   alpha[j] ~ dnorm(mu, tau.alpha)
```

```
   }
mu ~ dnorm(0, 0.01)
tau.alpha <- 1 / (sd.alpha*sd.alpha)
sd.alpha ~ dunif(0, 2)
beta ~ dnorm(0, 0.01)

tau.year <- 1 / (sd.year*sd.year)
sd.year ~ dunif(0, 5)

# Likelihood
for (i in 1:nyear){
   eps[i] ~ dnorm(0, tau.year)
   for (j in 1:nsite){
      C[i,j] ~ dpois(lambda[i,j])
      lambda[i,j] <- exp(log.lambda[i,j])
      log.lambda[i,j] <- alpha[j] + beta * year[i] + eps[i]
      }  #j
   }  #i
}
",fill = TRUE)
sink()


# Bundle data
win.data <- list(C = data$C, nsite = ncol(data$C), nyear = nrow(data$C),
year = (data$year-20) / 20)

# Initial values
inits <- function() list(mu = runif(1, 0, 2), alpha = runif(data$nsite, -1,
1), beta = runif(1, -1, 1), sd.alpha = runif(1, 0, 0.1), sd.year = runif(1,
0, 0.1))

# Parameters monitored
params <- c("mu", "alpha", "beta", "sd.alpha", "sd.year")

# MCMC settings
ni <- 25000
nt <- 5
nb <- 15000
nc <- 3

# Call WinBUGS from R (BRT 4 min)
out.a <- bugs(win.data, inits, params, "GLMM_Poisson_a.txt", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory =
bugs.dir, working.directory = getwd())

# Summarize posteriors
print(out.a, dig = 2)
Inference for Bugs model at "GLMM_Poisson_a.txt", fit using WinBUGS,
 3 chains, each with 25000 iterations (first 15000 discarded), n.thin = 5
 n.sims = 6000 iterations saved
             mean   sd   2.5%    25%    50%    75%   97.5% Rhat n.eff
mu           4.28 0.08   4.11   4.22   4.28   4.33   4.43 1.00  1100
alpha[1]     4.47 0.04   4.38   4.44   4.47   4.50   4.56 1.01   430
alpha[2]     4.33 0.04   4.24   4.30   4.34   4.36   4.42 1.01   560
alpha[3]     4.19 0.05   4.10   4.16   4.19   4.22   4.28 1.00   510
alpha[4]     4.08 0.05   3.99   4.05   4.08   4.11   4.17 1.00   690
alpha[5]     4.38 0.04   4.28   4.35   4.38   4.41   4.46 1.01   360
alpha[6]     4.24 0.04   4.15   4.21   4.25   4.27   4.33 1.00   830
alpha[7]     3.91 0.05   3.81   3.88   3.91   3.94   3.99 1.00   760
alpha[8]     4.29 0.05   4.20   4.26   4.29   4.32   4.37 1.00   680
alpha[9]     4.45 0.04   4.36   4.42   4.45   4.48   4.54 1.00   680
```

```
alpha[10]    4.41 0.04    4.32    4.38    4.42    4.44    4.50 1.00   650
beta         1.19 0.07    1.05    1.15    1.19    1.24    1.33 1.02    90
sd.alpha     0.21 0.06    0.13    0.17    0.20    0.24    0.37 1.00  1400
sd.year      0.26 0.03    0.21    0.24    0.26    0.28    0.33 1.00  2100
deviance  2872.64 9.89 2855.00 2866.00 2872.00 2879.00 2894.00 1.00  6000
```

b. To further modify the model from exercise 7a into one that has a linear predictor representing a two-way, main effects ANOVA, we need to drop the linear effect of year and change the effects of the year and site factors from fixed to random. Since we have done this so many times now, this should be easy.

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

# Priors
for (j in 1:nsite){
   alpha[j] ~ dnorm(0, 0.01)
   }

for (i in 2:nyear){
   eps[i] ~ dnorm(0, 0.01)
   }
eps[1] <- 0 # This is the corner constraint to avoid overparamaterization

# Likelihood
for (i in 1:nyear){
   for (j in 1:nsite){
      C[i,j] ~ dpois(lambda[i,j])
      lambda[i,j] <- exp(log.lambda[i,j])
      log.lambda[i,j] <- alpha[j] + eps[i]
      }  #j
   }  #i
}
",fill = TRUE)
sink()


# Bundle data
win.data <- list(C = data$C, nsite = ncol(data$C), nyear = nrow(data$C))

# Initial values
inits <- function() list(alpha = runif(data$nsite, -1, 1), eps = c(NA,
runif(39, 0, 1)))

# Parameters monitored
params <- c("alpha", "eps")

# MCMC settings
ni <- 25000
nt <- 5
nb <- 15000
nc <- 3

# Call WinBUGS from R (BRT 2 min)
out.b <- bugs(win.data, inits, params, "GLM_Poisson_b.txt", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory =
bugs.dir, working.directory = getwd())
```

```
# Summarize posteriors
print(out.b, dig = 2)
Inference for Bugs model at "GLM_Poisson_b.txt", fit using WinBUGS,
 3 chains, each with 25000 iterations (first 15000 discarded), n.thin = 5
 n.sims = 6000 iterations saved
             mean   sd    2.5%     25%     50%     75%   97.5% Rhat n.eff
alpha[1]     3.93 0.05    3.83    3.89    3.93    3.96    4.03 1.00   620
alpha[2]     3.79 0.05    3.69    3.75    3.79    3.82    3.89 1.00   510
alpha[3]     3.64 0.05    3.54    3.61    3.64    3.68    3.75 1.00   650
alpha[4]     3.53 0.05    3.43    3.49    3.53    3.56    3.63 1.00   570
alpha[5]     3.83 0.05    3.73    3.79    3.83    3.87    3.93 1.00   580
alpha[6]     3.70 0.05    3.60    3.66    3.70    3.73    3.80 1.00   690
alpha[7]     3.36 0.05    3.25    3.32    3.36    3.39    3.46 1.00   690
alpha[8]     3.74 0.05    3.64    3.70    3.74    3.78    3.84 1.00   540
alpha[9]     3.91 0.05    3.81    3.87    3.91    3.94    4.01 1.00   620
alpha[10]    3.87 0.05    3.77    3.83    3.87    3.90    3.97 1.00   530
eps[2]      -0.26 0.08   -0.41   -0.31   -0.26   -0.21   -0.12 1.00   910
eps[3]      -0.34 0.08   -0.49   -0.39   -0.34   -0.29   -0.19 1.00  2000
eps[4]      -0.02 0.07   -0.15   -0.07   -0.02    0.03    0.12 1.00   760
eps[5]      -0.63 0.08   -0.80   -0.69   -0.63   -0.58   -0.47 1.00  2000
eps[6]      -0.39 0.08   -0.55   -0.45   -0.39   -0.34   -0.25 1.00  1100
eps[7]      -0.62 0.08   -0.79   -0.67   -0.62   -0.56   -0.46 1.00  1500
eps[8]      -0.16 0.07   -0.30   -0.21   -0.16   -0.11   -0.02 1.00  1100
eps[9]      -0.09 0.07   -0.24   -0.14   -0.09   -0.04    0.05 1.00  1200
eps[10]      0.02 0.07   -0.12   -0.03    0.02    0.07    0.15 1.00   950
eps[11]     -0.34 0.08   -0.49   -0.39   -0.34   -0.29   -0.20 1.00  1600
eps[12]     -0.16 0.07   -0.30   -0.21   -0.16   -0.11   -0.02 1.00  2000
eps[13]     -0.28 0.08   -0.43   -0.33   -0.28   -0.23   -0.14 1.00  1400
eps[14]     -0.03 0.07   -0.17   -0.08   -0.03    0.02    0.11 1.00   770
eps[15]      0.33 0.06    0.20    0.28    0.33    0.37    0.45 1.00   640
eps[16]      0.18 0.07    0.04    0.13    0.18    0.22    0.31 1.00   940
eps[17]     -0.07 0.07   -0.21   -0.12   -0.07   -0.02    0.07 1.00   970
eps[18]      0.55 0.06    0.42    0.51    0.55    0.59    0.67 1.00   670
eps[19]      0.50 0.06    0.37    0.46    0.50    0.54    0.62 1.00   600
eps[20]      0.51 0.06    0.39    0.47    0.51    0.55    0.63 1.00   690
eps[21]      0.60 0.06    0.48    0.56    0.60    0.64    0.72 1.00   800
eps[22]      0.65 0.06    0.52    0.60    0.65    0.69    0.77 1.00  1200
eps[23]      0.73 0.06    0.61    0.69    0.73    0.77    0.85 1.01   410
eps[24]      1.42 0.06    1.32    1.39    1.42    1.46    1.53 1.00   780
eps[25]      0.60 0.06    0.47    0.55    0.60    0.64    0.72 1.01   470
eps[26]      1.10 0.06    0.99    1.07    1.10    1.14    1.22 1.00   700
eps[27]      1.20 0.06    1.09    1.16    1.20    1.24    1.31 1.00   550
eps[28]      0.82 0.06    0.70    0.77    0.82    0.86    0.93 1.00   580
eps[29]      0.96 0.06    0.84    0.92    0.95    1.00    1.07 1.01   460
eps[30]      1.44 0.05    1.33    1.40    1.44    1.48    1.54 1.00   680
eps[31]      1.17 0.06    1.05    1.13    1.17    1.21    1.28 1.00  1100
eps[32]      1.64 0.05    1.53    1.60    1.64    1.68    1.74 1.00   740
eps[33]      1.39 0.06    1.28    1.36    1.39    1.43    1.50 1.00   580
eps[34]      1.32 0.06    1.20    1.28    1.32    1.36    1.43 1.00   560
eps[35]      1.56 0.05    1.46    1.53    1.56    1.60    1.67 1.00   530
eps[36]      1.36 0.06    1.26    1.32    1.36    1.40    1.47 1.00   610
eps[37]      1.74 0.05    1.63    1.70    1.74    1.77    1.84 1.00   800
eps[38]      1.38 0.06    1.27    1.34    1.38    1.42    1.49 1.00   530
eps[39]      1.59 0.05    1.48    1.56    1.59    1.63    1.70 1.00   640
eps[40]      1.61 0.05    1.50    1.57    1.61    1.65    1.72 1.00   640
deviance  2872.44 9.92 2855.00 2866.00 2872.00 2879.00 2893.00 1.00  1200
```

<u>Exercise 8</u>

Task: Take the model and the data from Section 4.3.1 and model the response as coming from a normal distribution. This will clarify some of the differences between a normal and a Poisson GLMM.

Solution: We start again with the original model with cubic effects of year as a continuous explanatory variable plus random site effects. We note that with a normal response, it is no longer possible to have extra random year effects; the normal has a second parameter (apart from the mean) for the residuals and it would not be possible to estimate both the residual variance AND random year effects. Effectively, the residuals represent the random year effects. Here is how this model looks like. The upper bound of the uniform priors for the standard deviations of the site effects and the residuals need be upped quite a but, because now we no longer model log(expected counts) but directly the expected counts.

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

# Priors
for (j in 1:nsite){
   alpha[j] ~ dnorm(mu, tau.alpha)       # 4. Random site effects
   }
mu ~ dnorm(0, 0.01)                      # Hyperparameter 1
tau.alpha <- 1 / (sd.alpha*sd.alpha)     # Hyperparameter 2
sd.alpha ~ dunif(0, 500)
for (p in 1:3){
   beta[p] ~ dnorm(0, 0.01)
   }
tau.res <- pow(sd.res, -2)
sd.res ~ dunif(0, 100)

# Likelihood
for (i in 1:nyear){
   for (j in 1:nsite){
      C[i,j] ~ dnorm(lambda[i,j], tau.res)# 1. Distribution for random part
      lambda[i,j] <- alpha[j] + beta[1] * year[i] + beta[2] *
pow(year[i],2) + beta[3] * pow(year[i],3) # 3. Linear predictor includes
random site effects, link is identity
      }  #j
   }  #i
}
",fill = TRUE)
sink()

# Bundle data
win.data <- list(C = data$C, nsite = ncol(data$C), nyear = nrow(data$C),
year = (data$year-20) / 20)

# Initial values
inits <- function() list(mu = runif(1, 0, 2), alpha = runif(data$nsite, -1,
1), beta = runif(3, -1, 1), sd.alpha = runif(1, 0, 1), sd.res = runif(1, 0,
1))

# Parameters monitored (may want to add "lambda")
params <- c("mu", "alpha", "beta", "sd.alpha", "sd.res")

# MCMC settings (for normal GLM shorter chains suffice)
ni <- 2500
```

```
nt <- 2
nb <- 500
nc <- 3

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

# Summarize posteriors
print(out.b, dig = 2)
Inference for Bugs model at "GLMM_Normal.txt", fit using WinBUGS,
 3 chains, each with 2500 iterations (first 500 discarded), n.thin = 2
 n.sims = 3000 iterations saved
             mean    sd    2.5%     25%     50%     75%   97.5% Rhat n.eff
mu          11.61 10.78   -9.06    4.06   11.67   18.78   33.23    1  3000
alpha[1]    79.28  5.65   68.48   75.36   79.34   83.14   90.25    1  1400
alpha[2]    88.03  5.64   76.97   84.16   88.04   91.93   98.90    1  3000
alpha[3]    90.50  5.67   79.43   86.64   90.44   94.19  101.70    1  3000
alpha[4]   108.35  5.73   97.00  104.60  108.40  112.20  119.80    1  2300
alpha[5]   128.69  5.71  117.30  124.90  128.60  132.50  139.80    1  1700
alpha[6]    87.17  5.67   75.81   83.36   87.25   91.10   97.79    1  3000
alpha[7]    45.83  5.69   34.49   42.05   45.66   49.71   57.22    1  2700
alpha[8]    52.62  5.67   41.86   48.85   52.65   56.48   63.63    1  2400
alpha[9]    83.61  5.71   72.52   79.85   83.51   87.49   94.69    1  3000
alpha[10]   60.59  5.69   49.71   56.87   60.60   64.34   71.95    1  1300
beta[1]     91.55  5.09   81.31   88.02   91.59   95.07  101.30    1  3000
beta[2]     25.19  5.09   15.18   21.78   25.24   28.65   35.22    1  2600
beta[3]     -8.79  7.26  -23.00  -13.75   -8.93   -3.99    5.72    1  3000
sd.alpha    86.17 26.11   49.50   67.80   82.01   98.93  150.62    1   850
sd.res      34.54  1.32   32.11   33.64   34.51   35.43   37.23    1  2200
deviance  3967.84 12.13 3945.00 3959.00 3967.00 3976.00 3993.00    1  3000

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

DIC info (using the rule, pD = var(deviance)/2)
pD = 73.6 and DIC = 4041.5
DIC is an estimate of expected predictive error (lower deviance is better).
```

Chapter 5

Exercise 1

Task: Random variability in detection probability: quite often we cannot assume that detection probability is constant over time, e.g. because of weather factors that affect the counts. Simulate and analyze data for a population, whose size remains constant at 50 individuals over a) 25 years and b) 50 years, but where the annual detection probability varies randomly in the interval from 0.3 and 0.7. Does the state-space model perform well in this situation?

Solution: We simulate data with the parameters defined in exercise 1. Then we fit the same state-space model as used in the book, i.e. we do not make an adaptation of the code due to the random variation in detection probability.

Data simulation

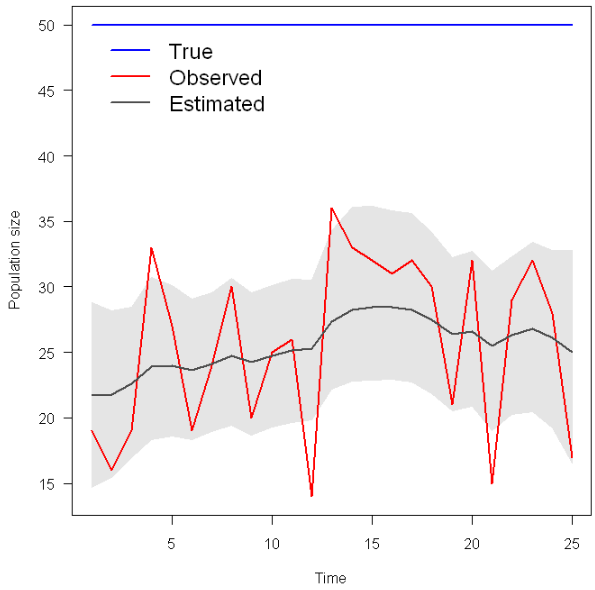
A) 25 years

```
# Simulate the development of the population
n.years <- 25                  # Number of years
N <- rep(50, n.years)

# Simulate detection probability and counts
p <- runif(n.years, 0.3, 0.7)
y <- numeric()
for (t in 1:n.years){
   y[t] <- rbinom(1, N[t],p[t])
   }
```

Data analysis

```
# Specify model in BUGS language
sink("ssm.bug")
cat("
model {
# Priors and constraints
N.est[1] ~ dunif(0, 500)          # Prior for initial population size
mean.lambda ~ dunif(0, 10)
sigma.proc ~ dunif(0, 10)
tau.proc <- pow(sigma.proc, -2)
sigma2.proc <- pow(sigma.proc, 2)
sigma.obs ~ dunif(0, 100)
tau.obs <- pow(sigma.obs, -2)
sigma2.obs <- pow(sigma.obs, 2)

# State process
for (t in 1:(T-1)){
   lambda[t] ~ dnorm(mean.lambda, tau.proc)
   N.est[t+1] <- N.est[t] * lambda[t]
   }

# Observation process
for (t in 1:T) {
   y[t] ~ dnorm(N.est[t], tau.obs)
   }
}
",fill=TRUE)
```

```
sink()

# Bundle data
bugs.data <- list(y = y, T = n.years)

# Initial values
inits <- function(){list(sigma.proc = runif(1, 0, 1), mean.lambda =
runif(1, 0.1, 2), sigma.obs = runif(1, 5, 50), N.est = c(runif(1, 20, 40),
rep(NA, (n.years-1))))}

# Parameters monitored
parameters <- c("lambda", "mean.lambda", "sigma2.obs", "sigma2.proc",
"N.est")

# MCMC settings
niter <- 25000
nthin <- 3
nburn <- 10000
nchains <- 3

# Call WinBUGS from R (BRT < 1)
ssmex <- bugs(bugs.data, inits, parameters, "ssm.bug", n.chains = nchains,
n.thin = nthin, n.iter = niter, n.burnin = nburn, debug = TRUE,
bugs.directory = bugs.dir)

# Define function for visualisation of results
graph.ssm <- function(ssm, N, y){
   fitted <- lower <- upper <- numeric()
   n.years <- length(y)
   for (i in 1:n.years){
      fitted[i] <- mean(ssm$sims.list$N.est[,i])
      lower[i] <- quantile(ssm$sims.list$N.est[,i], 0.025)
      upper[i] <- quantile(ssm$sims.list$N.est[,i], 0.975)}
   m1 <- min(c(y, fitted, N, lower))
   m2 <- max(c(y, fitted, N, upper))
   par(mar = c(4.5, 4, 1, 1))
   plot(0, 0, ylim = c(m1, m2), xlim = c(1, n.years), ylab = "Population
size", xlab = "Time", las = 1, col = "black", type = "l", lwd = 2)
   polygon(x=c(1:n.years,n.years:1), y = c(lower, upper[n.years:1]), col =
"grey90", border = "grey90")
   points(N, type = "l", col = "blue", lwd = 2)
   points(y, type = "l", col = "red", lwd = 2)
   points(fitted, type = "l", col = "grey30", lwd = 2)
   legend(x = 1, y = m2, legend = c("True", "Observed", "Estimated"), lty =
c(1, 1, 1),lwd = c(2, 2, 2), col = c("blue", "red", "grey30"), bty = "n",
cex = 1.5)
}

# Apply graph function
graph.ssm(ssmex, N, y)
```



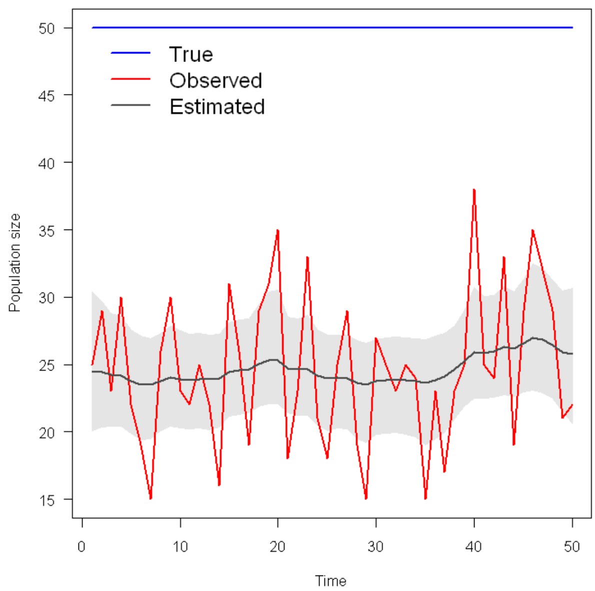
In this situation, the state-space model is useful to get a relatively smoothed population index. This shows that the model is not only able to account for the binomial sampling variation but also for a random change in the average detection. The longer the data series, the better is the smoothing. This can be seen in the graph below, which uses the same settings as above, but for a data set comprising 50 years.

B) 50 years

The only change required in the previous code is the following line:

```
n.years <- 50                    # Number of years
```

The resulting graph shows that the smoothing is better than with the shorter time series.



Exercise 2

Task: Modeling of variance structures: in the house martin data set we saw that from year *t* = 9 onwards, a different data collection protocol (questionnaires) was used. Adapt the model to account for possibly different observation errors in the two periods.

Solution: There are at least two different ways how we can model a different observation error in the state-space model. First we can use two loops for the observation process, the first extending from year 1 to 8, and the second from year 9 onwards. In these two loops we use different precision measures (tau.obs). An alternative, but more elegant way to specify the same model is via a GLM. Instead of modelling the observation process for the two periods separately, we consider the categorical covariate period, indicating by a 1 the first and by a 2 the second observation period. We then index the precision of the observation (tau.obs) with period. An advantage of this formulation is the greater flexibility, as we could easily model different observation errors of a collection of years that are not in a row, and it allows the inclusion of more periods in a handy way. We show both solutions below.

Read in data set

```
# Load data: House martin population from Magden
hm <- c(271, 261, 309, 318, 231, 216, 208, 226, 195, 226, 233, 209, 226,
192, 191, 225, 245, 205, 191, 174)
year <- 1990:2009
```

Data analysis

```
# Specify model in BUGS language
sink("ssm.bug")
cat("
model {
# Priors and constraints
logN.est[1] ~ dnorm(5.6, 0.01)      # Prior for initial population size
mean.r ~ dnorm(1, 0.001)            # Prior for mean growth rate
sigma.proc ~ dunif(0, 1)            # Prior for sd of state process
sigma2.proc <- pow(sigma.proc, 2)
tau.proc <- pow(sigma.proc, -2)
sigma.obs1 ~ dunif(0, 1)            # Prior for sd of obs. process period 1
sigma2.obs1 <- pow(sigma.obs1, 2)
tau.obs1 <- pow(sigma.obs1, -2)
sigma.obs2 ~ dunif(0, 1)            # Prior for sd of obs. process period 2
sigma2.obs2 <- pow(sigma.obs2, 2)
tau.obs2 <- pow(sigma.obs2, -2)

# State process
for (t in 1:(T-1)){
   r[t] ~ dnorm(mean.r, tau.proc)
   logN.est[t+1] <- logN.est[t] + r[t]
   }

# Observation process: the observation error changes
for (t in 1:8) {
   y[t] ~ dnorm(logN.est[t], tau.obs1)
   }
for (t in 9:T) {
   y[t] ~ dnorm(logN.est[t], tau.obs2)
   }

# Population sizes on real scale
for (t in 1:T) {
```

```
    N.est[t] <- exp(logN.est[t])
    }
}
",fill=TRUE)
sink()

# Bundle data
bugs.data <- list(y = log(hm), T = length(year))

# Initial values
inits <- function(){list(sigma.proc = runif(1, 0, 1), mean.r = rnorm(1),
sigma.obs1 = runif(1, 0, 1), sigma.obs2 = runif(1, 0, 1), logN.est =
c(rnorm(1, 5.6, 0.1), rep(NA, (length(year)-1))))}

# Parameters monitored
parameters <- c("r", "mean.r", "sigma2.obs1", "sigma2.obs2", "sigma2.proc",
"N.est")

# MCMC settings
niter <- 50000
nthin <- 3
nburn <- 25000
nchains <- 3

# Call WinBUGS from R (BRT < 1)
hm.562 <- bugs(bugs.data, inits, parameters, "ssm.bug", n.chains = nchains,
n.thin = nthin, n.iter = niter, n.burnin = nburn, debug = TRUE,
bugs.directory = bugs.dir, working.directory = getwd())
```

Rather long chains are required to get convergence. It appears as if the observation error is actually larger in the first than in the second part of the time series.

```
print(hm.562, digits = 3)
               mean     sd    2.5%     25%     50%     75%   97.5%  Rhat n.eff
r[1]         -0.015  0.071  -0.162  -0.052  -0.018   0.025   0.137 1.006  7100
r[2]          0.060  0.089  -0.104  -0.009   0.055   0.134   0.211 1.005   560
r[3]         -0.016  0.069  -0.157  -0.054  -0.014   0.027   0.117 1.007   560
r[4]         -0.156  0.109  -0.342  -0.246  -0.149  -0.064   0.025 1.002  1900
r[5]         -0.073  0.066  -0.215  -0.112  -0.069  -0.029   0.048 1.005   460
r[6]         -0.038  0.063  -0.173  -0.074  -0.035  -0.001   0.086 1.003  1200
r[7]          0.012  0.074  -0.142  -0.034   0.011   0.065   0.152 1.007   340
r[8]         -0.060  0.072  -0.203  -0.109  -0.056  -0.012   0.078 1.002  1400
r[9]          0.074  0.075  -0.072   0.013   0.082   0.136   0.196 1.002  2600
r[10]         0.017  0.055  -0.097  -0.016   0.020   0.047   0.130 1.001 11000
r[11]        -0.058  0.064  -0.173  -0.105  -0.062  -0.015   0.074 1.002  2700
r[12]         0.023  0.062  -0.107  -0.018   0.027   0.071   0.134 1.002  2500
r[13]        -0.091  0.073  -0.211  -0.152  -0.097  -0.033   0.048 1.002  2400
r[14]        -0.002  0.053  -0.112  -0.030  -0.005   0.025   0.114 1.002 25000
r[15]         0.093  0.078  -0.053   0.031   0.105   0.157   0.219 1.002  3200
r[16]         0.043  0.062  -0.082  -0.002   0.050   0.086   0.159 1.001  7900
r[17]        -0.106  0.074  -0.228  -0.168  -0.113  -0.047   0.037 1.002  2900
r[18]        -0.067  0.057  -0.186  -0.098  -0.068  -0.031   0.044 1.001 11000
r[19]        -0.069  0.060  -0.188  -0.103  -0.074  -0.030   0.055 1.002  3600
mean.r       -0.023  0.025  -0.075  -0.037  -0.022  -0.008   0.028 1.001 25000
sigma2.obs1   0.015  0.022   0.000   0.003   0.009   0.019   0.066 1.010  2300
sigma2.obs2   0.006  0.007   0.000   0.001   0.004   0.009   0.025 1.030   160
sigma2.proc   0.012  0.008   0.000   0.005   0.010   0.016   0.032 1.012  1400
N.est[1]    275.791 23.131  232.800 263.300 273.100 286.500 328.800 1.009  4000
N.est[2]    271.506 19.990  235.900 259.900 268.800 282.100 317.100 1.007  1200
N.est[3]    288.533 24.831  238.600 272.000 290.600 306.500 330.000 1.003  1100
N.est[4]    284.296 27.048  233.100 264.100 284.900 306.700 328.800 1.001 25000
N.est[5]    242.779 16.507  213.400 231.700 240.700 252.100 279.600 1.003   990
N.est[6]    225.681 16.331  197.300 215.000 223.500 235.500 262.200 1.003   970
N.est[7]    217.307 15.697  189.400 207.100 214.900 227.400 251.100 1.003   940
```

```
N.est[8]    219.729 14.201  189.400 211.100 220.700 228.200 246.800 1.004  1500
N.est[9]    206.964 13.800  187.100 196.100 203.800 215.800 237.600 1.002  2500
N.est[10]   222.550 10.750  199.700 216.500 223.600 228.300 243.900 1.001 12000
N.est[11]   226.388 11.679  201.500 219.300 227.900 233.600 248.000 1.002  3200
N.est[12]   213.647 10.495  194.300 207.700 212.200 219.300 237.200 1.001 10000
N.est[13]   218.728 11.187  194.800 211.800 220.000 226.000 239.300 1.002  3600
N.est[14]   199.699 11.418  181.200 191.900 197.400 206.700 225.600 1.001  4800
N.est[15]   199.196 11.228  181.100 191.200 197.100 206.000 224.500 1.001  3800
N.est[16]   218.695 12.106  193.200 211.200 220.600 226.100 241.000 1.001 14000
N.est[17]   228.703 17.249  193.000 216.100 231.900 243.000 254.300 1.001  4400
N.est[18]   205.359 10.353  184.500 199.800 205.200 210.800 227.400 1.001 25000
N.est[19]   192.092  9.879  173.000 186.800 191.400 196.900 213.900 1.001 21000
N.est[20]   179.477 11.720  159.800 172.600 177.000 185.400 206.900 1.001  3900
deviance    -57.659 26.685 -124.700 -71.698 -51.340 -37.420 -23.950 1.016   270
```

Here is the second solution with the GLM formulation:

```
# Specify model in BUGS language
sink("ssm.bug")
cat("
model {
# Priors and constraints
logN.est[1] ~ dnorm(5.6, 0.01)      # Prior for initial population size
mean.r ~ dnorm(1, 0.001)            # Prior for mean growth rate
sigma.proc ~ dunif(0, 1)            # Prior for sd of state process
sigma2.proc <- pow(sigma.proc, 2)
tau.proc <- pow(sigma.proc, -2)
for (i in 1:2){
   sigma.obs[i] ~ dunif(0, 100)     # Priors for sd of obs proccesses
   tau.obs[i] <- pow(sigma.obs[i], -2)
   sigma2.obs[i] <- pow(sigma.obs[i], 2)
   }

# State process
for (t in 1:(T-1)){
   r[t] ~ dnorm(mean.r, tau.proc)
   logN.est[t+1] <- logN.est[t] + r[t]
   }

# Observation process: the observation error changes
for (t in 1:T) {
   y[t] ~ dnorm(logN.est[t], tau.obs[period[t]])
   }

# Population sizes on real scale
for (t in 1:T) {
   N.est[t] <- exp(logN.est[t])
   }
}
",fill=TRUE)
sink()


# Bundle data
bugs.data <- list(y = log(hm), T = length(year), period = c(rep(1, 8),
rep(2, 12)))

# Initial values
inits <- function(){list(sigma.proc = runif(1, 0, 1), mean.r = rnorm(1),
sigma.obs = runif(2, 0, 1), logN.est = c(rnorm(1, 5.6, 0.1), rep(NA,
(length(year)-1))))}
```

```
# Parameters monitored
parameters <- c("r", "mean.r", "sigma2.obs", "sigma2.proc", "N.est")

# MCMC settings
niter <- 50000
nthin <- 3
nburn <- 25000
nchains <- 3

# Call WinBUGS from R (BRT 2 min)
hm.562alt <- bugs(bugs.data, inits, parameters, "ssm.bug", n.chains =
nchains, n.thin = nthin, n.iter = niter, n.burnin = nburn, debug = TRUE,
bugs.directory = bugs.dir, working.directory = getwd())

print(hm.562alt, digits = 3)
                  mean     sd    2.5%     25%     50%     75%   97.5%  Rhat n.eff
r[1]            -0.013  0.067  -0.147  -0.048  -0.020   0.021   0.139 1.011   740
r[2]             0.060  0.093  -0.121  -0.011   0.058   0.141   0.213 1.012   190
r[3]            -0.008  0.064  -0.147  -0.045  -0.006   0.030   0.119 1.004  1800
r[4]            -0.165  0.111  -0.339  -0.264  -0.159  -0.069   0.015 1.007   310
r[5]            -0.070  0.067  -0.222  -0.106  -0.067  -0.029   0.062 1.002  5100
r[6]            -0.037  0.064  -0.179  -0.068  -0.036  -0.006   0.095 1.013  7500
r[7]             0.015  0.074  -0.143  -0.032   0.016   0.072   0.150 1.018   130
r[8]            -0.065  0.073  -0.207  -0.118  -0.059  -0.014   0.072 1.014   180
r[9]             0.072  0.075  -0.069   0.011   0.078   0.136   0.196 1.001  5500
r[10]            0.018  0.056  -0.096  -0.015   0.021   0.047   0.135 1.001 25000
r[11]           -0.057  0.062  -0.168  -0.104  -0.060  -0.016   0.071 1.002  3100
r[12]            0.022  0.062  -0.104  -0.019   0.025   0.070   0.131 1.002  1500
r[13]           -0.089  0.073  -0.211  -0.152  -0.094  -0.031   0.051 1.002  2400
r[14]           -0.003  0.054  -0.114  -0.030  -0.005   0.024   0.113 1.002 13000
r[15]            0.092  0.079  -0.051   0.027   0.102   0.157   0.220 1.001 18000
r[16]            0.043  0.062  -0.080  -0.003   0.049   0.086   0.159 1.001 13000
r[17]           -0.105  0.074  -0.227  -0.168  -0.111  -0.044   0.033 1.002  2700
r[18]           -0.066  0.057  -0.186  -0.096  -0.067  -0.030   0.044 1.001  3900
r[19]           -0.068  0.060  -0.188  -0.102  -0.073  -0.028   0.056 1.001 25000
mean.r          -0.023  0.025  -0.076  -0.037  -0.022  -0.008   0.029 1.001 25000
sigma2.obs[1]    0.014  0.022   0.000   0.003   0.009   0.018   0.060 1.017   270
sigma2.obs[2]    0.006  0.007   0.000   0.001   0.004   0.009   0.025 1.037   180
sigma2.proc      0.012  0.009   0.000   0.005   0.010   0.016   0.033 1.003  5300
N.est[1]       275.041 20.900  235.002 263.900 272.400 285.200 322.797 1.007  1500
N.est[2]       271.378 20.004  237.000 259.900 268.000 281.200 319.700 1.013   270
N.est[3]       288.523 23.446  240.302 272.100 291.100 307.200 327.700 1.003  1000
N.est[4]       286.422 26.661  234.800 265.600 288.100 309.800 327.400 1.005   560
N.est[5]       242.341 15.889  215.600 231.300 240.000 251.400 278.600 1.003   820
N.est[6]       226.021 15.827  199.400 215.400 223.000 236.300 260.700 1.004   870
N.est[7]       217.729 15.545  190.300 207.500 214.800 227.700 251.900 1.002  1400
N.est[8]       220.949 13.556  192.600 212.800 222.300 228.400 247.800 1.018   140
N.est[9]       207.144 13.956  187.000 196.000 204.200 216.300 238.000 1.001 17000
N.est[10]      222.339 10.717  199.400 216.400 223.500 228.000 243.700 1.001  6300
N.est[11]      226.387 11.608  201.900 219.000 227.800 233.500 248.300 1.001 13000
N.est[12]      213.743 10.445  194.500 207.900 212.200 219.400 237.397 1.001  5300
N.est[13]      218.615 11.265  194.702 211.600 219.900 226.000 239.297 1.002  3200
N.est[14]      199.961 11.503  181.700 192.000 197.700 206.900 225.600 1.001  8800
N.est[15]      199.357 11.375  181.100 191.200 197.300 206.500 224.400 1.001 11000
N.est[16]      218.499 12.177  192.900 210.800 220.400 225.800 240.800 1.001 25000
N.est[17]      228.349 17.381  192.500 215.400 231.550 243.000 253.900 1.001 25000
N.est[18]      205.209 10.469  183.700 199.700 205.100 210.500 227.600 1.002  2800
N.est[19]      192.165  9.827  172.900 187.000 191.300 197.000 214.500 1.001 25000
N.est[20]      179.664 11.684  160.600 172.800 177.000 185.700 207.300 1.001 25000
deviance       -60.388 31.944 -150.998 -74.990 -51.245 -37.180 -24.210 1.033   190
```

The results are identical (up to MCMC error) to those under the first model specification.

Exercise 3

Task: Unstructured and dynamic hierarchical model for population counts: In section 4.2.2 we encountered a different two-level hierarchical model for a single time-series of population counts. What is the difference to a state-space model in this chapter? Fit the exponential population state-space model to the peregrine data from Section 4.2.2 and compare the inference about the population trajectory under the two models. In addition, construct a model with a linear trend in the population growth rate and another one with a linear trend in the observation error and fit them to the peregrine data.

Solution: There are two differences between the two kinds of hierarchical models. First, the equivalent of the state equation in the model in Chapter 4 is simply a reasonably smooth regression (namely a cubic polynomial), which lacks any biological justification. This is a purely phenomenological description of the underlying "true" population trajectory. In contrast, the state-space models in Chapter 5 contain what is perhaps the simplest kind of population model, namely that for exponential growth. This model accommodates the fact that the number of individuals in year *t*+1 must be related in some way to the number of individuals in year *t*. In this way, the state-space model also accounts for autocorrelation in the counts, while the cubic polynomial model in Chapter 4 does not. The state-space model is also more flexible than the cubic polynomial model, because it allows the population growth rate to be different for each year. Second, both models contain an extra component of variation (which, incidentally, is assumed to be normal for both). However, in the model in Chapter 4 this is called overdispersion (or as "unexplained year effects"), while in the state-space model in Chapter 5, it is called the observation error.

Read in the data

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

Data analysis

```
# Specify model in BUGS language
sink("ssm.bug")
cat("
model {
# Priors and constraints
N.est[1] ~ dunif(0, 200)            # Prior for initial population size
mean.lambda ~ dunif(0, 2)           # Prior for mean growth rate
sigma.proc ~ dunif(0, 5)            # Prior sd of state process
sigma2.proc <- pow(sigma.proc, 2)
tau.proc <- pow(sigma.proc, -2)
sigma.obs ~ dunif(0, 20)           # Prior sd of observation process
sigma2.obs <- pow(sigma.obs, 2)
tau.obs <- pow(sigma.obs, -2)

# State process
for (t in 1:(T-1)){
   lambda[t] ~ dnorm(mean.lambda, tau.proc)
   N.est[t+1] <- N.est[t] * lambda[t]
   }

# Observation process
for (t in 1:T) {
   y[t] ~ dnorm(N.est[t], tau.obs)
   }
```

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


# Bundle data
bugs.data <- list(y = peregrine$Pairs, T = length(peregrine$Pairs))

# Initial values
inits <- function(){list(sigma.proc = runif(1, 0, 1), mean.lambda =
runif(1, 0.9, 1.1), sigma.obs = runif(1, 0.5, 5), N.est = c(runif(1, 20,
40), rep(NA, (length(peregrine$Pairs)-1))))}

# Parameters monitored
parameters <- c("lambda", "mean.lambda", "sigma2.obs", "sigma2.proc",
"N.est")

# MCMC settings
niter <- 100000
nthin <- 10
nburn <- 50000
nchains <- 3

# Call WinBUGS from R (BRT 8 min)
per1 <- bugs(bugs.data, inits, parameters, "ssm.bug", n.chains = nchains,
n.thin = nthin, n.iter = niter, n.burnin = nburn, debug = TRUE,
bugs.directory = bugs.dir)
```

When running this model, it may sometimes happen that you get the error message “undefined real result”. This can occur because by chance some awkward initial values have been generated. In this case, just shut down WinBUGS and run the bugs command again. The model is quite difficult to get to convergence. It is advisable to make the prior distributions for the unknown parameters not too wide.

```
print(per1, 3)
Inference for Bugs model at "ssm.bug", fit using WinBUGS,
 3 chains, each with 1e+05 iterations (first 50000 discarded), n.thin = 10
 n.sims = 15000 iterations saved
              mean    sd  2.5%   25%   50%   75% 97.5%  Rhat n.eff
lambda[1]    1.238 0.087 1.034 1.187 1.256 1.302 1.371 1.045    64
lambda[2]    0.898 0.055 0.813 0.863 0.886 0.924 1.039 1.031   110
lambda[3]    0.910 0.058 0.785 0.878 0.914 0.945 1.016 1.011   200
lambda[4]    0.593 0.077 0.489 0.542 0.574 0.627 0.794 1.031   150
lambda[5]    0.926 0.078 0.762 0.883 0.923 0.976 1.086 1.006  1600
lambda[6]    1.070 0.091 0.869 1.014 1.082 1.131 1.229 1.013   260
lambda[7]    0.966 0.075 0.827 0.916 0.965 1.011 1.129 1.003  1000
lambda[8]    0.949 0.087 0.818 0.882 0.936 1.004 1.143 1.024    92
lambda[9]    1.096 0.085 0.918 1.042 1.102 1.154 1.256 1.018   130
lambda[10]   1.142 0.080 0.975 1.094 1.140 1.190 1.313 1.004   610
lambda[11]   1.167 0.075 1.012 1.121 1.166 1.214 1.321 1.007   570
lambda[12]   1.108 0.068 0.994 1.063 1.101 1.145 1.269 1.003   870
lambda[13]   1.283 0.077 1.115 1.234 1.294 1.340 1.405 1.019   120
lambda[14]   1.123 0.059 1.018 1.085 1.115 1.156 1.257 1.016   140
lambda[15]   1.107 0.050 1.013 1.077 1.103 1.132 1.219 1.006   490
lambda[16]   1.192 0.052 1.078 1.163 1.199 1.223 1.287 1.010   350
lambda[17]   1.002 0.045 0.926 0.975 0.995 1.023 1.109 1.020   150
lambda[18]   1.118 0.043 1.031 1.094 1.120 1.142 1.209 1.007 15000
lambda[19]   1.114 0.039 1.036 1.093 1.113 1.135 1.197 1.004  1100
lambda[20]   1.166 0.039 1.076 1.147 1.168 1.186 1.242 1.010   360
lambda[21]   1.101 0.033 1.037 1.084 1.099 1.116 1.176 1.012   820
lambda[22]   1.058 0.029 0.999 1.043 1.057 1.073 1.124 1.012   620
lambda[23]   1.124 0.029 1.061 1.110 1.125 1.138 1.183 1.011   480
```

```
lambda[24]     1.067  0.025   1.016   1.054   1.066   1.079   1.121 1.008  1800
lambda[25]     1.062  0.024   1.013   1.050   1.061   1.073   1.116 1.009   890
lambda[26]     1.090  0.023   1.041   1.079   1.090   1.101   1.136 1.005  2200
lambda[27]     1.017  0.021   0.976   1.007   1.016   1.027   1.063 1.006  8400
lambda[28]     1.067  0.021   1.023   1.056   1.067   1.077   1.111 1.014  8500
lambda[29]     1.049  0.019   1.008   1.039   1.049   1.058   1.089 1.012  3900
lambda[30]     1.028  0.018   0.992   1.019   1.027   1.037   1.067 1.007  2500
lambda[31]     1.114  0.018   1.073   1.106   1.115   1.124   1.150 1.010  1200
lambda[32]     0.950  0.016   0.919   0.942   0.949   0.957   0.987 1.015  1600
lambda[33]     1.006  0.017   0.970   0.998   1.006   1.014   1.040 1.011  2800
lambda[34]     0.981  0.016   0.946   0.973   0.981   0.989   1.015 1.011 15000
lambda[35]     0.855  0.016   0.826   0.846   0.853   0.862   0.894 1.017   790
lambda[36]     1.141  0.022   1.091   1.131   1.143   1.153   1.181 1.013   920
lambda[37]     0.979  0.018   0.947   0.970   0.977   0.986   1.023 1.020   900
lambda[38]     1.243  0.021   1.192   1.234   1.245   1.254   1.279 1.023   440
lambda[39]     1.012  0.014   0.985   1.005   1.011   1.019   1.045 1.012  1700
mean.lambda    1.054  0.023   1.010   1.040   1.054   1.069   1.099 1.001 15000
sigma2.obs     3.877  4.781   0.222   0.970   2.359   4.915  16.760 1.027   550
sigma2.proc    0.019  0.005   0.011   0.016   0.019   0.022   0.032 1.010   220
N.est[1]      35.088  1.749  32.200  34.040  34.840  35.920  39.250 1.032   180
N.est[2]      43.340  2.273  37.780  42.260  43.910  44.850  46.380 1.051    99
N.est[3]      38.823  1.783  34.830  38.000  38.930  39.750  42.300 1.013  2100
N.est[4]      35.270  2.006  30.570  34.320  35.600  36.510  38.620 1.027   130
N.est[5]      20.824  2.063  17.870  19.490  20.330  21.750  26.190 1.020   340
N.est[6]      19.191  1.565  16.810  18.150  18.870  19.970  23.100 1.022   340
N.est[7]      20.439  1.442  17.070  19.690  20.490  21.280  23.220 1.009   260
N.est[8]      19.693  1.376  17.000  18.860  19.730  20.480  22.620 1.021   120
N.est[9]      18.629  1.553  16.400  17.490  18.350  19.520  22.370 1.025   120
N.est[10]     20.342  1.405  17.960  19.450  20.170  21.050  23.690 1.014   290
N.est[11]     23.152  1.322  20.730  22.330  23.070  23.860  26.080 1.010  2500
N.est[12]     26.947  1.378  24.190  26.170  26.940  27.690  29.850 1.007   370
N.est[13]     29.805  1.425  27.440  28.850  29.590  30.580  33.160 1.017   170
N.est[14]     38.163  1.657  34.300  37.267  38.420  39.210  40.970 1.016   170
N.est[15]     42.769  1.596  39.280  41.960  42.845  43.650  45.920 1.009   580
N.est[16]     47.277  1.582  44.070  46.390  47.230  48.150  50.600 1.007  5300
N.est[17]     56.313  1.809  52.100  55.440  56.535  57.390  59.510 1.019   190
N.est[18]     56.355  1.727  52.940  55.440  56.240  57.210  60.220 1.010   640
N.est[19]     62.974  1.752  59.360  62.050  62.980  63.870  66.680 1.010   660
N.est[20]     70.123  1.753  66.590  69.210  70.050  71.010  73.930 1.003  2300
N.est[21]     81.686  1.887  77.320  80.850  81.845  82.690  85.290 1.018   290
N.est[22]     89.890  1.852  85.860  88.980  89.920  90.840  93.640 1.011  3400
N.est[23]     95.104  1.853  91.200  94.180  95.050  96.000  99.160 1.012   650
N.est[24]    106.831  1.875 102.800 105.900 106.900 107.800 110.600 1.011  1500
N.est[25]    113.960  1.903 109.800 113.100 114.000 114.900 117.900 1.012   890
N.est[26]    121.003  1.915 117.000 120.100 121.000 121.900 125.100 1.010  6700
N.est[27]    131.830  1.930 127.600 130.900 131.900 132.800 135.700 1.009  2000
N.est[28]    134.059  1.934 130.000 133.100 134.100 135.000 138.100 1.012 15000
N.est[29]    142.967  1.959 138.800 142.000 143.000 143.900 147.100 1.016 15000
N.est[30]    149.938  1.920 145.800 149.000 150.000 150.900 153.900 1.010  4700
N.est[31]    154.114  1.892 150.300 153.200 154.100 155.000 158.102 1.009  5800
N.est[32]    171.711  1.970 167.297 170.800 171.800 172.700 175.600 1.016  1300
N.est[33]    163.066  1.968 159.000 162.100 163.000 164.000 167.200 1.014 12000
N.est[34]    163.946  1.932 159.800 163.000 164.000 164.900 167.900 1.010  3100
N.est[35]    160.820  1.898 156.700 159.900 160.900 161.800 164.500 1.015  2500
N.est[36]    137.423  2.001 133.800 136.400 137.200 138.300 142.100 1.014  1100
N.est[37]    156.741  1.961 152.400 155.800 156.900 157.700 160.600 1.013  3400
N.est[38]    153.440  2.000 149.900 152.400 153.200 154.300 158.102 1.019   990
N.est[39]    190.664  1.971 186.100 189.800 190.800 191.700 194.400 1.018   670
N.est[40]    193.019  1.955 189.000 192.100 193.000 194.000 197.100 1.009 15000
deviance     143.608 44.235  53.860 111.300 147.200 175.900 222.805 1.049  1600
```

An alternative way to fit this model is to transform the response to the log-scale. This has the advantage that convergence is obtained more easily (this is perhaps specific to this data set and may not be a general feature) and that the model is directly comparable to stochastic growth models which are usually written on the log-scale (e.g. Lande et al. 2003).

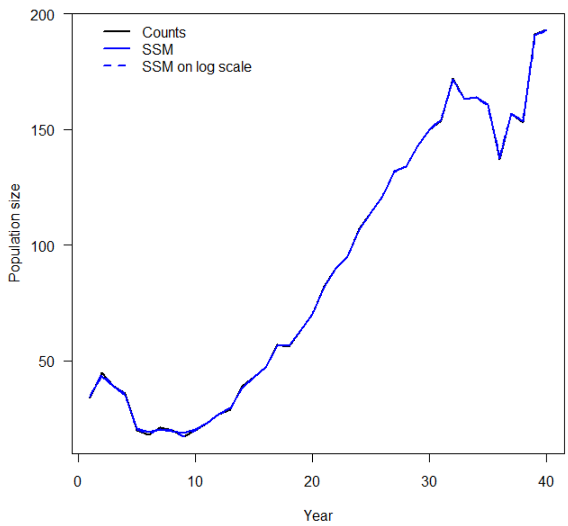
```
# Specify model in BUGS language
sink("ssm.bug")
cat("
model {
# Priors and constraints
logN[1] ~ dnorm(0, 0.01)            # Prior for initial population size
mean.r ~ dnorm(0, 0.01)             # Prior for mean stochastic growth rate
sigma.proc ~ dunif(0, 2)            # Prior sd of state process
sigma2.proc <- pow(sigma.proc, 2)
tau.proc <- pow(sigma.proc, -2)
sigma.obs ~ dunif(0, 10)            # Prior sd of observation process
sigma2.obs <- pow(sigma.obs, 2)
tau.obs <- pow(sigma.obs, -2)

# State process
for (t in 1:(T-1)){
   r[t] ~ dnorm(mean.r, tau.proc)
   logN[t+1] <- logN[t] + r[t]
   }

# Observation process
for (t in 1:T) {
   y[t] ~ dnorm(logN[t], tau.obs)
   N.est[t] <- exp(logN[t])        # Backtransformation from log-scale
   }
}
",fill = TRUE)
sink()


# Bundle data
bugs.data <- list(y = log(peregrine$Pairs), T = length(peregrine$Pairs))

# Initial values
inits <- function(){list(sigma.proc = runif(1, 0, 1), mean.r = rnorm(1),
sigma.obs = runif(1, 0.5, 5), logN = c(runif(1, -1, 5), rep(NA,
(length(peregrine$Pairs)-1))))}

# Parameters monitored
parameters <- c("r", "mean.r", "sigma2.obs", "sigma2.proc", "N.est")

# MCMC settings
niter <- 100000
nthin <- 10
nburn <- 50000
nchains <- 3

# Call WinBUGS from R (BRT 8 min)
per2 <- bugs(bugs.data, inits, parameters, "ssm.bug", n.chains = nchains,
n.thin = nthin, n.iter = niter, n.burnin = nburn, debug = TRUE,
bugs.directory = bugs.dir)

print(per2, 3)
Inference for Bugs model at "ssm.bug", fit using WinBUGS,
 3 chains, each with 1e+05 iterations (first 50000 discarded), n.thin = 10
 n.sims = 15000 iterations saved
                mean     sd     2.5%      25%      50%      75%    97.5%  Rhat n.eff
r[1]           0.255  0.051    0.120    0.237    0.270    0.284    0.328 1.003  1100
r[2]          -0.128  0.043   -0.198   -0.150   -0.137   -0.111   -0.020 1.001 10000
r[3]          -0.097  0.043   -0.207   -0.113   -0.088   -0.074   -0.027 1.004 10000
r[4]          -0.552  0.058   -0.622   -0.588   -0.571   -0.531   -0.391 1.011   440
r[5]          -0.112  0.040   -0.206   -0.129   -0.108   -0.094   -0.032 1.004  4600
```

r[6]	0.137	0.045	0.021	0.120	0.146	0.160	0.208	1.010	440
r[7]	-0.046	0.040	-0.131	-0.063	-0.048	-0.030	0.040	1.004	1400
r[8]	-0.145	0.044	-0.216	-0.168	-0.155	-0.129	-0.029	1.006	640
r[9]	0.150	0.041	0.048	0.134	0.157	0.170	0.223	1.005	670
r[10]	0.140	0.040	0.050	0.124	0.140	0.157	0.225	1.003	2400
r[11]	0.157	0.040	0.064	0.141	0.159	0.174	0.242	1.004	2300
r[12]	0.083	0.041	0.007	0.063	0.076	0.098	0.183	1.007	330
r[13]	0.281	0.043	0.173	0.264	0.289	0.302	0.357	1.008	360
r[14]	0.103	0.040	0.021	0.086	0.100	0.120	0.195	1.003	1900
r[15]	0.094	0.041	0.011	0.077	0.091	0.110	0.188	1.002	5800
r[16]	0.182	0.042	0.080	0.166	0.188	0.201	0.258	1.003	890
r[17]	-0.005	0.042	-0.080	-0.026	-0.012	0.012	0.097	1.004	2000
r[18]	0.112	0.039	0.020	0.096	0.115	0.129	0.190	1.004	15000
r[19]	0.108	0.039	0.024	0.091	0.107	0.124	0.195	1.004	5800
r[20]	0.154	0.040	0.061	0.139	0.157	0.172	0.234	1.002	11000
r[21]	0.093	0.039	0.006	0.077	0.093	0.110	0.177	1.002	15000
r[22]	0.057	0.040	-0.022	0.039	0.055	0.073	0.151	1.004	15000
r[23]	0.116	0.039	0.026	0.100	0.118	0.133	0.195	1.003	3100
r[24]	0.065	0.039	-0.019	0.049	0.064	0.081	0.152	1.003	3600
r[25]	0.060	0.039	-0.026	0.044	0.060	0.077	0.146	1.003	15000
r[26]	0.084	0.038	-0.004	0.068	0.086	0.100	0.165	1.003	4500
r[27]	0.020	0.039	-0.059	0.002	0.017	0.035	0.113	1.004	960
r[28]	0.062	0.040	-0.028	0.046	0.064	0.080	0.144	1.002	3900
r[29]	0.048	0.039	-0.037	0.031	0.048	0.065	0.133	1.002	15000
r[30]	0.030	0.039	-0.051	0.012	0.027	0.046	0.122	1.001	15000
r[31]	0.101	0.041	0.002	0.086	0.107	0.121	0.177	1.001	4600
r[32]	-0.046	0.041	-0.124	-0.065	-0.051	-0.030	0.051	1.003	1400
r[33]	0.003	0.039	-0.087	-0.013	0.005	0.021	0.084	1.005	820
r[34]	-0.021	0.040	-0.111	-0.037	-0.020	-0.004	0.063	1.003	2900
r[35]	-0.146	0.043	-0.218	-0.168	-0.155	-0.130	-0.036	1.003	1400
r[36]	0.121	0.043	0.010	0.105	0.130	0.143	0.193	1.003	1500
r[37]	-0.013	0.041	-0.086	-0.033	-0.020	0.003	0.090	1.003	1700
r[38]	0.206	0.044	0.095	0.190	0.216	0.229	0.277	1.004	1100
r[39]	0.018	0.041	-0.061	0.000	0.013	0.034	0.120	1.002	9500
mean.r	0.044	0.025	-0.004	0.028	0.044	0.061	0.092	1.001	6700
sigma2.obs	0.001	0.002	0.000	0.000	0.000	0.001	0.005	1.012	180
sigma2.proc	0.024	0.006	0.014	0.019	0.023	0.027	0.038	1.002	2300
N.est[1]	34.313	1.153	32.280	33.780	34.110	34.690	37.230	1.003	2400
N.est[2]	44.302	1.533	40.160	43.750	44.710	45.140	46.550	1.002	1600
N.est[3]	38.986	1.160	36.420	38.530	39.000	39.460	41.460	1.003	2200
N.est[4]	35.386	1.239	32.000	34.920	35.720	36.080	37.200	1.006	1300
N.est[5]	20.375	0.761	19.380	19.960	20.160	20.610	22.430	1.011	410
N.est[6]	18.207	0.617	17.250	17.900	18.070	18.400	19.850	1.009	750
N.est[7]	20.871	0.617	19.420	20.620	20.940	21.150	22.070	1.004	1000
N.est[8]	19.923	0.596	18.530	19.690	19.970	20.180	21.120	1.003	2600
N.est[9]	17.231	0.594	16.370	16.930	17.090	17.410	18.800	1.006	580
N.est[10]	20.022	0.583	18.830	19.770	20.000	20.240	21.380	1.002	3300
N.est[11]	23.023	0.679	21.550	22.750	23.010	23.280	24.530	1.006	7800
N.est[12]	26.934	0.792	25.140	26.610	26.970	27.270	28.610	1.004	1300
N.est[13]	29.252	0.907	27.690	28.820	29.100	29.560	31.580	1.008	470
N.est[14]	38.739	1.162	36.030	38.260	38.880	39.260	41.010	1.004	1200
N.est[15]	42.951	1.288	40.060	42.450	42.990	43.470	45.650	1.005	15000
N.est[16]	47.179	1.416	44.410	46.570	47.060	47.710	50.420	1.003	2400
N.est[17]	56.595	1.717	52.520	55.930	56.830	57.370	59.840	1.005	1500
N.est[18]	56.317	1.683	53.150	55.560	56.120	56.940	60.430	1.002	15000
N.est[19]	62.982	1.821	59.110	62.210	62.960	63.680	66.890	1.003	15000
N.est[20]	70.167	2.081	65.870	69.250	70.070	70.970	75.000	1.003	8600
N.est[21]	81.885	2.394	76.649	80.910	81.940	82.880	86.980	1.004	8800
N.est[22]	89.882	2.663	83.870	88.810	89.960	90.962	95.520	1.004	15000
N.est[23]	95.176	2.799	89.460	93.970	95.030	96.250	101.500	1.005	8900
N.est[24]	106.877	3.088	100.100	105.600	106.900	108.200	113.600	1.003	5500
N.est[25]	114.060	3.367	107.097	112.700	114.000	115.400	121.600	1.004	9300
N.est[26]	121.113	3.485	113.800	119.600	121.000	122.500	128.900	1.003	15000
N.est[27]	131.672	3.815	122.900	130.200	131.900	133.300	139.700	1.006	2400
N.est[28]	134.296	3.997	126.200	132.600	134.100	135.800	143.400	1.004	1900
N.est[29]	142.957	4.285	133.800	141.200	143.000	144.600	152.500	1.002	15000
N.est[30]	149.959	4.432	140.400	148.100	149.900	151.800	159.502	1.003	15000
N.est[31]	154.522	4.512	145.297	152.600	154.200	156.300	165.200	1.001	15000
N.est[32]	171.024	5.192	158.500	169.100	171.600	173.300	180.800	1.002	3600

```
N.est[33]     163.302  4.790  153.500  161.300  163.100  165.200 174.200 1.006   1700
N.est[34]     163.798  4.838  153.200  161.900  163.900  165.700 174.200 1.003   1900
N.est[35]     160.386  4.761  149.500  158.500  160.800  162.400 170.100 1.003   9900
N.est[36]     138.573  4.560  131.300  136.300  137.600  140.000 150.700 1.005   1500
N.est[37]     156.366  4.569  146.000  154.500  156.700  158.300 165.800 1.003   9500
N.est[38]     154.378  4.838  146.100  152.100  153.500  156.000 166.400 1.003   1700
N.est[39]     189.695  5.767  175.697  187.500  190.500  192.300 200.600 1.004   2700
N.est[40]     193.223  5.984  180.697  190.800  193.000  195.500 206.900 1.003 15000
deviance     -203.878 69.995 -358.600 -245.525 -191.900 -151.200 -97.989 1.012    180
```

We now produce a plot with the counts and the estimates from the two models:

```
plot(peregrine$Pairs, type = "l", ylab = "Population size", xlab = "Year",
lwd = 2, las = 1)
points(per1$mean$N.est, type = "l", col = "blue", lwd = 2)
points(per2$mean$N.est, type = "l", col = "blue", lwd = 2, lty = 2)
legend(y = 200, x = 1, legend = c("Counts", "SSM", "SSM on log scale"), lty
= c(1, 1, 2), col = c("black", "blue", "blue"), bty = "n", lwd = rep(2, 3))
```



The estimated population sizes are almost identical to the population counts; you can hardly see the different lines. This suggests that the observation error is small and the counts are accurate.

We may also constrain the growth rate under the state-space model to follow a linear trend. That is what we do next. We use again the model on the log-scale.

```
# Specify model in BUGS language
sink("ssm.bug")
cat("
model {
# Priors and constraints
logN[1] ~ dnorm(0, 0.01)           # Prior for initial population size
alpha ~ dnorm(0, 0.01)             # Prior for mean stochastic growth rate
beta ~ dnorm(0, 0.01)              # Prior for mean stochastic growth rate
sigma.proc ~ dunif(0, 2)           # Prior sd of state process
sigma2.proc <- pow(sigma.proc, 2)
tau.proc <- pow(sigma.proc, -2)
sigma.obs ~ dunif(0, 10)           # Prior sd of observation process
sigma2.obs <- pow(sigma.obs, 2)
tau.obs <- pow(sigma.obs, -2)
```

```
# State process
for (t in 1:(T-1)){
   r[t] ~ dnorm(mu.r[t], tau.proc)
   mu.r[t] <- alpha + beta * t
   logN[t+1] <- logN[t] + r[t]
   }

# Observation process
for (t in 1:T) {
   y[t] ~ dnorm(logN[t], tau.obs)
   N.est[t] <- exp(logN[t])        # Backtransformation from log-scale
   }
}
",fill = TRUE)
sink()


# Bundle data
bugs.data <- list(y = log(peregrine$Pairs), T = length(peregrine$Pairs))

# Initial values
inits <- function(){list(alpha = rnorm(1), beta = rnorm(1), sigma.obs =
runif(1, 0.5, 5), sigma.proc = runif(1, 0.5, 2), logN = c(runif(1, -1, 5),
rep(NA, (length(peregrine$Pairs)-1))))}

# Parameters monitored
parameters <- c("r", "alpha", "beta", "sigma2.obs", "sigma2.proc", "N.est")

# MCMC settings
niter <- 100000
nthin <- 10
nburn <- 50000
nchains <- 3

# Call WinBUGS from R (BRT 8 min)
per3 <- bugs(bugs.data, inits, parameters, "ssm.bug", n.chains = nchains,
n.thin = nthin, n.iter = niter, n.burnin = nburn, debug = TRUE,
bugs.directory = bugs.dir)

print(per3, 3)
Inference for Bugs model at "ssm.bug", fit using WinBUGS,
 3 chains, each with 1e+05 iterations (first 50000 discarded), n.thin = 10
 n.sims = 15000 iterations saved
               mean     sd     2.5%      25%      50%      75%    97.5%  Rhat n.eff
r[1]          0.254  0.050    0.122    0.234    0.269    0.283    0.326 1.013   240
r[2]         -0.127  0.044   -0.204   -0.150   -0.137   -0.108   -0.019 1.003  1000
r[3]         -0.099  0.046   -0.216   -0.119   -0.087   -0.075   -0.024 1.003  1200
r[4]         -0.548  0.061   -0.621   -0.588   -0.569   -0.523   -0.381 1.006   410
r[5]         -0.113  0.042   -0.210   -0.130   -0.109   -0.095   -0.028 1.005  4900
r[6]          0.135  0.049    0.008    0.116    0.148    0.160    0.209 1.003   940
r[7]         -0.045  0.041   -0.132   -0.064   -0.048   -0.028    0.050 1.005 15000
r[8]         -0.145  0.045   -0.220   -0.168   -0.156   -0.127   -0.030 1.008  1300
r[9]          0.150  0.042    0.046    0.134    0.158    0.170    0.230 1.006  2500
r[10]         0.139  0.040    0.047    0.123    0.140    0.156    0.224 1.006  6200
r[11]         0.157  0.041    0.065    0.140    0.159    0.174    0.245 1.005  1200
r[12]         0.082  0.042    0.003    0.062    0.075    0.099    0.181 1.005  1900
r[13]         0.281  0.044    0.172    0.264    0.290    0.303    0.354 1.009   710
r[14]         0.105  0.041    0.022    0.086    0.100    0.122    0.200 1.008  1000
r[15]         0.093  0.040    0.009    0.075    0.090    0.109    0.185 1.003  2100
r[16]         0.181  0.043    0.077    0.165    0.189    0.202    0.259 1.005  3500
r[17]        -0.005  0.043   -0.082   -0.026   -0.013    0.012    0.102 1.002  8900
r[18]         0.113  0.041    0.020    0.096    0.116    0.131    0.200 1.002  7200
r[19]         0.106  0.041    0.017    0.089    0.105    0.123    0.195 1.004  4400
```

```
r[20]          0.155  0.041    0.060    0.137    0.157    0.173    0.241 1.002 15000
r[21]          0.094  0.041    0.003    0.077    0.093    0.111    0.183 1.004 11000
r[22]          0.058  0.041   -0.027    0.040    0.055    0.074    0.152 1.006  2800
r[23]          0.114  0.041    0.020    0.098    0.117    0.132    0.197 1.005  2600
r[24]          0.065  0.041   -0.021    0.048    0.064    0.082    0.155 1.009  6100
r[25]          0.061  0.040   -0.026    0.044    0.060    0.077    0.152 1.005  4900
r[26]          0.084  0.040   -0.008    0.067    0.086    0.102    0.168 1.006   720
r[27]          0.019  0.041   -0.067    0.001    0.016    0.035    0.113 1.006  1300
r[28]          0.064  0.041   -0.029    0.047    0.065    0.081    0.153 1.005  1800
r[29]          0.047  0.041   -0.043    0.030    0.048    0.064    0.136 1.004  2600
r[30]          0.030  0.041   -0.056    0.012    0.028    0.047    0.124 1.002 15000
r[31]          0.101  0.043    0.000    0.085    0.107    0.121    0.181 1.005  2300
r[32]         -0.046  0.043   -0.127   -0.065   -0.051   -0.029    0.057 1.005  2700
r[33]          0.003  0.042   -0.091   -0.014    0.005    0.021    0.091 1.004 15000
r[34]         -0.022  0.042   -0.119   -0.040   -0.020   -0.004    0.064 1.004  2600
r[35]         -0.145  0.045   -0.220   -0.168   -0.156   -0.128   -0.032 1.006   580
r[36]          0.121  0.044    0.013    0.103    0.130    0.143    0.198 1.004   920
r[37]         -0.012  0.044   -0.089   -0.033   -0.021    0.005    0.096 1.004  1800
r[38]          0.204  0.045    0.087    0.186    0.215    0.228    0.278 1.007   730
r[39]          0.021  0.043   -0.060    0.001    0.014    0.038    0.123 1.005  1800
alpha          0.013  0.051   -0.086   -0.021    0.013    0.047    0.113 1.001 11000
beta           0.002  0.002   -0.003    0.000    0.002    0.003    0.006 1.001 10000
sigma2.obs     0.001  0.002    0.000    0.000    0.001    0.002    0.005 1.016   150
sigma2.proc    0.024  0.006    0.014    0.020    0.023    0.028    0.039 1.001  5800
N.est[1]      34.342  1.156   32.410   33.800   34.100   34.760   37.310 1.008   610
N.est[2]      44.276  1.567   40.220   43.640   44.700   45.130   46.680 1.009   350
N.est[3]      38.982  1.231   36.170   38.510   39.000   39.480   41.590 1.004  1800
N.est[4]      35.301  1.311   31.810   34.770   35.700   36.070   37.160 1.005   570
N.est[5]      20.407  0.811   19.310   19.960   20.160   20.690   22.550 1.006   580
N.est[6]      18.221  0.655   17.200   17.900   18.060   18.430   19.910 1.004  1200
N.est[7]      20.851  0.659   19.260   20.580   20.950   21.150   22.120 1.003  2800
N.est[8]      19.926  0.604   18.570   19.670   19.970   20.180   21.170 1.004  7700
N.est[9]      17.235  0.601   16.300   16.930   17.080   17.430   18.780 1.009  1500
N.est[10]     20.030  0.602   18.810   19.770   20.000   20.250   21.470 1.005 15000
N.est[11]     23.017  0.698   21.500   22.730   23.010   23.300   24.550 1.006  2500
N.est[12]     26.934  0.826   25.110   26.600   26.970   27.270   28.690 1.004  2800
N.est[13]     29.243  0.922   27.600   28.810   29.080   29.570   31.510 1.005  2900
N.est[14]     38.727  1.196   35.880   38.250   38.910   39.260   40.980 1.008   850
N.est[15]     43.000  1.306   40.200   42.460   42.990   43.520   45.900 1.006  5200
N.est[16]     47.174  1.430   44.350   46.547   47.050   47.730   50.540 1.003  2800
N.est[17]     56.562  1.789   52.180   55.870   56.850   57.390   59.920 1.007 12000
N.est[18]     56.290  1.752   52.940   55.520   56.090   56.940   60.500 1.002 15000
N.est[19]     63.050  1.919   59.030   62.230   63.000   63.780   67.420 1.004  8600
N.est[20]     70.121  2.160   65.710   69.210   70.020   70.960   75.100 1.004  6300
N.est[21]     81.843  2.505   76.230   80.810   81.940   82.890   87.220 1.003 15000
N.est[22]     89.888  2.730   83.600   88.790   89.950   90.990   95.770 1.006  4500
N.est[23]     95.235  2.906   89.310   94.010   95.050   96.320  102.000 1.005  6500
N.est[24]    106.788  3.220   99.720  105.400  106.900  108.100  113.602 1.006  4600
N.est[25]    113.980  3.418  106.497  112.500  114.000  115.400  121.500 1.006 15000
N.est[26]    121.119  3.661  113.397  119.600  121.000  122.500  129.400 1.005  2300
N.est[27]    131.704  3.967  122.800  130.000  131.900  133.300  140.300 1.005  1200
N.est[28]    134.202  4.151  125.600  132.500  134.000  135.800  143.600 1.005  5500
N.est[29]    143.045  4.421  133.697  141.200  143.000  144.800  153.000 1.007  2800
N.est[30]    149.923  4.573  140.100  148.000  149.900  151.700  160.000 1.003 15000
N.est[31]    154.491  4.805  144.800  152.500  154.200  156.300  165.600 1.003  7500
N.est[32]    170.954  5.361  158.100  168.900  171.600  173.400  181.400 1.005  2800
N.est[33]    163.345  5.032  153.000  161.300  163.100  165.300  174.900 1.002 15000
N.est[34]    163.898  5.055  152.800  161.800  163.900  165.900  175.000 1.004 12000
N.est[35]    160.272  4.951  148.800  158.200  160.700  162.400  170.300 1.004  3300
N.est[36]    138.600  4.683  131.000  136.300  137.600  140.200  150.700 1.006   630
N.est[37]    156.441  4.833  145.500  154.500  156.800  158.400  166.400 1.003  9900
N.est[38]    154.530  5.060  146.000  152.100  153.500  156.400  167.500 1.006  1300
N.est[39]    189.510  5.900  175.200  187.100  190.500  192.200  200.700 1.004  1600
N.est[40]    193.461  6.176  180.900  190.900  193.100  195.800  207.600 1.005 13000
deviance    -202.747 75.384 -382.100 -249.500 -183.900 -145.500  -98.399 1.016   150
```

Note that the process variation did not change compared to the model without the linear trend in the annual population growth rates. The linear trend in the annual growth rates was small (`beta`). Thus, we can conclude that the average growth rate hardly changed deterministically over time. Since the mean growth rate is larger than 0, the population size increased.

Next we fit a model with a linear change in the observation error. We have to think about a reasonable link function for this model. The log-link appears to be the most appropriate, since it ensures that the error (variance) cannot become negative. Thus, we write the model in the following way:

```
# Specify model in BUGS language
sink("ssmTrendError.bug")
cat("
model {
# Priors and constraints
logN[1] ~ dnorm(3.5, 100)              # Prior for initial population size
mean.r ~ dnorm(0, 0.01)I(-20,20)
beta ~ dnorm(0, 0.01)I(-20,20)         # Prior for slope
mean.err ~ dnorm(0, 0.01) I(-20,20)    # Prior for mean log(obs error)
sigma.proc ~ dunif(0, 5)               # Prior sd of state process
sigma2.proc <- pow(sigma.proc, 2)
tau.proc <- pow(sigma.proc, -2)

# State process
for (t in 1:(T-1)){
   r[t] ~ dnorm(mean.r, tau.proc)
   logN[t+1] <- logN[t] + r[t]
   }

# Observation process
for (t in 1:T) {
   y[t] ~ dnorm(logN[t], tau.obs[t])
   tau.obs[t] <- 1/sigma2.obs[t]
   log(sigma2.obs[t]) <- mean.err + beta * period[t]
   N.est[t] <- exp(logN[t])        # Backtransformation from log-scale
   }
}
",fill=TRUE)
sink()

# Bundle data
bugs.data <- list(y = log(peregrine$Pairs), period = scale(1:
length(peregrine$Pairs))[,1], T = length(peregrine$Pairs))

# Initial values
inits <- function(){list(sigma.proc = runif(1, 0, 1), mean.r = rnorm(1),
beta = runif(1, -1, 1), mean.err = rnorm(1), logN = c(runif(1, 3, 4),
rep(NA, (length(peregrine$Pairs)-1))))}

# Parameters monitored
parameters <- c("r", "mean.r", "mean.err", "sigma2.obs", "sigma2.proc",
"N.est", "beta")

# MCMC settings
niter <- 100000
nthin <- 10
nburn <- 50000
nchains <- 3

# Call WinBUGS from R (BRT 8 min)
```

```
per4 <- bugs(bugs.data, inits, parameters, "ssmTrendError.bug", n.chains =
nchains, n.thin = nthin, n.iter = niter, n.burnin = nburn, debug = TRUE,
bugs.directory = bugs.dir)

print(per4, 3)
Inference for Bugs model at "ssmTrendError.bug", fit using WinBUGS,
 3 chains, each with 1e+05 iterations (first 50000 discarded), n.thin = 10
 n.sims = 15000 iterations saved
                mean     sd    2.5%     25%     50%     75%   97.5%  Rhat n.eff
r[1]           0.021  0.160  -0.114  -0.096  -0.089   0.177   0.280 6.750     3
r[2]          -0.065  0.075  -0.146  -0.141  -0.093   0.023   0.056 5.287     3
r[3]          -0.111  0.045  -0.235  -0.142  -0.095  -0.080  -0.057 2.697     4
r[4]          -0.394  0.153  -0.588  -0.497  -0.437  -0.271  -0.061 1.564     8
r[5]          -0.084  0.094  -0.235  -0.168  -0.105  -0.003   0.070 3.726     4
r[6]           0.040  0.088  -0.097  -0.043   0.050   0.114   0.154 1.962     5
r[7]          -0.078  0.047  -0.180  -0.121  -0.055  -0.049  -0.010 2.607     4
r[8]          -0.071  0.092  -0.163  -0.148  -0.120   0.033   0.078 5.176     3
r[9]           0.092  0.052   0.020   0.061   0.073   0.160   0.164 1.531     8
r[10]          0.119  0.070  -0.054   0.050   0.140   0.162   0.240 1.759     6
r[11]          0.161  0.041   0.109   0.129   0.160   0.183   0.269 2.298     4
r[12]          0.069  0.034   0.011   0.048   0.071   0.090   0.162 1.350    13
r[13]          0.305  0.035   0.189   0.296   0.316   0.324   0.340 1.299    14
r[14]          0.110  0.036   0.016   0.092   0.098   0.143   0.173 2.802     4
r[15]          0.077  0.029   0.011   0.071   0.085   0.089   0.124 2.020     5
r[16]          0.195  0.029   0.132   0.187   0.193   0.207   0.266 1.129    82
r[17]         -0.013  0.024  -0.076  -0.030  -0.017   0.002   0.026 1.395     9
r[18]          0.117  0.018   0.087   0.106   0.118   0.126   0.159 1.143    48
r[19]          0.104  0.017   0.064   0.097   0.105   0.111   0.142 1.130   140
r[20]          0.158  0.016   0.119   0.153   0.158   0.166   0.191 1.147    78
r[21]          0.094  0.013   0.063   0.089   0.093   0.099   0.122 1.057    88
r[22]          0.053  0.010   0.033   0.049   0.054   0.056   0.079 1.103   290
r[23]          0.119  0.010   0.095   0.116   0.119   0.122   0.142 1.076   350
r[24]          0.063  0.009   0.044   0.061   0.063   0.066   0.081 1.104   180
r[25]          0.060  0.008   0.046   0.058   0.060   0.062   0.078 1.102   260
r[26]          0.087  0.007   0.070   0.085   0.087   0.089   0.100 1.095   230
r[27]          0.015  0.006   0.002   0.013   0.015   0.017   0.029 1.089   740
r[28]          0.065  0.006   0.052   0.063   0.065   0.066   0.075 1.113   400
r[29]          0.048  0.006   0.039   0.047   0.048   0.049   0.060 1.160   420
r[30]          0.026  0.006   0.016   0.025   0.026   0.027   0.036 1.188  4000
r[31]          0.110  0.007   0.099   0.110   0.110   0.111   0.120 1.223  1600
r[32]         -0.054  0.007  -0.064  -0.054  -0.054  -0.053  -0.043 1.247  1400
r[33]          0.006  0.007  -0.004   0.006   0.006   0.007   0.017 1.258  5000
r[34]         -0.019  0.009  -0.030  -0.019  -0.018  -0.018  -0.010 1.270  1400
r[35]         -0.161  0.012  -0.169  -0.162  -0.161  -0.161  -0.145 1.291   170
r[36]          0.135  0.013   0.119   0.136   0.136   0.137   0.144 1.295   170
r[37]         -0.024  0.016  -0.034  -0.026  -0.026  -0.026  -0.005 1.302   120
r[38]          0.220  0.020   0.191   0.222   0.222   0.222   0.229 1.310    84
r[39]          0.012  0.020   0.000   0.010   0.010   0.011   0.033 1.296   290
mean.r         0.041  0.022  -0.001   0.027   0.041   0.055   0.084 1.034    70
mean.err     -11.145  3.409 -19.510 -11.270 -10.130  -9.016  -7.198 1.802     6
sigma2.obs[1]     0.267   0.331   0.000   0.044   0.176   0.368   1.099  2.114     5
sigma2.obs[2]     0.173   0.204   0.000   0.033   0.117   0.243   0.697  2.102     5
sigma2.obs[3]     0.113   0.128   0.000   0.025   0.077   0.162   0.444  2.089     5
sigma2.obs[4]     0.074   0.081   0.000   0.018   0.050   0.109   0.282  2.075     5
sigma2.obs[5]     0.049   0.052   0.000   0.013   0.032   0.073   0.179  2.060     5
sigma2.obs[6]     0.032   0.034   0.000   0.010   0.021   0.049   0.116  2.043     5
sigma2.obs[7]     0.022   0.022   0.000   0.007   0.014   0.033   0.077  2.024     5
sigma2.obs[8]     0.014   0.015   0.000   0.005   0.009   0.022   0.051  2.004     5
sigma2.obs[9]     0.010   0.010   0.000   0.003   0.006   0.015   0.034  1.982     5
sigma2.obs[10]    0.007   0.007   0.000   0.002   0.004   0.010   0.023  1.958     6
sigma2.obs[11]    0.004   0.005   0.000   0.001   0.003   0.007   0.016  1.931     6
sigma2.obs[12]    0.003   0.003   0.000   0.001   0.002   0.004   0.012  1.902     6
sigma2.obs[13]    0.002   0.002   0.000   0.001   0.001   0.003   0.008  1.869     6
sigma2.obs[14]    0.001   0.002   0.000   0.000   0.001   0.002   0.006  1.834     6
sigma2.obs[15]    0.001   0.001   0.000   0.000   0.001   0.001   0.004  1.795     6
sigma2.obs[16]    0.001   0.001   0.000   0.000   0.000   0.001   0.003  1.753     7
sigma2.obs[17]    0.000   0.001   0.000   0.000   0.000   0.001   0.002  1.707     7
sigma2.obs[18]    0.000   0.001   0.000   0.000   0.000   0.000   0.002  1.657     7
sigma2.obs[19]    0.000   0.000   0.000   0.000   0.000   0.000   0.001  1.605     8
sigma2.obs[20]    0.000   0.000   0.000   0.000   0.000   0.000   0.001  1.550     9
sigma2.obs[21]    0.000   0.000   0.000   0.000   0.000   0.000   0.001  1.495    10
sigma2.obs[22]    0.000   0.000   0.000   0.000   0.000   0.000   0.001  1.441    11
sigma2.obs[23]    0.000   0.000   0.000   0.000   0.000   0.000   0.001  1.392    12
sigma2.obs[24]    0.000   0.000   0.000   0.000   0.000   0.000   0.000  1.350    13
sigma2.obs[25]    0.000   0.000   0.000   0.000   0.000   0.000   0.000  1.322    13
sigma2.obs[26]    0.000   0.000   0.000   0.000   0.000   0.000   0.000  1.310    13
sigma2.obs[27]    0.000   0.000   0.000   0.000   0.000   0.000   0.000  1.320    12
```

```
sigma2.obs[28]    0.000   0.000    0.000    0.000    0.000    0.000    0.000  1.352    11
sigma2.obs[29]    0.000   0.000    0.000    0.000    0.000    0.000    0.000  1.403    10
sigma2.obs[30]    0.000   0.000    0.000    0.000    0.000    0.000    0.000  1.470     8
sigma2.obs[31]    0.000   0.000    0.000    0.000    0.000    0.000    0.000  1.545     8
sigma2.obs[32]    0.000   0.000    0.000    0.000    0.000    0.000    0.000  1.624     7
sigma2.obs[33]    0.000   0.000    0.000    0.000    0.000    0.000    0.001  1.703     6
sigma2.obs[34]    0.000   0.000    0.000    0.000    0.000    0.000    0.001  1.777     6
sigma2.obs[35]    0.000   0.001    0.000    0.000    0.000    0.000    0.001  1.846     6
sigma2.obs[36]    0.000   0.001    0.000    0.000    0.000    0.000    0.001  1.908     5
sigma2.obs[37]    0.000   0.001    0.000    0.000    0.000    0.000    0.001  1.964     5
sigma2.obs[38]    0.000   0.002    0.000    0.000    0.000    0.000    0.002  2.013     5
sigma2.obs[39]    0.000   0.003    0.000    0.000    0.000    0.000    0.002  2.056     5
sigma2.obs[40]    0.000   0.005    0.000    0.000    0.000    0.000    0.003  2.094     5
sigma2.proc    0.018   0.006    0.010    0.014    0.017    0.021    0.033 1.219    16
N.est[1]      39.153   6.889   30.689   34.000   36.310   47.590   51.210 3.932     3
N.est[2]      39.834   5.846   32.320   33.270   44.680   45.010   46.870 2.954     4
N.est[3]      37.115   3.710   31.300   33.400   38.990   40.700   42.960 1.743     6
N.est[4]      33.202   3.130   24.490   31.100   34.495   35.980   37.050 1.279    13
N.est[5]      22.577   3.709   18.090   19.840   20.000   27.000   28.920 1.972     5
N.est[6]      20.619   2.208   16.930   18.010   20.760   22.510   24.570 1.457     9
N.est[7]      21.397   1.472   15.730   21.000   21.750   22.150   23.490 1.310    25
N.est[8]      19.781   1.327   15.100   19.400   20.000   20.810   21.080 2.120     5
N.est[9]      18.429   1.322   15.450   17.010   18.330   19.810   20.510 1.702     7
N.est[10]     20.188   1.160   17.410   19.630   20.000   20.470   23.410 1.467    11
N.est[11]     22.717   0.935   20.730   22.150   22.990   23.090   24.440 1.362    11
N.est[12]     26.689   0.908   25.370   26.090   26.660   27.000   29.190 1.539     7
N.est[13]     28.612   1.130   27.190   27.720   28.590   29.000   31.730 1.435    10
N.est[14]     38.791   0.919   37.330   37.950   39.000   39.290   41.210 1.972     5
N.est[15]     43.319   0.870   41.620   42.930   43.010   43.700   45.260 2.027     5
N.est[16]     46.773   0.993   44.580   46.300   46.950   47.050   48.890 1.241    25
N.est[17]     56.834   0.996   54.940   56.240   56.930   57.310   58.920 1.224    14
N.est[18]     56.099   0.799   54.110   55.820   56.000   56.460   57.560 1.247    15
N.est[19]     63.084   0.768   61.590   62.740   63.000   63.362   65.000 1.151    43
N.est[20]     70.009   0.737   68.710   69.740   70.000   70.180   71.820 1.106   450
N.est[21]     82.010   0.858   80.140   81.690   82.000   82.370   83.930 1.074   100
N.est[22]     90.057   0.699   88.590   89.830   90.010   90.300   91.590 1.092   260
N.est[23]     94.974   0.669   93.560   94.700   95.000   95.200   96.450 1.079  2800
N.est[24]    106.977   0.699  105.500  106.800  107.000  107.200  108.400 1.101   100
N.est[25]    113.974   0.678  112.600  113.800  114.000  114.100  115.200 1.086   580
N.est[26]    121.043   0.620  119.900  120.900  121.000  121.200  122.400 1.092   230
N.est[27]    132.000   0.645  130.700  131.800  132.000  132.200  133.300 1.086  2700
N.est[28]    134.009   0.614  132.900  133.900  134.000  134.100  135.100 1.106   930
N.est[29]    142.961   0.632  141.700  142.800  143.000  143.100  144.000 1.141   470
N.est[30]    150.015   0.664  148.900  149.900  150.000  150.100  151.200 1.177  1800
N.est[31]    154.017   0.675  153.000  153.900  154.000  154.100  155.200 1.203  3200
N.est[32]    171.981   0.893  170.700  171.900  172.000  172.100  173.100 1.241  1700
N.est[33]    163.001   0.844  161.800  162.900  163.000  163.100  164.100 1.251  4200
N.est[34]    164.011   0.908  162.900  163.900  164.000  164.100  165.200 1.264 15000
N.est[35]    160.959   1.096  159.600  160.900  161.000  161.000  162.000 1.278   830
N.est[36]    137.092   1.336  136.300  137.000  137.000  137.000  138.700 1.293   220
N.est[37]    156.956   1.648  155.300  157.000  157.000  157.000  158.100 1.289   860
N.est[38]    153.179   2.191  152.100  153.000  153.000  153.000  155.300 1.300   160
N.est[39]    190.798   2.748  187.800  191.000  191.000  191.000  192.100 1.300   170
N.est[40]    193.031   3.327  190.900  193.000  193.000  193.000  194.900 1.290 15000
beta          -3.250   2.674   -5.687   -4.774   -4.152   -3.009    5.177 1.960     6
deviance    -332.346 136.303 -666.402 -335.000 -291.600 -246.900 -176.000 1.807     6
```

Unfortunately, we have not managed to get the model to convergence - even if we run the model much longer, it did not converge. Probably this has to do with the fact that the observation variance is very small, and estimation of this small number is apparently very hard. As the observation variance is small, a temporal trend is likely to be unimportant for the estimates of the other parameters, even if it would exist. It is also important to note that we accounted for a possible trend in the observer variance (called error), and not in the detection probability *per se* with this model.

Chapter 6

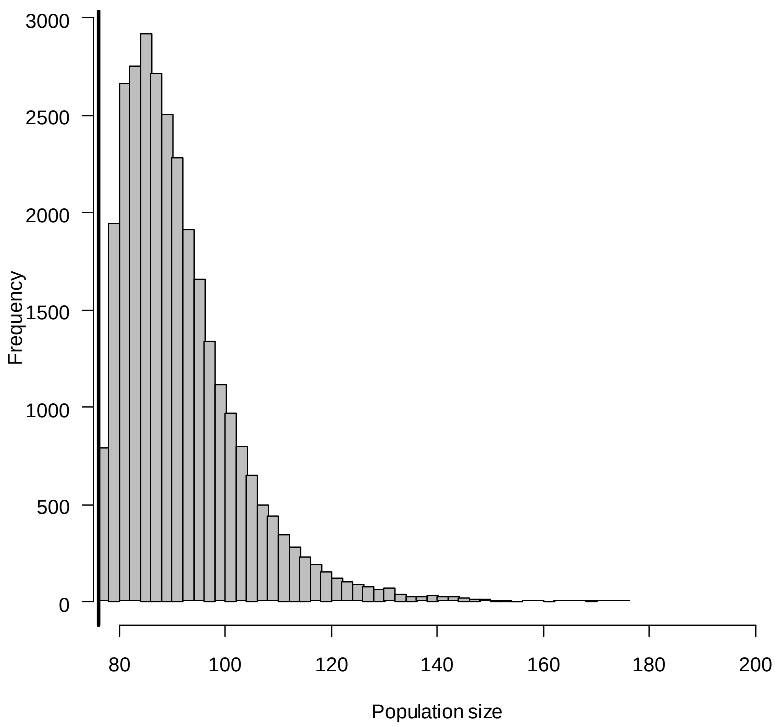
Exercise 1

Task: Rewrite the model M_h in section 6.2.4. for encounter history instead of capture frequency data and fit it. (The response will be Bernoulli instead of Binomial.)

Solution: We start by executing the data-generating function from section 6.2.4 once to get one data set.

```
data <- data.fn(N = 100, mean.p = 0.4, T = 5, sd = 1)
attach(data)

# Augment data set
nz <- 100
yaug <- rbind(yobs, array(0, dim=c(nz, T)))


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

# Priors
omega ~ dunif(0, 1)
mean.lp <- logit(mean.p)
mean.p ~ dunif(0, 1)
tau <- 1 / (sd * sd)
sd ~ dunif(0, 5)

# Likelihood
for (i in 1:M){
   z[i] ~ dbern(omega)
   logit(p[i]) <- eps[i]
   eps[i] ~ dnorm(mean.lp, tau)I(-16, 16)
   for(j in 1:T){
      p.eff[i, j] <- z[i] * p[i]
      y[i, j] ~ dbern(p.eff[i, j])
      } #j
   } #i

# Derived quantities
N <- sum(z[])
}
",fill = TRUE)
sink()

# Bundle data
win.data <- list(y = yaug, M = nrow(yaug), T = ncol(yaug))

# Initial values
inits <- function() list(z = rep(1, nrow(yaug)), sd = runif(1, 0.1,
0.9))
```

```
# Parameters monitored
params <- c("N", "mean.p", "sd", "omega")

# MCMC settings
ni <- 25000
nt <- 2
nb <- 5000
nc <- 3

# Call WinBUGS from R (BRT 4 min)
out <- bugs(win.data, inits, params, "model.txt", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE,
bugs.directory = bugs.dir, working.directory = getwd())

# Summarize posteriors
print(out, dig = 3)
hist(out$sims.list$N, nclass = 50, col = "gray", main = "", xlab =
"Population size", las = 1, xlim = c(80, 200))
abline(v = data$C, col = "black", lwd = 3)
```



The Bernoulli and the binomial version with index N of the model are equivalent. If we don't need to model any time-specific effects (this includes behavioural response), we can fit the Binomial version of the model to the aggregate data (the capture frequencies), since the binomial is simply a sum of N Bernoullis.

Exercise 2

Task: Try to fit the model with permanent trap response to the bird survey data. Imagine a biological situation that might represent this model, on the part of the observer? On the part of the animal?

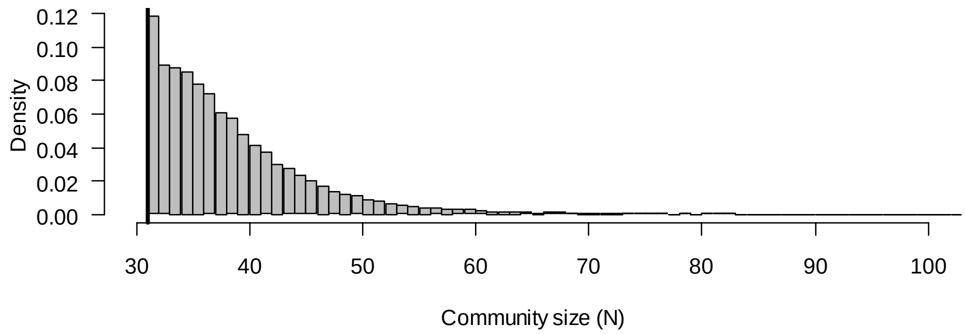
Solution: You must construct an explanatory array of the same dimension as has the augmented data set. The explanatory array indicates whether individual *i* at occasion *j* has ever been captured before (1) or not (0). Here we provide code to construct this array in R, but you could also do this in Excel by hand or even program it directly in WinBUGS using the function `step()` (see the WinBUGS manual for how `step()` works).

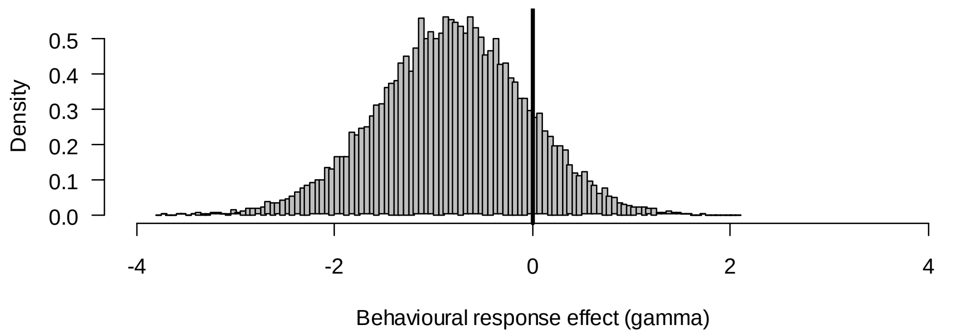
You will need long Markov chains, with long burnin, to obtain convergence. Here we assume that you have the data all read in your workspace. Once you have the covariate array X (‚seen.before'), essentially all you have to do is change one line of code in the model statement, as indicated below.

```
# Construct the array X, which indicates capture ever before
# of individual i at occasion j (solution due to Tomas Telensky)
X <- as.matrix(y)
for (i in 1:nrow(y)){
   seen.before <- 0
   for (j in 1:ncol(y)){
      X[i,j] <- seen.before
      if (y[i,j] == 1)
      seen.before <- 1
      }
   }

# Check whether X correct (it is)
head(y)
head(X)

# Bundle data, including array X
win.data <- list(y = as.matrix(y), M = nrow(y), T = ncol(y), X =
as.matrix(X))

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

# Priors
omega ~ dunif(0, 1)
for (j in 1:T){
   alpha[j] <- log(mean.p[j] / (1-mean.p[j])) # Define logit
   mean.p[j] ~ dunif(0, 1)      # Detection intercepts
   }
gamma ~ dnorm(0, 0.01)
tau <- 1 / (sd * sd)
sd ~ dunif(0, 5)

# Likelihood
for (i in 1:M){
   z[i] ~ dbern(omega)
   eps[i] ~ dnorm(0, tau)I(-16, 16)

   # First occasion: no term for recapture (gamma)
   y[i,1] ~ dbern(p.eff[i,1])
   p.eff[i,1] <- z[i] * p[i,1]
   p[i,1] <- 1 / (1 + exp(-lp[i,1]))
```

```
   lp[i,1] <- alpha[1] + eps[i]

   # All subsequent occasions: includes recapture term (gamma)
   for (j in 2:T){
      y[i,j] ~ dbern(p.eff[i,j])
      p.eff[i,j] <- z[i] * p[i,j]
      p[i,j] <- 1 / (1 + exp(-lp[i,j]))
      lp[i,j] <- alpha[j] + eps[i] + gamma * X[i,j] ## Only change
      } #j
   } #i

# Derived quantities
N <- sum(z[])
}
",fill = TRUE)
sink()

# Initial values
inits <- function() list(z = rep(1, nrow(y)), sd = runif(1, 0.1, 1))

# Parameters monitored
params <- c("N", "mean.p", "gamma", "sd", "omega")

# MCMC settings
ni <- 50000
nt <- 4
nb <- 10000
nc <- 3

# Call WinBUGS from R (BRT 33 min)
out <- bugs(win.data, inits, params, "M_tbh.txt", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE,
bugs.directory = bugs.dir, working.directory = getwd())

# Summarize posteriors and plot posteriors of N and gamma
print(out, dig = 2)
Inference for Bugs model at "M_tbh.txt", fit using WinBUGS,
 3 chains, each with 50000 iterations (first 10000 discarded),
n.thin = 4
 n.sims = 30000 iterations saved
            mean    sd   2.5%    25%    50%    75%  97.5% Rhat n.eff
N          39.08  7.66  31.00  34.00  37.00  42.00  59.00    1  6300
mean.p[1]   0.25  0.09   0.09   0.18   0.24   0.31   0.44    1 15000
mean.p[2]   0.35  0.12   0.14   0.27   0.35   0.43   0.59    1 12000
mean.p[3]   0.40  0.15   0.14   0.30   0.40   0.51   0.70    1  4400
mean.p[4]   0.34  0.15   0.09   0.23   0.33   0.44   0.65    1 16000
mean.p[5]   0.46  0.17   0.15   0.34   0.46   0.59   0.78    1  7000
gamma      -0.82  0.75  -2.32  -1.31  -0.81  -0.31   0.63    1  5000
sd          0.93  0.53   0.09   0.53   0.88   1.25   2.10    1  2900
omega       0.27  0.06   0.18   0.23   0.26   0.30   0.42    1 25000
deviance  213.86 18.11 185.20 201.10 211.40 223.90 256.70    1  3900

par(mfrow = c(2,1))
hist(out$sims.list$N, breaks = 100, col = "gray", main = "", xlab =
"Community size (N)", las = 1, xlim = c(30, 100), freq = FALSE)
abline(v = C, col = "black", lwd = 3)
```

```
hist(out$sims.list$gamma, breaks = 100, col = "gray", main = "",
xlab = "Behavioural response effect (gamma)", las = 1, xlim = c(-4,
4), freq = FALSE)
abline(v = 0, col = "black", lwd = 3)
```





The effect of having been seen before (i.e., the parameter gamma) is not “significant”; its 95%CRI includes zero. However, the point estimate is negative and 86% of the mass of its posterior distribution is negative. Thus, there is some evidence that there is actually a trap-shyness in this system: once detected, there may be a smaller chance of detecting a species again.

```
mean(out$sims.list$gamma < 0)
 [1] 0.8628333
```

What biological mechanisms could lead to permanent trap response ? On the part of the observer, it could be memory: if an elusive species has a very distinct activity centre of some kind that is stable over the entire course of a study, then, once discovered by the observer, that species may be much more likely to be detected again. An owl living in a tree hole where it may be seen at the hole entrance may provide one example and the active nest of a rare species another. On the part of the animal, a permanent trap response may arise if detection means capture and the capture event is a traumatic experience for the animal. It may then become much more shy after first capture and this effect may hold on for a while.

Exercise 3

Task: Check out the behavior of estimators in small sample situations, e.g., the heterogeneity model with 20 individuals and heterogeneity. Does this work?

Solution: Here, we only give a sketch of how a full-blown simulation study tackling such questions might be conducted whose aim is to gauge the accuracy (i.e., precision and bias) of the estimators under a model for a given scenario (e.g., the best guess of population size and detection probability in your favourite population of Ivory-billed woodpeckers). Normally, we would vary several factors in a factorial design, i.e., we would simulate, say, 100 or 1000 data sets for each combination of, say, N = (10, 20, 30, 40, 50), mean.p = (0.1, 0.2, 0.3, 0.4, 0.5), T = (2, 3, 4, 5) and sd = (0.1, 1, 5,). Clearly, this would be a major study, so here we only show how this may be done for a single design point, with N = 20, mean.p = 0.5, T = 5 and sd = 1.

We will define data structures (arrays) where we can save the results from the data simulation and the data analysis routines. Then, we will use a loop to produce 100 simulation replicates of the data generation/data analysis cycle and finally summarize the results, i.e., compare what the model told us about the population with what we know about that population. We will adapt the data generation function from the BPA book to directly return the capture frequencies as well. Also, we will package the analysis functions into an entire function, which we call `model.fn()`.

```
# New definition of data simulation function for model Mh
data.fn <- function(N = 100, mean.p = 0.4, T = 5, sd = 1){
   yfull <- yobs <- array(NA, dim = c(N, T))
   mean.lp <- log(mean.p / (1-mean.p))
   p.vec <- plogis(mean.lp+ rnorm(N, 0, sd))

   for (i in 1:N){
      yfull[i,] <- rbinom(n = T, size = 1, prob = p.vec[i])
      }

   ever.detected <- apply(yfull, 1, max)
   C <- sum(ever.detected)
   yobs <- yfull[ever.detected == 1,]
   yfreq <- sort(apply(yobs, 1, sum), decreasing = TRUE)

   cat(C, "out of", N, "animals present were detected.\n")
   hist(p.vec, xlim = c(0,1), nclass = 20, col = "gray", main = "", xlab =
"Detection probability", las = 1)
   return(list(N = N, p.vec = p.vec, mean.lp = mean.lp, C = C, T = T, yfull
= yfull, yobs = yobs, yfreq = yfreq))
   }

# Define a function to do data augmentation, run analysis using Mh and
return results all at once
model.fn <- function(nz = 200, ni = 25000, nt = 2, nb = 5000, nc = 3,
data.file = data, debg = FALSE){
# Function arguments:
# nz -- number of fake DA individuals
# ni/nt/nb/nc -- MCMC settings
# data.file -- name of the object with the simulated data
# debg -- setting of DEBUG argument in bugs()

# Do data augmentation and bundle data
yaug <- c(data.file$yfreq, rep(0, nz))
win.data <- list(y = yaug, M = length(yaug), T = data.file$T)
```

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

# Priors
omega ~ dunif(0, 1)
mean.lp <- logit(mean.p)
mean.p ~ dunif(0, 1)
tau <- pow(sd, -2)
sd ~ dunif(0, 5) # Might have to be be adapted depending on your data set

# Likelihood
for (i in 1:M){
   z[i] ~ dbern(omega)
   logit(p[i]) <- eps[i]
   eps[i] ~ dnorm(mean.lp, tau)I(-16, 16)
   p.eff[i] <- z[i] * p[i]
   y[i] ~ dbin(p.eff[i], T)
   }

# Derived quantities
N <- sum(z[])
}
",fill = TRUE)
sink()

# Initial values
inits <- function() list(z = rep(1, length(yaug)), sd = runif(1, 0.1, 0.9))

# Parameters monitored
params <- c("N", "mean.p", "sd", "omega")

# Call WinBUGS from R (BRT 6 min)
out <- bugs(win.data, inits, params, "model.txt", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = debg, bugs.directory =
bugs.dir, working.directory = getwd())

# Summarize posteriors
print(out, dig = 3)
hist(out$sims.list$N, nclass = 50, col = "gray", main = "", xlab =
"Population size", las = 1, xlim = c(0, 1.5*nz))
abline(v = data$C, col = "black", lwd = 3)

return(post.estimates = out$summary)
}
```

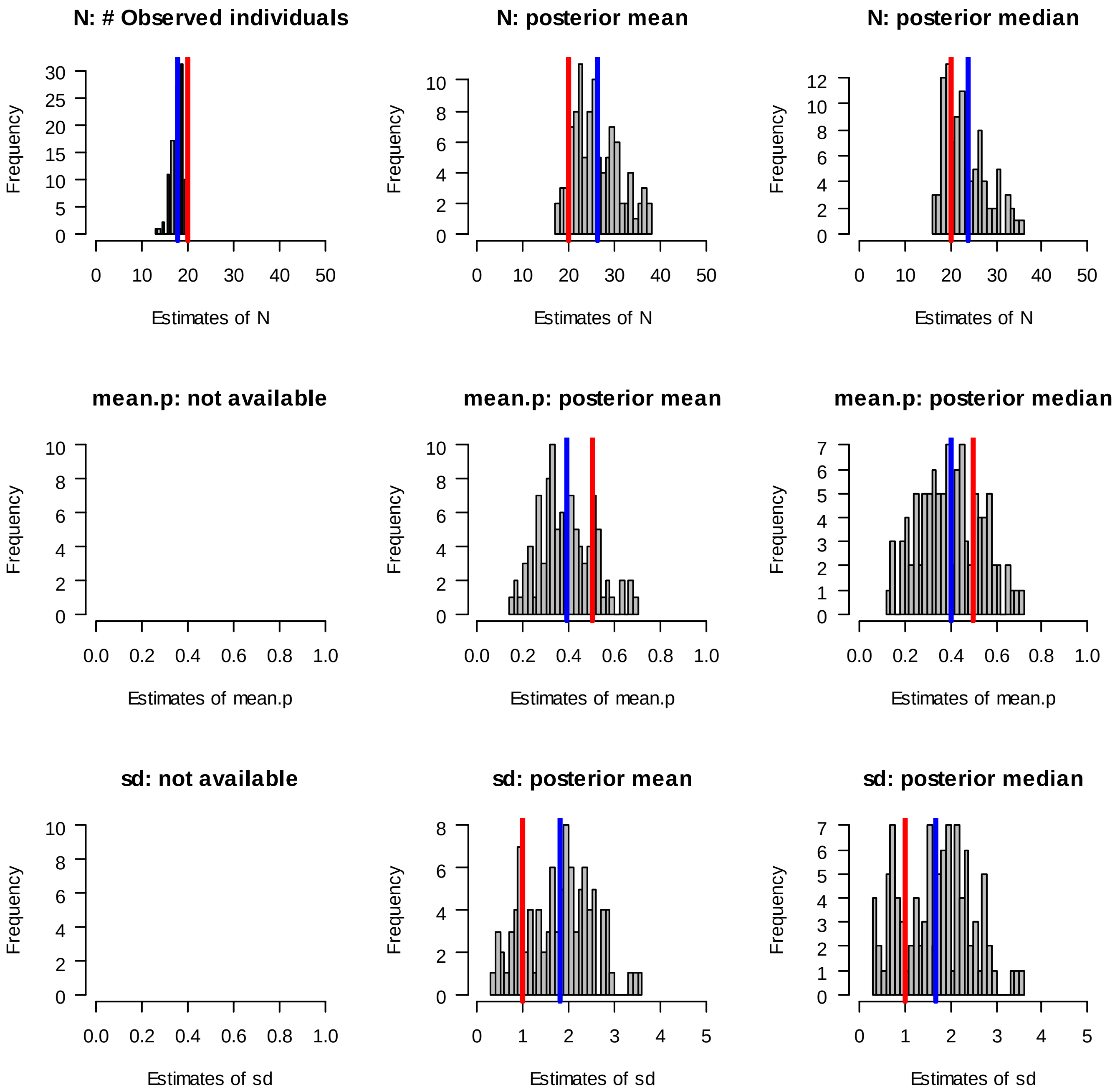
Try out a single data simulation/data analysis cycle.

```
data <- data.fn(N = 20, mean.p = 0.5, T = 5, sd = 1)
estimates <- model.fn(nz = 150, ni = 250, nt = 2, nb = 50, nc = 3,
data.file = data, debg = TRUE)
```

That seems to work. Now we run 100 simulation replicates for a single design point (N = 20, mean.p = 0.5, T = 5, sd = 1). The next block of code could be repeated for each design point of a larger, genuine simulation exercise.

```
# Set up data structures to hold the results
simreps <- 100
```

```
data.sets <- array(NA, dim = c(20, simreps))
solutions <- array(NA, dim = c(5, 9, simreps))
rownames(solutions) <- rownames(estimates)
colnames(solutions) <- colnames(estimates)

# Run data generation/data analysis cycle simrep times
for (i in 1:simreps){
   cat(paste("\n\n*** SimRep", i, "***\n"))
   data <- data.fn(N = 20, mean.p = 0.5, T = 5, sd = 1)
   data.sets[1:data$C,i] <- data$yfreq
   estimates <- model.fn(nz = 50, ni = 25000, nt = 5, nb = 10000, nc = 3,
data.file = data, debg = FALSE)
   solutions[,,i] <- estimates
   }

# Get the number of observed individuals as a simple estimator of N
Nobs <- array(NA, dim = simreps)
for (i in 1:simreps){
   Nobs[i] <- sum(!is.na(data.sets[,i]))
   }

# Summarize simulation results
par(mfrow = c(3, 3))
hist(Nobs, breaks = 25, col = "grey", xlab = "Estimates of N", main = "N: #
Observed individuals", xlim = c(0, 50), las = 1)
abline(v = 20, col = "red", lwd = 3)
abline(v = mean(Nobs), col = "blue", lwd = 3)

hist(solutions[1,1,], breaks = 25, col = "grey", xlab = "Estimates of N",
main = "N: posterior mean", xlim = c(0, 50), las = 1)
abline(v = 20, col = "red", lwd = 3)
abline(v = mean(solutions[1,1,]), col = "blue", lwd = 3)
hist(solutions[1,5,], breaks = 25, col = "grey", xlab = "Estimates of N",
main = "N: posterior median", xlim = c(0, 50), las = 1)
abline(v = 20, col = "red", lwd = 3)
abline(v = mean(solutions[1,5,]), col = "blue", lwd = 3)

hist(solutions[5,1,], breaks = 25, col = "grey", xlab = "Estimates of
mean.p", main = "mean.p: not available", xlim = c(0, 1), las = 1)
hist(solutions[2,1,], breaks = 25, col = "grey", xlab = "Estimates of
mean.p", main = "mean.p: posterior mean", xlim = c(0, 1), las = 1)
abline(v = 0.5, col = "red", lwd = 3)
abline(v = mean(solutions[2,1,]), col = "blue", lwd = 3)
hist(solutions[2,5,], breaks = 25, col = "grey", xlab = "Estimates of
mean.p", main = "mean.p: posterior median", xlim = c(0, 1), las = 1)
abline(v = 0.5, col = "red", lwd = 3)
abline(v = mean(solutions[2,5,]), col = "blue", lwd = 3)

hist(solutions[5,1,], breaks = 25, col = "grey", xlab = "Estimates of sd",
main = "sd: not available", xlim = c(0, 1), las = 1)
hist(solutions[3,1,], breaks = 25, col = "grey", xlab = "Estimates of sd",
main = "sd: posterior mean", xlim = c(0, 5), las = 1)
abline(v = 1, col = "red", lwd = 3)
abline(v = mean(solutions[3,1,]), col = "blue", lwd = 3)
hist(solutions[3,5,], breaks = 25, col = "grey", xlab = "Estimates of sd",
main = "sd: posterior median", xlim = c(0, 5), las = 1)
abline(v = 1, col = "red", lwd = 3)
abline(v = mean(solutions[3,5,]), col = "blue", lwd = 3)
```



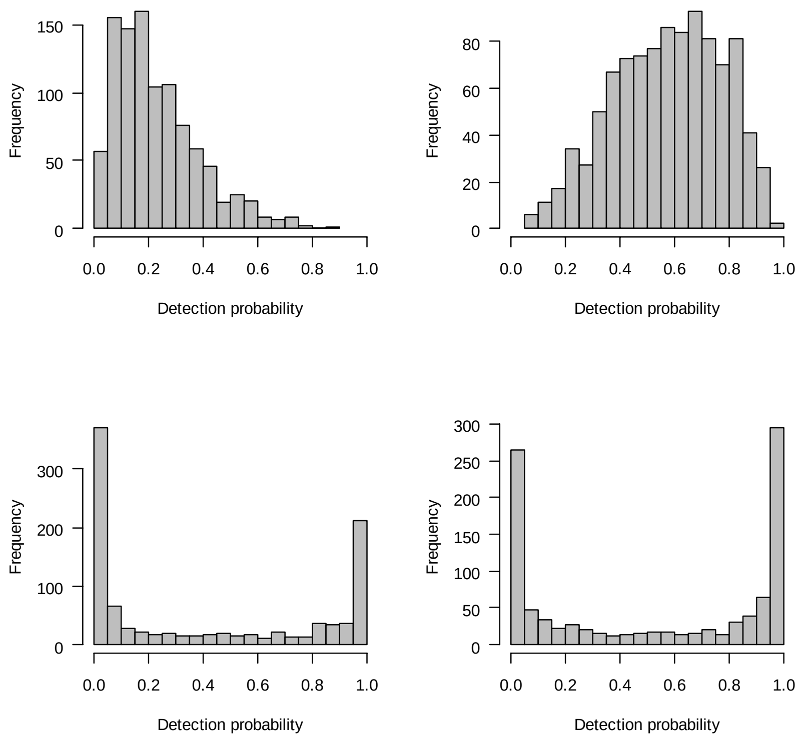
We see that with a very small sample size (N = 20) and with the chosen values of mean.p, T and sd, the model-based estimates of N can be quite bad. The posterior median may be a little better estimator than the posterior mean. The observed number of individuals as a conventional estimator of population size, though not based on an explicit model, is far better in this case in terms of its accuracy: for it, the blue line (the mean of the estimator) is much closer to the red line (truth) than what we get for either the posterior mean or the posterior median.

Exercise 4

Task: Generate data with individual heterogeneity in p and fit model M_0. See how well N is estimated.

Solution: We could run a little simulation as for exercise 3, but here we simply use the same data-generating function to obtain only one large data set for each of varying mean.p and sd and then fit model M_0 to each. Our scenarios will be low and high mean.p (0.2, 0.6) and small and large individual heterogeneity sd (1, 5). For investigations of bias or estimability, it is often useful to analyse just a few, but very large data sets.

```
par(mfrow = c(2,2))
data1 <- data.fn(N = 1000, mean.p = 0.2, T = 3, sd = 1)
data2 <- data.fn(N = 1000, mean.p = 0.6, T = 3, sd = 1)
data3 <- data.fn(N = 1000, mean.p = 0.2, T = 3, sd = 5)
data4 <- data.fn(N = 1000, mean.p = 0.6, T = 3, sd = 5)
```



The graph shows a histogram of the individual detection probability for the low and the high sd cases (top and bottom rows) and for low and high mean detection probability (left and right columns).

Now we fit model M0 to each data set. We prepare the common elements of the analysis first and then adapt the variable part of the code to each data set.

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

# Priors
omega ~ dunif(0, 1)
p ~ dunif(0, 1)

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

```
   z[i] ~ dbern(omega)            # Inclusion indicators
   for (j in 1:T){
      yaug[i,j] ~ dbern(p.eff[i,j])
      p.eff[i,j] <- z[i] * p      # Can only be detected if z=1
      } #j
   } #i

# Derived quantities
N <- sum(z[])
}
",fill = TRUE)
sink()

# Initial values
inits <- function() list(z = rep(1, nrow(yaug)), p = runif(1, 0, 1))

# Parameters monitored
params <- c("N", "p", "omega")

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

Here's the analysis for data set 1 (mean.p = 0.2, sd = 1).

```
# Augment data set and bundle data
nz <- 1000
yaug <- rbind(data1$yobs, array(0, dim = c(nz, data1$T)))
win.data <- list(yaug = yaug, M = nrow(yaug), T = ncol(yaug))

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

# Summarize posteriors
print(out1, dig = 2)
Inference for Bugs model at "model.txt", fit using WinBUGS,
 3 chains, each with 2500 iterations (first 500 discarded), n.thin = 2
 n.sims = 3000 iterations saved
            mean    sd    2.5%     25%     50%     75%   97.5% Rhat n.eff
N         772.49 38.34  703.00  745.00  770.00  798.00  854.02    1  1000
p           0.29  0.02    0.25    0.27    0.29    0.30    0.32    1  1300
omega       0.52  0.03    0.47    0.50    0.52    0.54    0.58    1  1400
deviance 2773.29 77.13 2626.97 2720.00 2772.00 2826.00 2931.02    1  1100
```

Here's the analysis for data set 2 (mean.p = 0.6, sd = 1).

```
# Augment data set and bundle data
nz <- 1000
yaug <- rbind(data2$yobs, array(0, dim = c(nz, data2$T)))
win.data <- list(yaug = yaug, M = nrow(yaug), T = ncol(yaug))

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

# Summarize posteriors
print(out2, dig = 2)
```

```
Inference for Bugs model at "model.txt", fit using WinBUGS,
 3 chains, each with 2500 iterations (first 500 discarded), n.thin = 2
 n.sims = 3000 iterations saved
            mean    sd    2.5%     25%     50%     75%   97.5% Rhat n.eff
N         925.12  8.09  910.00  919.00  925.00  930.00  942.02    1  3000
p           0.63  0.01    0.61    0.63    0.63    0.64    0.65    1  3000
omega       0.49  0.01    0.47    0.48    0.49    0.50    0.52    1  3000
deviance 3651.29 48.51 3560.00 3618.00 3650.00 3682.00 3752.10    1  3000
```

Here's the analysis for data set 3 (mean.p = 0.2, sd = 5).

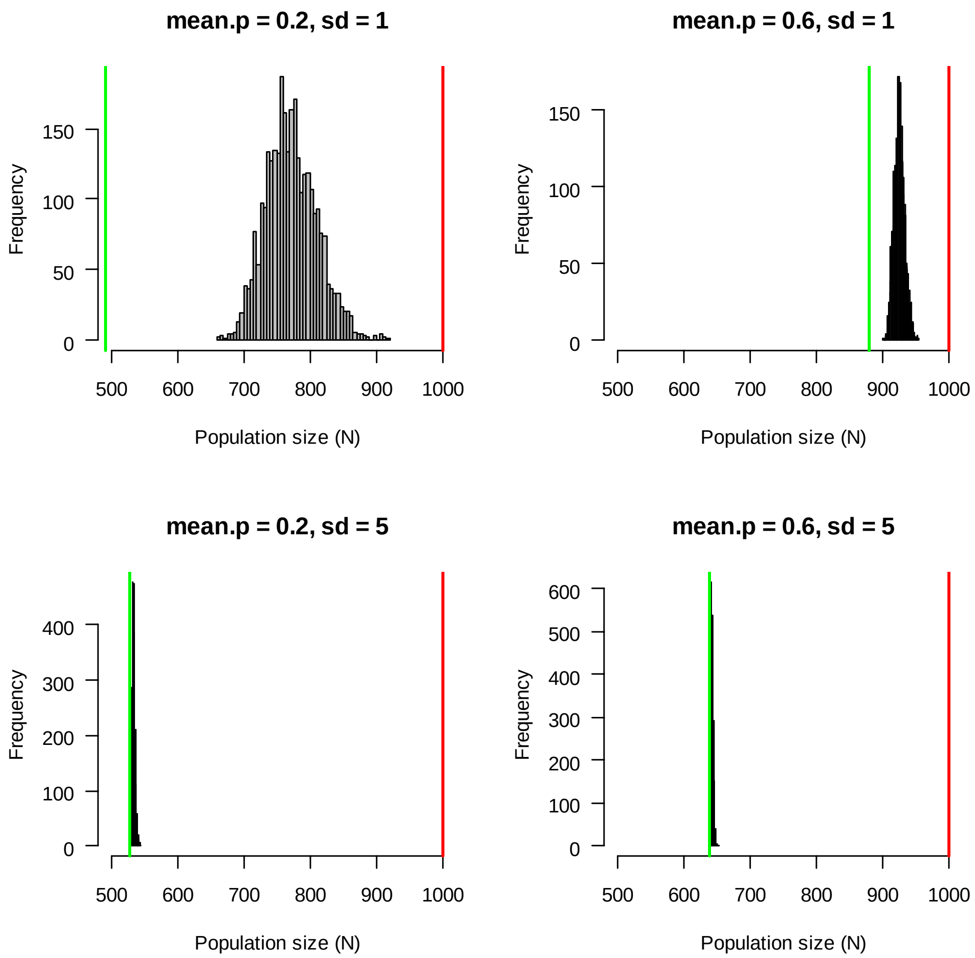
```
# Augment data set and bundle data
nz <- 1000
yaug <- rbind(data3$yobs, array(0, dim = c(nz, data3$T)))
win.data <- list(yaug = yaug, M = nrow(yaug), T = ncol(yaug))

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

# Summarize posteriors
print(out3, dig = 2)
Inference for Bugs model at "model.txt", fit using WinBUGS,
 3 chains, each with 2500 iterations (first 500 discarded), n.thin = 2
 n.sims = 3000 iterations saved
            mean    sd    2.5%     25%     50%     75%   97.5% Rhat n.eff
N         533.93  2.57  530.00  532.00  534.00  536.00  540.00    1  2500
p           0.78  0.01    0.76    0.77    0.78    0.78    0.80    1  3000
omega       0.35  0.01    0.33    0.34    0.35    0.36    0.37    1  2000
deviance 1697.32 23.25 1661.00 1680.00 1697.00 1715.00 1751.00    1  2700
```

And finally, here's the analysis for data set 4 (mean.p = 0.6, sd = 5).

```
# Augment data set and bundle data
nz <- 1000
yaug <- rbind(data4$yobs, array(0, dim = c(nz, data4$T)))
win.data <- list(yaug = yaug, M = nrow(yaug), T = ncol(yaug))

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

# Summarize posteriors
print(out4, dig = 2)
Inference for Bugs model at "model.txt", fit using WinBUGS,
 3 chains, each with 2500 iterations (first 500 discarded), n.thin = 2
 n.sims = 3000 iterations saved
            mean    sd    2.5%     25%     50%     75%   97.5% Rhat n.eff
N         641.85  2.05  638.00  640.00  642.00  643.00  646.00    1  3000
p           0.82  0.01    0.80    0.81    0.82    0.83    0.84    1  3000
omega       0.39  0.01    0.37    0.38    0.39    0.40    0.42    1  3000
deviance 1819.15 21.04 1780.00 1802.00 1820.00 1831.00 1863.02    1  3000
```

We see here an illustration of one of the 'laws' of capture-recapture: that unmodelled heterogeneity in detection probability (p) biases estimators of p high and consequently those of abundance N low. The degree of the underestimation in N depends, among other things, on the mean detection probability and on the magnitude of the individual heterogeneity in p, as you can see in the next figure.

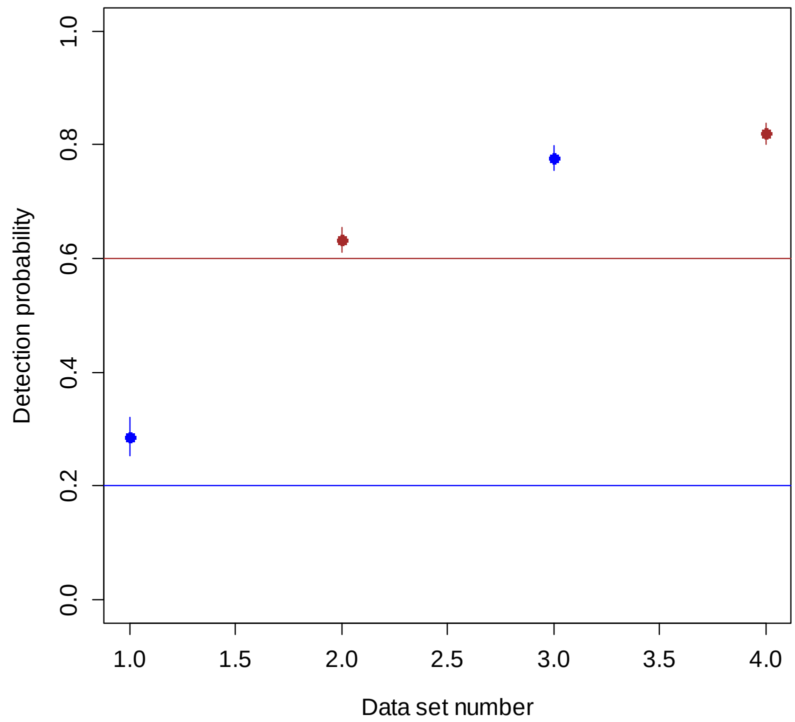
```
# Compare posteriors for N for 4 data sets
par(mfrow = c(2, 2))
hist(out1$sims.list$N, nclass = 50, col = "gray", main = "mean.p = 0.2, sd
= 1", xlab = "Population size (N)", las = 1, xlim = c(500, 1000))
abline(v = data1$C, lwd = 2, col = "green")
abline(v = 1000, lwd = 2, col = "red")
hist(out2$sims.list$N, nclass = 50, col = "gray", main = "mean.p = 0.6, sd
= 1", xlab = "Population size (N)", las = 1, xlim = c(500, 1000))
abline(v = data2$C, lwd = 2, col = "green")
abline(v = 1000, lwd = 2, col = "red")
hist(out3$sims.list$N, nclass = 50, col = "gray", main = "mean.p = 0.2, sd
= 5", xlab = "Population size (N)", las = 1, xlim = c(500, 1000))
abline(v = data3$C, lwd = 2, col = "green")
abline(v = 1000, lwd = 2, col = "red")
hist(out4$sims.list$N, nclass = 50, col = "gray", main = "mean.p = 0.6, sd
= 5", xlab = "Population size (N)", las = 1, xlim = c(500, 1000))
abline(v = data4$C, lwd = 2, col = "green")
abline(v = 1000, lwd = 2, col = "red")
```



In the figure, the observed number of individuals is shown in green and the truth (N = 1000) in red. We see that the observed number of individuals increases with increasing mean.p, but decreases with increasing individual heterogeneity in p (=sd). The bias in the estimator of N is more negative with small mean.p and large sd.

Interestingly, the posterior distributions of *N* in the analyses of the four data sets have widely different spread (i.e., uncertainty). This has to do with the estimate of detection probability, which increases from data set 1 through 4, as can be seen in the following graph.

```
plot(1:4, c(out1$mean$p, out2$mean$p, out3$mean$p, out4$mean$p), ylim =
c(0, 1), col = c("blue", "brown", "blue", "brown"), pch = 16, cex = 1.2,
xlab = "Data set number", ylab = "Detection probability", cex.lab = 1.2,
cex.axis = 1.2)
abline(h = c(0.2, 0.6), col = c("blue", "brown"))
segments(1, out1$summary[2,3], 1, out1$summary[2,7], col = "blue")
segments(2, out2$summary[2,3], 2, out2$summary[2,7], col = "brown")
segments(3, out3$summary[2,3], 3, out3$summary[2,7], col = "blue")
segments(4, out4$summary[2,3], 4, out4$summary[2,7], col = "brown")
```



The lines give the true values and the color indicates which estimate belongs to which value of the generating parameter.

Exercise 5

Task: Find out whether a model with trap response and time effects is estimable with T=2.

Solution: As for questions about bias, for practical matters, it is often sufficient to study questions about estimability by examining one large data set. So here, as a quick and dirty, though of course not mathematically waterproof, way to answer the question, we first simulate a few large data sets with T = 2 and with both effects present. Then, we fit the model in question and see whether we recover estimates that resemble the values chosen in the data simulation.

We will first adapt the data-generating function from section 6.2.3 to include both a behavioural and a time effect for T = 2. Normally, it would be convenient to combine effects on some transformed scale, for instance on the logit scale. Here, we don't do this, but then care has to be taken when chosing the numerical function arguments to avoid that they combine to probabilities outside of the permitted range (0, 1). Also, initial values must be chosen carefully; otherwise WinBUGS may crash immediately.

The three function arguments (apart from N, which of course is population size) are detection probability during the first occasion (p1) and the additive time effect (deltaT2), which is expressed as a *difference* in detection during the second relative to the first occasion. The behavioural effect of a capture event during occasion 1 on the detection probability during occasion 2 is called c and is *also expressed as a difference* (this is different from how we parameterized the trap response in the book).

As an example, (p1 = 0.3, c = 0.2, deltaT2 = 0.4) implies p1 = 0.3 during the first occasion and p2 = 0.9 and p2 = 0.7 on the second occasion, depending on whether the animal was or was not captured during the first occasion.

```
# Define function to simulate data under Mbt with T = 2 fixed
data.fn <- function(N = 100, p1 = 0.3, c = 0.2, deltaT2 = 0.4){
   yfull <- yobs <- array(NA, dim = c(N, 2) )

   # First capture occasion
   yfull[,1] <- rbinom(n = N, size = 1, prob = p1)

   # Second capture occasions
   p2 <- p1 + deltaT2 + yfull[,1]*c
   yfull[,2] <- rbinom(n = N, size = 1, prob = p2)

   ever.detected <- apply(yfull, 1, max)
   C <- sum(ever.detected)
   yobs <- yfull[ever.detected == 1,]
   cat(C, "out of", N, "animals present were detected.\n")
   return(list(N = N, p1 = p1, p2 = p2, c = c, deltaT2 = deltaT2, C = C, T
= 2, yfull = yfull, yobs = yobs))
   }
```

Do a trial run first:

```
str(data <- data.fn(N = 200, p1 = 0.3, c = 0.2, deltaT2 = 0.4))
167 out of 200 animals present were detected.
List of 9
 $ N      : num 200
 $ p1     : num 0.3
 $ p2     : num [1:200] 0.9 0.7 0.7 0.7 0.7 0.9 0.7 0.7 0.7 0.7 ...
 $ c      : num 0.2
 $ deltaT2: num 0.4
 $ C      : num 167
 $ T      : num 2
 $ yfull  : num [1:200, 1:2] 1 0 0 0 0 1 0 0 0 0 ...
 $ yobs   : num [1:167, 1:2] 1 0 0 0 1 0 0 1 0 1 ...
```

We define the model with both a time and a trap-response effect and also define initial values, parameters to save and MCMC settings.

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

```
omega ~ dunif(0, 1)
p1 ~ dunif(0, 1)      # Cap prob during 1st occasion
c ~ dunif(-1, 1)      # Diff. in cap.prob. during 2nd occasion
                      # if captured during 1st
deltaT2 ~ dunif(-1, 1)# Diff. in cap.prob. during 2nd occasion

# Likelihood
for (i in 1:M){
   z[i] ~ dbern(omega)

   # First occasion
   yaug[i,1] ~ dbern(p1.eff[i,1])
   p1.eff[i,1] <- z[i] * p1

   # Second occasion
   yaug[i,2] ~ dbern(p2.eff[i,2])
   p2.eff[i,2] <- z[i] * (p1 + deltaT2 + yaug[i,1] * c)
   } #i

# Derived quantities
N <- sum(z[])
} # end model
",fill = TRUE)
sink()

# Initial values (chose random starts for z, but fix for params)
inits <- function() list(z = round(runif(nrow(yaug), 0, 1)),
p1 = 0.5, c = 0.1, deltaT2 = 0.2)

# Parameters monitored
params <- c("N", "p1", "c", "deltaT2", "omega")

# MCMC settings
ni <- 4000
nt <- 1
nb <- 3000
nc <- 3
```

Now generate a few large data sets, fit the model and compare estimates and input values.

```
data <- data.fn(N = 1000, p1 = 0.3, c = 0.2, deltaT2 = 0.4)

# Augment data set and bundle data
nz <- 500
yaug <- rbind(data$yobs, array(0, dim = c(nz, data$T)))
win.data <- list(yaug = yaug, M = nrow(yaug))

# TRY initials right on spot (chose random starts for z)
inits <- function() list(z = round(runif(nrow(yaug), 0, 1)),
p1 = 0.4, c = 0.1, deltaT2 = 0.2)

# MCMC settings
ni <- 43000
nt <- 4
nb <- 3000
nc <- 3


# Call WinBUGS from R (BRT 1 min)
out <- bugs(win.data, inits, params, "model.txt", n.chains = nc,
```

```
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory =
bugs.dir, working.directory = getwd())

# Summarize posteriors
print(out, dig = 3)
Inference for Bugs model at "model.txt", fit using WinBUGS,
 3 chains, each with 43000 iterations (first 3000 discarded), n.thin = 4
 n.sims = 30000 iterations saved
            mean      sd    2.5%     25%     50%     75%    97.5%  Rhat n.eff
N        778.308  40.972 772.000 772.000 772.000 772.000  853.025 1.158   120
p1         0.494   0.036   0.361   0.487   0.499   0.511    0.534 1.105   130
c         -0.564   0.143  -0.638  -0.606  -0.589  -0.572    0.057 1.123   110
deltaT2    0.976   0.109   0.489   0.995   0.998   0.999    1.000 1.138   110
omega      0.612   0.035   0.580   0.598   0.607   0.617    0.671 1.110   160
deviance 928.319 233.064 883.100 884.600 886.100 888.300 1764.100 1.134   110
```

This does not look right. Indeed, it turns out that model M_{tb} does not have estimable parameters except under certain assumptions, but even then, we need data from at least three occasions (Otis et al. 1978: p. 111).

Exercise 6

Task: And what about pure model M_b with T=2?

Solution: We create a couple of large data sets under model M_b and fit model M_b. We directly use the functions provided in section 6.2.3, which we assume you have defined in your R workspace.

Example 1:

```
data <- data.fn(N = 10000, T = 2, p = 0.3, c = 0.4)
5138 out of 10000 animals present were detected.
```

Fitting the model (after setting nz=6000 and waiting for 6 mins) yields these estimates (we could run the chains longer to bring down those values of Rhat, but won't bother for now):

```
> print(out, dig = 2)
Inference for Bugs model at "model.txt", fit using WinBUGS,
 3 chains, each with 2500 iterations (first 500 discarded), n.thin = 2
 n.sims = 3000 iterations saved
                  mean     sd    2.5%      25%      50%      75%    97.5% Rhat n.eff
N             10144.75 530.18 9152.97  9775.00 10130.00 10580.00 11010.00 1.15    19
p                 0.30   0.02    0.27     0.28     0.30     0.31     0.34 1.15    19
c                 0.40   0.01    0.38     0.39     0.40     0.40     0.42 1.00  3000
trap.response     0.10   0.02    0.06     0.09     0.10     0.12     0.14 1.13    23
omega             0.91   0.05    0.82     0.88     0.91     0.95     0.99 1.15    19
deviance      25063.66 757.40 23580.00 24557.50 25070.00 25680.00 26240.00 1.15    19
```

Example 2:

```
data <- data.fn(N = 10000, T = 2, p = 0.6, c = 0.2)
8381 out of 10000 animals present were detected.
```

Fitting the model (after setting nz=3000 and waiting for 6 mins) yields these estimates:

```
> print(out, dig = 2)
Inference for Bugs model at "model.txt", fit using WinBUGS,
 3 chains, each with 2500 iterations (first 500 discarded), n.thin = 2
 n.sims = 3000 iterations saved
                  mean     sd    2.5%      25%      50%      75%    97.5% Rhat n.eff
N              9993.60  99.81 9808.97  9923.75  9989.00 10060.00 10200.00    1   510
p                 0.60   0.01    0.58     0.59     0.60     0.61     0.62    1   810
```

```
c                 0.20   0.01     0.19     0.19     0.20     0.20     0.21    1  1500
trap.response    -0.40   0.01    -0.42    -0.41    -0.40    -0.39    -0.38    1   760
omega             0.88   0.01     0.86     0.87     0.88     0.88     0.90    1   620
deviance      24792.97 362.63 24110.00 24540.00 24780.00 25040.00 25550.00    1   510
```

Example 3:

```
data <- data.fn(N = 10000, T = 2, p = 0.1, c = 0.9)
1927 out of 10000 animals present were detected.
```

Fitting the model (after setting nz=10000 and waiting during a short coffee break) yields these estimates (same comments on Rhat):

```
> print(out, dig = 2)
Inference for Bugs model at "model.txt", fit using WinBUGS,
 3 chains, each with 2500 iterations (first 500 discarded), n.thin = 2
 n.sims = 3000 iterations saved
                  mean     sd     2.5%      25%      50%      75%    97.5% Rhat n.eff
N             10936.61 592.89  9447.95 10680.00 11040.00 11380.00 11740.00 1.51     8
p                 0.09   0.01     0.08     0.09     0.09     0.10     0.11 1.44     9
c                 0.91   0.01     0.89     0.90     0.91     0.92     0.93 1.00  1700
trap.response     0.82   0.01     0.80     0.81     0.82     0.83     0.84 1.11    27
omega             0.92   0.05     0.79     0.90     0.93     0.95     0.98 1.50     8
deviance      13452.04 238.20 12830.00 13360.00 13500.00 13630.00 13760.00 1.51     8
```

These three examples strongly suggest that in the absence of time-dependence in *p*, a model with purely behavioural effect is estimable with T = 2 (of course, this is not a mathematical proof).

Exercise 7

Task: In M_t, adapt both the data generation and the model fitting code to random instead of fixed time effects.

Solution: In the data-generating function we draw T random time effects from a normal distribution, whose parameters we must define. To avoid trouble with non-admissible values for *p*, we do that on the logit-scale. We give the mean of the distribution on the probability scale and then transform that. sd.lp is the standard deviation of the distribution of logit(*p*).

```
# Define function to simulate data under Mt with random time effects
data.fn <- function(N = 100, mean.p = 0.5, T = 3, sd.lp = 1){
   yfull <- yobs <- array(NA, dim = c(N, T))
   mean.lp <- log(mean.p / (1 - mean.p))
   p.vec <- plogis(rnorm(T, mean.lp, sd.lp))   # Draw p for each T
   for (j in 1:T){
      yfull[,j] <- rbinom(n = N, size = 1, prob = p.vec[j])
      }
   ever.detected <- apply(yfull, 1, max)
   C <- sum(ever.detected)
   yobs <- yfull[ever.detected == 1,]
   cat(C, "out of", N, "animals present were detected.\n")
   return(list(N = N, p.vec = p.vec, C = C, T = T, yfull = yfull, yobs =
yobs, mean.p = mean.p, mean.lp = mean.lp, sd.lp = sd.lp))
   }
```

Try out the new function – it seems to work fine.

```
str(data <- data.fn(N = 100, mean.p = 0.3, T = 3, sd.lp = 1))
45 out of 100 animals present were detected.
List of 9
```

```
 $ N      : num 100
 $ p.vec  : num [1:3] 0.1411 0.3121 0.0957
 $ C      : num 45
 $ T      : num 3
 $ yfull  : num [1:100, 1:3] 0 0 0 0 0 0 0 0 0 0 ...
 $ yobs   : num [1:45, 1:3] 0 0 0 0 0 0 0 0 1 1 ...
 $ mean.p : num 0.3
 $ mean.lp: num -0.847
 $ sd.lp  : num 1
```

Same with more occasions:

```
str(data <- data.fn(N = 100, mean.p = 0.1, T = 10, sd.lp = 0.5))
81 out of 100 animals present were detected.
List of 9
 $ N      : num 100
 $ p.vec  : num [1:10] 0.068 0.16 0.139 0.164 0.15 ...
 $ C      : num 81
 $ T      : num 10
 $ yfull  : num [1:100, 1:10] 0 0 0 0 0 0 0 0 0 0 ...
 $ yobs   : num [1:81, 1:10] 0 0 0 0 0 0 0 0 0 1 ...
 $ mean.p : num 0.1
 $ mean.lp: num -2.2
 $ sd.lp  : num 0.5
```

We augment the latest data set ...

```
# Augment data set
nz <- 150
yaug <- rbind(data$yobs, array(0, dim = c(nz, data$T)))
```

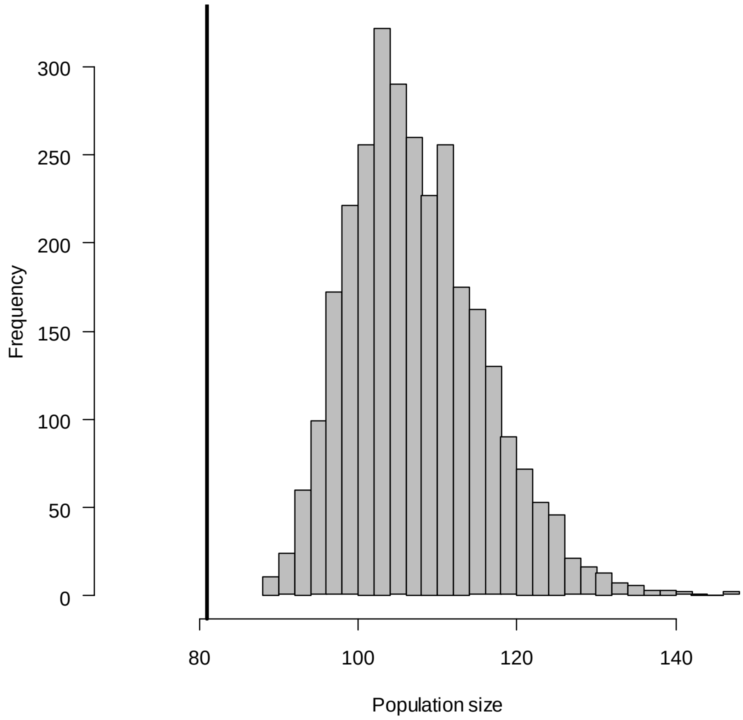
Redefine the model to have random time effects ...

```
# Specify model in BUGS language
sink("model.txt")
cat("
model {
# Priors
omega ~ dunif(0, 1)
mean.lp ~ dnorm(0, 0.001)
tau.lp <- pow(sd.lp, -2)
sd.lp ~ dunif(0, 3)

# Random effects distribution taken outside of loop below, to avoid
# multiple definitions of p[j]
for (j in 1:T){
   logit(p[j]) <- lp[j]
   lp[j] ~ dnorm(mean.lp, tau.lp)I(-16, 16)
   } #j

# Likelihood
for (i in 1:M){
   z[i] ~ dbern(omega)
   for (j in 1:T){
      yaug[i,j] ~ dbern(p.eff[i,j])
      p.eff[i,j] <- z[i] * p[j]
      } #j
   } #i

# Derived quantities
N <- sum(z[])
} # end model
```

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

# Bundle data
win.data <- list(yaug = yaug, M = nrow(yaug), T = ncol(yaug))

# Initial values
inits <- function() list(z = rep(1, nrow(yaug)), mean.lp = 0, sd.lp =
runif(1, 0, 1))

# Parameters monitored
params <- c("N", "p", "mean.lp", "sd.lp", "omega")

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

... and let it run.

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

# Summarize posteriors in table and graphs
print(out, dig = 2)
Inference for Bugs model at "model.txt", fit using WinBUGS,
 3 chains, each with 2500 iterations (first 500 discarded), n.thin = 2
 n.sims = 3000 iterations saved
           mean    sd   2.5%    25%    50%    75%  97.5% Rhat n.eff
N        107.89  8.51  94.00 102.00 107.00 113.00 126.00    1  1300
p[1]       0.05  0.02   0.02   0.03   0.04   0.06   0.09    1  3000
p[2]       0.10  0.03   0.06   0.08   0.10   0.12   0.17    1  3000
p[3]       0.10  0.03   0.05   0.08   0.10   0.11   0.16    1  3000
p[4]       0.11  0.03   0.06   0.09   0.11   0.13   0.18    1  1400
p[5]       0.10  0.03   0.05   0.08   0.09   0.11   0.16    1  3000
p[6]       0.26  0.05   0.18   0.23   0.26   0.29   0.36    1  1600
p[7]       0.09  0.03   0.04   0.07   0.09   0.11   0.14    1  2100
p[8]       0.11  0.03   0.06   0.09   0.11   0.13   0.17    1   550
p[9]       0.27  0.05   0.18   0.24   0.27   0.30   0.36    1   780
p[10]      0.10  0.03   0.05   0.08   0.10   0.12   0.16    1  3000
mean.lp   -2.07  0.30  -2.66  -2.27  -2.07  -1.89  -1.48    1  3000
sd.lp      0.81  0.28   0.42   0.61   0.75   0.94   1.53    1  2100
omega      0.47  0.05   0.38   0.43   0.47   0.50   0.57    1  3000
deviance 776.70 24.16 734.29 759.37 774.95 792.60 827.00    1  1200

# Observed value of N and estimate
hist(out$sims.list$N, nclass = 40, col = "gray", main = "", xlab =
"Population size", las = 1, xlim = c(70, 150))
abline(v = data$C, col = "black", lwd = 3)
```




```
# True and estimated values of the random time effects (p[])
cbind(data$p.vec, out$summary[2:11,c(1:3, 7)])
                         mean         sd       2.5%      97.5%
p[1]  0.06801732 0.04560702 0.01934555 0.01536598 0.0910635
p[2]  0.16002862 0.10434604 0.02791412 0.05635775 0.1665075
p[3]  0.13857382 0.09785143 0.02839731 0.05141850 0.1619025
p[4]  0.16376635 0.11137707 0.02962492 0.06029625 0.1768000
p[5]  0.14956783 0.09678913 0.02734596 0.05155925 0.1575025
p[6]  0.22180096 0.26184623 0.04658270 0.17689500 0.3567125
p[7]  0.07407098 0.08900841 0.02588661 0.04478800 0.1436000
p[8]  0.13179242 0.11280878 0.02852425 0.06219000 0.1735150
p[9]  0.31775770 0.26802573 0.04636472 0.18337996 0.3618075
p[10] 0.13010083 0.09814712 0.02737744 0.05204725 0.1562025
```

The estimates match reasonably well the true values.

Exercise 8

Task: Check the effects of assumption violations. Fit a model to a data set that was not generated under the same model. For instance, generate data under model M_t and analyze the resulting data set under M_0 to see what happens to your estimates of *N* and *p* when you ignore time variation in *p*. Do similar things to other pairs of models.

Solution: We will restrict attention here to the case of M_t and M_0. This exercise is fairly similar to exercise 4, except that now we generate a data set under model M_t and fit model M_0, while there, the data-generation was under M_h. We directly use the code for generating data under the random-effects model M_t (from the previous exercise) and for model fitting under M_0 from the BPA book.

```
# Data generation from Exercise 7
data.fn <- function(N = 100, mean.p = 0.5, T = 3, sd.lp = 1){
   yfull <- yobs <- array(NA, dim = c(N, T))
   mean.lp <- log(mean.p / (1 - mean.p))
   p.vec <- plogis(rnorm(T, mean.lp, sd.lp))   # Draw p for each T
   for (j in 1:T){
      yfull[,j] <- rbinom(n = N, size = 1, prob = p.vec[j])
      }
   ever.detected <- apply(yfull, 1, max)
   C <- sum(ever.detected)
   yobs <- yfull[ever.detected == 1,]
   cat(C, "out of", N, "animals present were detected.\n")
   return(list(N = N, p.vec = p.vec, C = C, T = T, yfull = yfull,
yobs = yobs, mean.p = mean.p, mean.lp = mean.lp, sd.lp = sd.lp))
   }
```

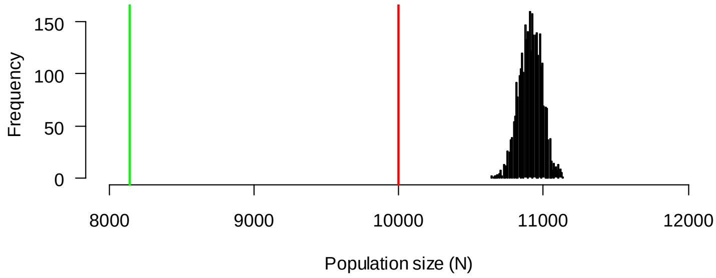
Generate one large data set. By the way, we generate large data sets, because then the effects of sampling variation are minimised and bias (or parameter estimability) can be seen most clearly.

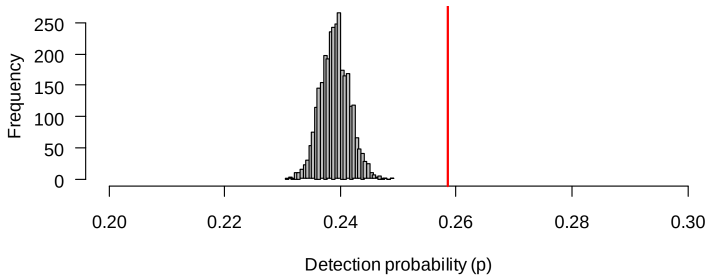
```
str(data <- data.fn(N = 10000, mean.p = 0.2, T = 5, sd.lp = 1))
8139 out of 10000 animals present were detected.
List of 9
 $ N      : num 10000
 $ p.vec  : num [1:5] 0.257 0.1241 0.3184 0.5457 0.0473
 $ C      : num 8139
 $ T      : num 5
 $ yfull  : num [1:10000, 1:5] 0 0 1 1 0 0 1 0 0 0 ...
 $ yobs   : num [1:8139, 1:5] 0 1 1 0 0 1 0 0 0 0 ...
 $ mean.p : num 0.2
 $ mean.lp: num -1.39
 $ sd.lp  : num 1
```

No we fit model M_0 to this data set.

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

# Priors
omega ~ dunif(0, 1)
p ~ dunif(0, 1)

# Likelihood
for (i in 1:M){
   z[i] ~ dbern(omega)
   for (j in 1:T){
      yaug[i,j] ~ dbern(p.eff[i,j])
      p.eff[i,j] <- z[i] * p
      } #j
   } #i

# Derived quantities
N <- sum(z[])
}
",fill = TRUE)
```

```
sink()

# Initial values
inits <- function() list(z = rep(1, nrow(yaug)), p = runif(1, 0, 1))

# Parameters monitored
params <- c("N", "p", "omega")

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

# Augment data set and bundle data
nz <- 3000
yaug <- rbind(data$yobs, array(0, dim = c(nz, data$T)))
win.data <- list(yaug = yaug, M = nrow(yaug), T = ncol(yaug))

# Call WinBUGS from R (BRT 15 min)
out <- bugs(win.data, inits, params, "model.txt", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE,
bugs.directory = bugs.dir, working.directory = getwd())

# Summarize posteriors in table and sketch them in graphs
print(out, dig = 2)
Inference for Bugs model at "model.txt", fit using WinBUGS,
 3 chains, each with 2500 iterations (first 500 discarded), n.thin = 2
 n.sims = 3000 iterations saved
          mean    sd     2.5%      25%      50%      75%    97.5% Rhat n.eff
N     10915.84 78.03 10760.00 10860.00 10920.00 10970.00 11060.00    1  3000
p         0.24  0.00     0.23     0.24     0.24     0.24     0.24    1  1800
omega     0.98  0.01     0.97     0.98     0.98     0.98     0.99    1  2300

par(mfrow = c(2, 1))
hist(out$sims.list$N, nclass = 50, col = "gray", main = "", xlab =
"Population size (N)", las = 1, xlim = c(8000, 12000))
abline(v = data$C, lwd = 2, col = "green")
abline(v = 10000, lwd = 2, col = "red")

hist(out$sims.list$p, nclass = 50, col = "gray", main = "", xlab =
"Detection probability (p)", las = 1, xlim = c(0.2, 0.3))
#abline(v = data$mean.p, lwd = 2, col = "green")
abline(v = mean(data$p.vec), lwd = 2, col = "red")
```



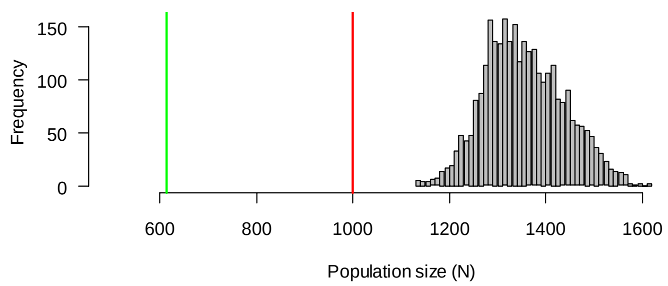


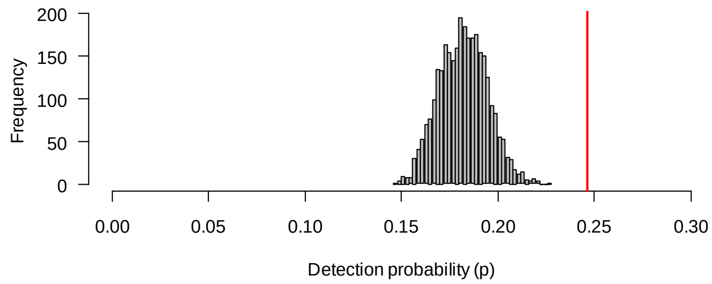
We see that unmodeled temporal heterogeneity in *p* leads to a positive bias in the estimator of *N* and a negative one in that for *p*. The green line is the observed number of individuals and the red line (top panel) true *N*, while in the bottom panel it is the mean of the true temporal *p*'s.

We repeat the exercise for another data set to make the generality of this conclusion more likely. We choose a smaller *N*, larger mean.p, smaller T and the same sd.lp. We don't need so long MC chains.

```
str(data <- data.fn(N = 1000, mean.p = 0.4, T = 3, sd.lp = 1))
612 out of 1000 animals present were detected.
List of 9
 $ N       : num 1000
 $ p.vec   : num [1:3] 0.137 0.102 0.501
 $ C       : num 612
 $ T       : num 3
 $ yfull   : num [1:1000, 1:3] 0 0 0 0 0 0 0 0 0 0 ...
 $ yobs    : num [1:612, 1:3] 0 0 0 0 0 1 0 0 0 0 ...
 $ mean.p  : num 0.4
 $ mean.lp: num -0.405
 $ sd.lp   : num 1

# MCMC settings
ni <- 1200
nt <- 1
nb <- 200
nc <- 3

# Augment data set and bundle data
nz <- 1500
yaug <- rbind(data$yobs, array(0, dim = c(nz, data$T)))
win.data <- list(yaug = yaug, M = nrow(yaug), T = ncol(yaug))
```

```
# Call WinBUGS from R (BRT 15 min)
out <- bugs(win.data, inits, params, "model.txt", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE,
bugs.directory = bugs.dir, working.directory = getwd())

# Summarize posteriors in table and sketch them in graphs
print(out, dig = 2)
Inference for Bugs model at "model.txt", fit using WinBUGS,
 3 chains, each with 1200 iterations (first 200 discarded)
 n.sims = 3000 iterations saved
            mean    sd    2.5%     25%     50%     75%   97.5% Rhat n.eff
N        1356.08 82.21 1210.00 1295.00 1349.00 1413.25 1522.00 1.02   180
p           0.18  0.01    0.16    0.17    0.18    0.19    0.21 1.01   250
omega       0.64  0.04    0.57    0.61    0.64    0.67    0.72 1.02   190
deviance 3853.49 98.59 3669.97 3781.00 3848.00 3924.00 4043.02 1.02   180

par(mfrow = c(2, 1))
hist(out$sims.list$N, nclass = 50, col = "gray", main = "", xlab =
"Population size (N)", las = 1, xlim = c(500, 1700))
abline(v = data$C, lwd = 2, col = "green")
abline(v = 1000, lwd = 2, col = "red")

hist(out$sims.list$p, nclass = 50, col = "gray", main = "", xlab =
"Detection probability (p)", las = 1, xlim = c(0.0, 0.3))
#abline(v = data$mean.p, lwd = 2, col = "green")
abline(v = mean(data$p.vec), lwd = 2, col = "red")
```





Same picture here; so it appears that unmodelled temporal heterogeneity in *p* causes a negative bias in *p* and consequently a positive bias in *N*.

<u>Exercise 9</u>

Task: Use the Czech point count data and estimate species richness, where detection is a function of body mass, similar as in section 6.4.1. But this time, include the body mass of all unobserved species. Hint: you then no longer have to give a prior for body mass. Does the estimate of population size and of detection probability become more precise?

Solution: In this analysis, we exploit the fact that we really *know* the distribution of body masses in the entire community: it's the known body masses of the 146 species. So we no longer estimate the parameters of that distribution and therefore should get more precise estimates. Let's try that out.

```
p610 <- read.table("p610.txt", header = TRUE)
y <- as.matrix(p610[,5:9])             # Grab counts and convert to matrix
y[y > 1] <- 1                          # Convert to det-nondetections
dimnames(y) <- NULL
```

We use the full, 'naturally augmented' data set comprising all 146 Czech species, so we no longer need to augment the data set. We still take the log of body weight, and center it for the analysis.

```
# Specify model in BUGS language
sink("M_t+X.txt")
cat("
model {

# Priors
omega ~ dunif(0, 1)
for (j in 1:T){
   alpha[j] <- log(mean.p[j] / (1-mean.p[j]))
   mean.p[j] ~ dunif(0, 1)
   }
beta ~ dnorm(0, 0.01)

# Likelihood
for (i in 1:Nspec){  # Loop over individuals
   z[i] ~ dbern(omega)
   for (j in 1:T){  # Loop over occasions
      y[i,j] ~ dbern(p.eff[i,j])
      p.eff[i,j] <- z[i] * p[i,j]
      p[i,j] <- 1 / (1 + exp(-lp[i,j]))
      lp[i,j] <- alpha[j] + beta * size[i]
      } #j
   } #i

# Derived quantities
N <- sum(z[])
}
",fill = TRUE)
sink()

# Bundle data
win.data <- list(y = y, size = log(p610$bm)-mean(log(p610$bm)), Nspec =
nrow(y), T = ncol(y))

# Initial values
inits <- function() list(z = rep(1, nrow(y)), beta = runif(1, 0, 1))

# Parameters monitored
params <- c("N", "mean.p", "beta", "omega")
```

```
# MCMC settings
ni <- 5000
nt <- 2
nb <- 1000
nc <- 3

# Call WinBUGS from R (BRT 1 min)
outX2 <- bugs(win.data, inits, params, "M_t+X.txt", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory =
bugs.dir, working.directory = getwd())

# Summarize posteriors and plot posterior for N
print(outX2, dig = 3)
Inference for Bugs model at "M_t+X.txt", fit using WinBUGS,
 3 chains, each with 5000 iterations (first 1000 discarded), n.thin = 2
 n.sims = 6000 iterations saved
             mean     sd   2.5%    25%    50%    75%  97.5%  Rhat n.eff
N          46.579  5.842 37.000 42.000 46.000 50.000 59.000 1.001  6000
mean.p[1]   0.190  0.061  0.091  0.146  0.184  0.228  0.328 1.001  4600
mean.p[2]   0.231  0.069  0.117  0.181  0.224  0.273  0.385 1.001  6000
mean.p[3]   0.230  0.069  0.116  0.180  0.224  0.275  0.378 1.001  3700
mean.p[4]   0.172  0.058  0.078  0.130  0.166  0.206  0.303 1.001  6000
mean.p[5]   0.252  0.072  0.128  0.201  0.247  0.298  0.404 1.001  6000
beta       -0.578  0.177 -0.914 -0.699 -0.579 -0.460 -0.233 1.002  2200
 [ ... ]
```

For comparison, here's the analysis from the book, where the body mass distribution is estimated.

```
             mean     sd   2.5%    25%    50%    75%  97.5%  Rhat n.eff
N          41.638 10.384 32.000 36.000 39.000 44.000 68.000 1.099    65
mean.p[1]   0.270  0.075  0.138  0.216  0.264  0.318  0.430 1.003  1100
mean.p[2]   0.320  0.080  0.176  0.263  0.316  0.373  0.488 1.003  1200
mean.p[3]   0.321  0.080  0.175  0.263  0.317  0.374  0.487 1.003  1100
mean.p[4]   0.244  0.072  0.120  0.192  0.239  0.290  0.399 1.003  1200
mean.p[5]   0.345  0.083  0.197  0.287  0.341  0.400  0.518 1.003  1000
beta       -1.313  0.875 -3.143 -1.873 -1.308 -0.725  0.346 1.061    40
omega       0.233  0.064  0.152  0.195  0.222  0.256  0.387 1.049   100
mu.size     0.070  0.111 -0.092  0.002  0.055  0.116  0.342 1.083    63
sd.size     0.366  0.065  0.272  0.323  0.356  0.398  0.520 1.019   230
```

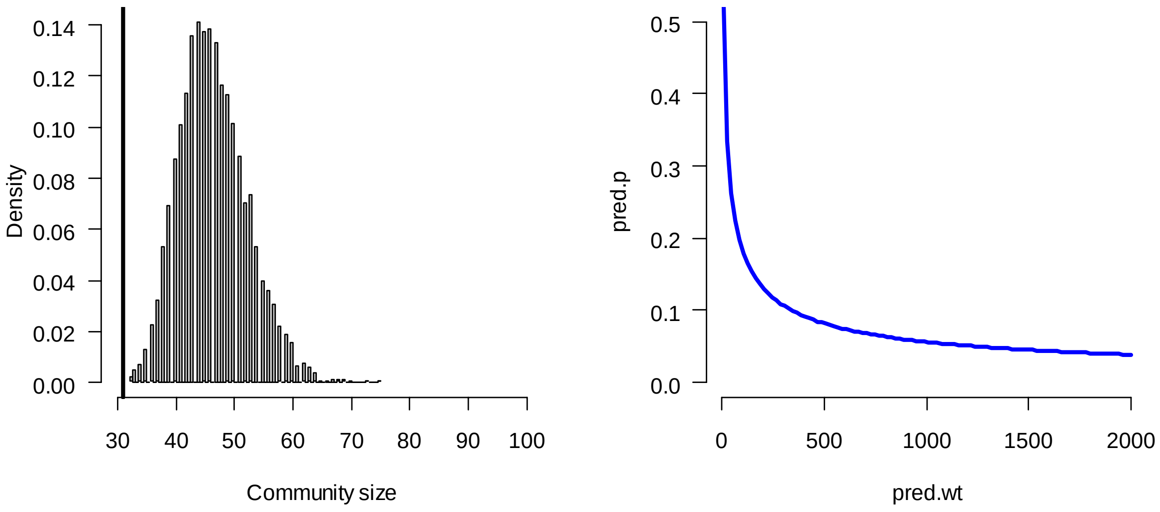
In the new analysis, the community is estimated to be comprised of about five more species; consequently, the point estimates of detection probability are lower. The slope of the regression of detection probability on body mass is also different, but we can't compare the slope directly, because we used a different transformation of body mass.

As expected, the estimates in the new analysis are more precise. For instance, the %CV of the estimate of *N* is now about 12.5%, while in the old analysis it was 24.9%. Similarly, the %CV of the slope estimate (beta) is now 30.6%, while it was 66.6% before. So if we do have information about the individual covariate in individuals (here, species) not captured, then it pays directly using that information.

Finally, here's the pictures of the posterior of *N* and of the estimated relationship between detection probability and body mass under the new model.

```
par(mfrow = c(1, 2))
hist(outX2$sims.list$N, breaks = 100, col = "gray", main = "", xlab =
"Community size", las = 1, xlim = c(30, 100), freq = FALSE)
abline(v = 31, col = "black", lwd = 3)
pred.wt <- seq(5, 2000, length.out = 100)# Cov. vals for prediction
pred.wt.st <- log(pred.wt)- mean(log(p610$bm)) # Transform them in the same
was as in the analysis
```

```
pred.p<- plogis(log(mean(outX2$mean$mean.p)/(1- mean(outX2$mean$mean.p))) +
outX2$mean$beta * pred.wt.st) # Compute predicted response
plot(pred.wt, pred.p, type = "l", lwd = 3, col = "blue", las = 1,
frame.plot = FALSE, ylim = c(0, 0.5))
```



Chapter 7

Exercise 1

Task: 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. Take 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.

Solution: This requires a change in the likelihood part of the model. Instead of using `phi[i,t]` and `p[i,t]` which "define" for each individual and each time step a different survival and recapture probability, we use directly `mean.phi` and `mean.p`. We also have to specify appropriate priors for these two parameters.

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

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

This model can be used with the data simulated in section 7.3. No changes to the other ingredients of the analysis are necessary.

Exercise 2

Task: 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 0.5 and recapture is for both 0.4. Assume that each year 30 individuals of each sex are newly marked. Fit the model {ϕ_{sex}, *p*} to the data using the multinomial likelihood.

Solution: We first simulate capture-recapture data of males and females using the simulation function `simul.cjs`. We then create m-arrays from the two sets of capture-recapture data. Finally, we fit the CJS model with the multinomial likelihood. We here write a separate likelihood for the male and female data and use in both likelihoods the same parameter for the recapture probability.

Data simulation

```
# Define the parameters
n.occasions <- 15                    # Number of capture occasions
marked <- rep(30, n.occasions-1)     # Number of newly marked individuals
phi.m <- rep(0.6, n.occasions-1)
phi.f <- rep(0.5, n.occasions-1)
p <- rep(0.4, n.occasions-1)

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

# Apply simulation function
CH.M <- simul.cjs(PHI.M, P, marked)
CH.F <- simul.cjs(PHI.F, P, marked)

# Create the m-arrays from the capture-histories
marr.m <- marray(CH.M)
marr.f <- marray(CH.F)
```

Data analysis

```
# Specify model in BUGS language
sink("cjs-mnl-g.bug")
cat("
model {
# Priors and constraints
for (t in 1:(n.occasions-1)){
   phi.m[t] <- mean.phim
   phi.f[t] <- mean.phif
   p[t] <- mean.p
   }
mean.phim ~ dunif(0, 1)      # Prior for mean survival
mean.phif ~ dunif(0, 1)      # Prior for mean survival
mean.p ~ dunif(0, 1)         # Prior for mean recapture

# Define the multinomial likelihood
for (t in 1:(n.occasions-1)){
   marr.m[t,1:n.occasions] ~ dmulti(pr.m[t, ], r.m[t])
   marr.f[t,1:n.occasions] ~ dmulti(pr.f[t, ], r.f[t])
   }
# Calculate the number of birds released each year
for (t in 1:(n.occasions-1)){
   r.m[t] <- sum(marr.m[t, ])
   r.f[t] <- sum(marr.f[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.m[t,t] <- phi.m[t]*p[t]
   pr.f[t,t] <- phi.f[t]*p[t]
```

```
   # Above main diagonal
   for (j in (t+1):(n.occasions-1)){
      pr.m[t,j] <- prod(phi.m[t:j])*prod(q[t:(j-1)])*p[j]
      pr.f[t,j] <- prod(phi.f[t:j])*prod(q[t:(j-1)])*p[j]
     } #j
   # Below main diagonal
   for (j in 1:(t-1)){
      pr.m[t,j] <- 0
      pr.f[t,j] <- 0
      } #j
   } #t
# Last column: probability of non-recapture
for (t in 1:(n.occasions-1)){
   pr.m[t,n.occasions] <- 1-sum(pr.m[t,1:(n.occasions-1)])
   pr.f[t,n.occasions] <- 1-sum(pr.f[t,1:(n.occasions-1)])
   } #t
}
",fill = TRUE)
sink()


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

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

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

# MCMC settings
niter <- 10000
nthin <- 3
nburn <- 5000
nchains <- 3

# Call WinBUGS from R (BRT 0.8 min)
cjs <- bugs(bugs.data, inits, parameters, "cjs-mnl-g.bug", n.chains =
nchains, n.thin = nthin, n.iter = niter, n.burnin = nburn, debug = TRUE,
bugs.directory = bugs.dir, working.directory = getwd())


print(cjs, 3)
             mean    sd   2.5%    25%    50%    75%  97.5%  Rhat n.eff
mean.phim   0.615 0.021  0.575  0.601  0.616  0.629  0.656 1.001  5000
mean.phif   0.500 0.024  0.452  0.483  0.500  0.516  0.548 1.002  2000
mean.p      0.395 0.024  0.350  0.380  0.395  0.411  0.443 1.001  5000
```

Exercise 3

Task: 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.

Solution: We first simulate the data using the simulation function `simul.cjs`. Simulations are performed separately for each sex and age class. We then combine the four capture-recapture data sets to one data set and construct a variable indicating the sex of each individual and a matrix indicating the age of each individual and at each occasion. Finally, we analyse the data with the state-space likelihood. While the survival parameters are modelled on the real scale, the recapture probabilities are modelled on the logit scale because of the additive model that needs to be used. The logit scale ensures that estimates are between 0 and 1 when back-transformed to the real scale.

Data simulation

```
# Define the parameters
n.occasions <- 10                     # Number of recapture occasions
marked.j <- rep(30, n.occasions-1)    # Annual number of newly marked
juveniles
marked.a <- rep(20, n.occasions-1)    # Annual number of newly marked adults
phi.juvm <- 0.3                       # Juvenile annual survival of males
phi.juvf <- 0.2                       # Juvenile annual survival of females
phi.ad <- 0.65                        # Adult annual survival
p.m <- c(0.5, 0.6, 0.4, 0.4, 0.7, 0.5, 0.8, 0.3, 0.8)   # Recapture males
diff <- 0.3
p.f <- plogis(qlogis(p.m)+diff)
phi.jm <- c(phi.juvm, rep(phi.ad, n.occasions-2))
phi.jf <- c(phi.juvf, rep(phi.ad, n.occasions-2))
phi.a <- rep(phi.ad, n.occasions-1)

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

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

```
# Create vector with occasion of marking
get.first <- function(x) min(which(x!=0))
f.jm <- apply(CH.JM, 1, get.first)
f.am <- apply(CH.AM, 1, get.first)
f.jf <- apply(CH.JF, 1, get.first)
f.af <- apply(CH.AF, 1, get.first)

# Create matrices X indicating the age class
x.jm <- matrix(NA, ncol = dim(CH.JM)[2]-1, nrow = dim(CH.JM)[1])
x.am <- matrix(NA, ncol = dim(CH.AM)[2]-1, nrow = dim(CH.AM)[1])
x.jf <- matrix(NA, ncol = dim(CH.JF)[2]-1, nrow = dim(CH.JF)[1])
x.af <- matrix(NA, ncol = dim(CH.AF)[2]-1, nrow = dim(CH.AF)[1])

for (i in 1:dim(CH.JM)[1]){
   for (t in f.jm[i]:(dim(CH.JM)[2]-1)){
      x.jm[i,t] <- 2
      x.jm[i,f.jm[i]] <- 1
      } #t
   } #i
for (i in 1:dim(CH.AM)[1]){
   for (t in f.am[i]:(dim(CH.AM)[2]-1)){
      x.am[i,t] <- 2
      } #t
   } #i
for (i in 1:dim(CH.JF)[1]){
   for (t in f.jf[i]:(dim(CH.JF)[2]-1)){
      x.jf[i,t] <- 2
      x.jf[i,f.jf[i]] <- 1
      } #t
   } #i
for (i in 1:dim(CH.AF)[1]){
   for (t in f.af[i]:(dim(CH.AF)[2]-1)){
      x.af[i,t] <- 2
      } #t
   } #i

# Combine the data, and create group variable
CH <- rbind(CH.JM, CH.AM, CH.JF, CH.AF)
f <- c(f.jm, f.am, f.jf, f.af)
x <- rbind(x.jm, x.am, x.jf, x.af)
group <- c(rep(1, dim(CH.JM)[1]), rep(1, dim(CH.AM)[1]), rep(2,
dim(CH.JF)[1]), rep(2, dim(CH.AF)[1]))
```

Data analysis

```
# Specify model in BUGS language
sink("cjs-age2.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],group[i]]
      logit(p[i,t]) <- gamma[t] + delta[group[i]]
      } #t
   } #i
beta[1,1] ~ dunif(0, 1)           # Prior for juv survival of male
beta[1,2] ~ dunif(0, 1)           # Prior for juv survival of female
beta[2,1] ~ dunif(0, 1)           # Prior for ad survival of male
beta[2,2] <- beta[2,1]            # Ad survival of female identical to male

for (t in 1:(n.occasions-1)){
```

```
   gamma[t] ~ dnorm(0,0.001)I(-15,15)  # Prior for recapture of males
   }
delta[1] <- 0
delta[2] ~ dnorm(0, 0.001)I(-15,15)    # Prior for recapture (diff. between
sexes)

# Back-transformations
for (t in 1:(n.occasions-1)){
   p.m[t] <- 1/(1+exp(-gamma[t]))
   p.f[t] <- 1/(1+exp(-gamma[t]-delta[2]))
   }

# Likelihood
for (i in 1:nind){
   # Ensures that individuals enter the sample with probability 1
   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, group = group)

# Initial values
inits <- function(){list(z = cjs.init.z(CH, f), beta = matrix(c(runif(3, 0,
1), NA), ncol = 2, byrow = TRUE), gamma = rnorm(dim(CH)[2]-1), delta =
c(NA, rnorm(1)))}

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

# MCMC settings
niter <- 5000
nthin <- 3
nburn <- 3000
nchains <- 3

# Call WinBUGS from R (BRT 5 min)
cjs.age <- bugs(bugs.data, inits, parameters, "cjs-age2.bug", n.chains =
nchains, n.thin = nthin, n.iter = niter, n.burnin = nburn, debug = TRUE,
bugs.directory = bugs.dir, working.directory = getwd())

print(cjs.age, 3)
Inference for Bugs model at "cjs-age2.bug", fit using WinBUGS,
 3 chains, each with 5000 iterations (first 3000 discarded), n.thin = 3
 n.sims = 2001 iterations saved
            mean    sd  2.5%   25%   50%   75% 97.5%  Rhat n.eff
beta[1,1]  0.266 0.033 0.204 0.243 0.264 0.288 0.333 1.002  1100
beta[1,2]  0.206 0.028 0.152 0.185 0.205 0.225 0.262 1.002  1300
beta[2,1]  0.659 0.018 0.623 0.647 0.659 0.672 0.696 1.003   970
beta[2,2]  0.659 0.018 0.623 0.647 0.659 0.672 0.696 1.003   970
p.m[1]     0.425 0.083 0.273 0.367 0.424 0.481 0.597 1.001  2000
```

```
p.m[2]       0.624  0.071    0.480    0.576    0.625    0.673    0.758 1.002  1500
p.m[3]       0.440  0.059    0.328    0.399    0.438    0.480    0.559 1.001  2000
p.m[4]       0.418  0.058    0.308    0.378    0.417    0.456    0.534 1.004   560
p.m[5]       0.634  0.059    0.519    0.596    0.634    0.674    0.748 1.001  1800
p.m[6]       0.475  0.059    0.360    0.432    0.473    0.518    0.587 1.001  1800
p.m[7]       0.788  0.057    0.665    0.751    0.791    0.830    0.890 1.002  1300
p.m[8]       0.330  0.053    0.232    0.292    0.326    0.363    0.443 1.002   900
p.m[9]       0.758  0.103    0.577    0.687    0.748    0.817    1.000 1.004   930
p.f[1]       0.525  0.089    0.353    0.463    0.528    0.586    0.713 1.002  1000
p.f[2]       0.713  0.066    0.573    0.670    0.717    0.760    0.834 1.004   510
p.f[3]       0.542  0.060    0.425    0.500    0.542    0.583    0.663 1.001  2000
p.f[4]       0.519  0.061    0.404    0.476    0.519    0.561    0.638 1.001  2000
p.f[5]       0.723  0.052    0.614    0.691    0.726    0.757    0.819 1.001  2000
p.f[6]       0.577  0.056    0.463    0.540    0.578    0.615    0.692 1.001  2000
p.f[7]       0.848  0.044    0.753    0.820    0.852    0.878    0.922 1.001  2000
p.f[8]       0.426  0.055    0.321    0.388    0.424    0.461    0.540 1.000  2000
p.f[9]       0.824  0.078    0.680    0.773    0.820    0.869    1.000 1.003   810
deviance  1859.891 47.497 1757.000 1831.000 1863.000 1893.000 1944.000 1.000  2000
```

Models with additive effects are typically harder to get to convergence than models without additive effects. The use of a range restriction for the prior distributions of the parameters of additive effects (e.g. `delta[2] ~ dnorm(0, 0.001)I(-15,15)`) often helps a lot.

Exercise 4

Task: For the model in 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?

Solution: We first define the CJS model that is used to analyse the simulated data. Here we use the state-space formulation, but of course the multinomial likelihood could be used as well. Within a loop, we then simulate capture-recapture data using the function `simul.cjs`, fit the model, and store the estimated survival and recapture probability. The difference between the parameters used to simulate the data and the mean of the estimated survival or recapture probabilities is an estimate of the bias, and the standard deviation of the estimated survival and recapture probabilities is a measure of the precision of the estimator. Another way to check whether a model produces unbiased estimates is to analyse a data set with very large sample size.

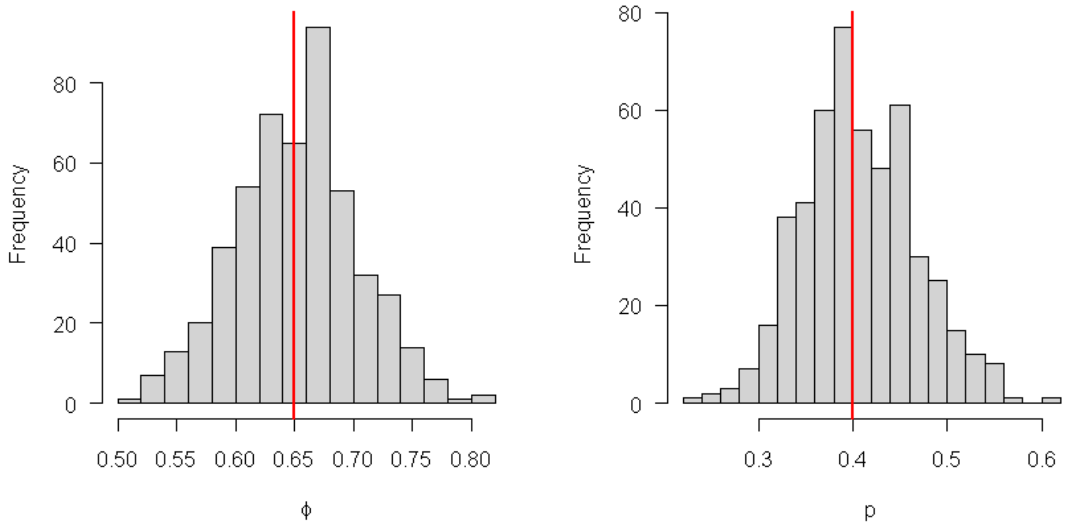
```
# Specify model in BUGS language
sink("cjs-c-c.bug")
cat("
model {

# 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

# Define the likelihood
for (i in 1:nind){
   # Ensures that individuals enter the sample with probability 1
   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()

# MCMC settings
niter <- 2000
nthin <- 1
nburn <- 1000
nchains <- 1

# Define the simulation parameters and data structures to store the output
nsim <- 500
phi.est <- p.est <- numeric()

# Start loop for the simulation
for (sim in 1:nsim){
   # Define the parameters
   n.occasions <- 6                  # Number of capture occasions
   marked <- rep(30, n.occasions-1)
   phi <- rep(0.65, n.occasions-1)
   p <- rep(0.4, n.occasions-1)

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

   # Apply simulation function
   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)

   # 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))}

   # Parameters monitored
```

```
   parameters <- c("mean.phi", "mean.p")

   # Call WinBUGS from R
   out <- bugs(bugs.data, inits, parameters, "cjs-c-c.bug", n.chains =
nchains, n.thin = nthin, n.iter = niter, n.burnin = nburn, debug = FALSE,
bugs.directory = bugs.dir, working.directory = getwd())

   # Store results
   phi.est[sim] <- out$mean$mean.phi
   p.est[sim] <- out$mean$mean.p
} #sim
```

After completion of the simulations, we produce histograms of the two quantities of interest and compare them with the parameter values used to simulate the data.

```
par(mfrow = c(1, 2))
hist(phi.est, nclass = 15, col = "lightgrey", las = 1, ylab = "Frequency",
xlab = expression(phi), main = "")
abline(v = 0.65, col = "red", lwd = 2)
hist(p.est, nclass = 15, col = "lightgrey", las = 1, ylab = "Frequency",
xlab = "p", main = "")
abline(v = 0.4, col = "red", lwd = 2)
```



The bias is the difference between the mean of the estimates of survival and recapture, respectively, and the parameters values used to simulate the data.

```
mean(phi.est)-phi[1]
0.001830816
mean(p.est)-p[1]
0.007406956
```

Finally, we calculate the standard deviation of the two quantities as a measure of the precision. They can be seen on the graphs as the spread of the distribution.

```
sd(phi.est)
```

```
0.05173908
sd(p.est)
0.05960649
```

Overall, this exercise shows that the parameter estimates are unbiased.

There are several ways to check whether a model produces unbiased and accurate estimates. The classical way is to use simulations, as we have seen just before. A second option is to analyse an m-array of expected values (Burnham et al. 1987). The expected values are constructed under the model of consideration, and thus the analysis of the model should yield unbiased parameter estimates if the model performs well. A last option, similar to the latter, is to simulate a very large data set. If the number of released individuals is large, then all possible capture histories should occur in the data. We will illustrate this option here. In order to avoid a long running time, we use the multinomial likelihood.

```
# Define the parameters
n.occasions <- 6                      # Number of recapture occasions
marked <- rep(10000, n.occasions-1)   # Annual number of newly marked
individuals
phi <- rep(0.65, n.occasions-1)
p <- rep(0.4, n.occasions-1)

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

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

# Specify model in BUGS language
sink("cjs-mnl.bug")
cat("
model {
# Priors and constraints
for (t in 1:(n.occasions-1)){
   phi[t] <- mean.phi
   p[t] <- mean.p
   }
mean.phi ~ dunif(0, 1)     # Prior for mean survival
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]               # 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
}
",fill = TRUE)
sink()

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

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

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

# MCMC settings
niter <- 2000
nthin <- 3
nburn <- 1000
nchains <- 3

# Call WinBUGS from R (BRT 0.03 min)
cjs.sim <- bugs(bugs.data, inits, parameters, "cjs-mnl.bug", n.chains =
nchains, n.thin = nthin, n.iter = niter, n.burnin = nburn, debug = T,
bugs.directory = bugs.dir, working.directory = getwd())
```

We see that the estimates are unbiased, i.e. nearly identical to the input parameters used for the simulations.

```
print(cjs.sim, 3)
Inference for Bugs model at "cjs-mnl.bug", fit using WinBUGS,
 3 chains, each with 2000 iterations (first 1000 discarded), n.thin = 3
 n.sims = 1002 iterations saved
            mean    sd    2.5%     25%     50%     75%   97.5%  Rhat n.eff
mean.phi   0.648 0.003   0.642   0.646   0.648   0.650   0.654 1.005   370
mean.p     0.402 0.003   0.396   0.400   0.402   0.404   0.409 1.005   370
deviance 147.372 2.050 145.400 145.900 146.700 148.200 152.800 1.000  1000
```

Exercise 5

Task: Take the data where survival of young and adult individuals is different (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.

Solution: We first simulate a capture-recapture data set with the function `simul.cjs`. Next, we create matrix **x**, which indicates the exact age of each individual at each occasion. The exact age is known, because all individuals have age 1 (are just born) when marked. Finally, we analyse the data with the CJS model formulated with the state-space likelihood.

The key here is the modelling of survival. We first index survival with **x**, which would result in a different estimate of survival for each age class, if no further modelling is applied. Yet, we want to induce a linear relationship for all but the first age class. Thus, we use an additional model (a GLM) which models the age-specific survival probabilities of all but the first age class as a linear function of age. We need to give priors for the survival probability of the first age class and for the intercept and the slope of the linear relationship.

Data simulation

```
# Define the parameters
n.occasions <- 10                   # Number of recapture occasions
marked <- rep(200, n.occasions-1)   # Annual number of newly marked juv.
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))

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

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

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

# Create matrix X indicating the age class
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
```

Data analysis

```
# Specify model in BUGS language
sink("cjs-age2.bug")
cat("
model {
# Priors and constraints
for (i in 1:nind){
   for (t in f[i]:(n.occasions-1)){
      logit(phi[i,t]) <- beta[x[i,t]]
      p[i,t] <- mean.p
      } #t
   } #i
beta[1] ~ dnorm(0, 0.001)          # Prior for first year survival
for (u in 2:(n.occasions-1)){
   beta[u] <- mu + gamma*(u-1)     # Linear model for age > 1y
   }
mu ~ dnorm(0, 0.001)I(-15,15)      # Prior for intercept of linear model
```

```
gamma ~ dnorm(0, 0.001)I(-15,15)   # Prior for slope of linear model
mean.p ~ dunif(0, 1)               # Prior for mean recapture
# Back-transformations
for (t in 1:(n.occasions-1)){
   phi.a[t] <- 1/(1+exp(-beta[t]))
   }

# Define the likelihood
for (i in 1:nind){
   # Ensures that individuals enter the sample with probability 1
   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 = c(rnorm(1), rep(NA,
n.occasions-2)), mu = rnorm(1), gamma = rnorm(1), mean.p = runif(1, 0, 1))}

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

# MCMC settings
niter <- 2000
nthin <- 3
nburn <- 1000
nchains <- 3

# Call WinBUGS from R (BRT 6 min)
cjs.age <- bugs(bugs.data, inits, parameters, "cjs-age2.bug", n.chains =
nchains, n.thin = nthin, n.iter = niter, n.burnin = nburn, debug = TRUE,
bugs.directory = bugs.dir, working.directory = getwd())

print(cjs.age, 3)
Inference for Bugs model at "cjs-age2.bug", fit using WinBUGS,
 3 chains, each with 2000 iterations (first 1000 discarded), n.thin = 3
 n.sims = 1002 iterations saved
            mean    sd    2.5%     25%     50%     75%   97.5%  Rhat n.eff
phi.a[1]   0.283 0.016   0.251   0.272   0.283   0.294   0.316 1.003  1000
phi.a[2]   0.648 0.027   0.596   0.629   0.648   0.667   0.703 1.010   180
phi.a[3]   0.658 0.019   0.622   0.644   0.657   0.671   0.697 1.003   660
phi.a[4]   0.666 0.022   0.626   0.650   0.666   0.681   0.709 1.001  1000
phi.a[5]   0.674 0.032   0.613   0.651   0.675   0.697   0.736 1.005   350
phi.a[6]   0.682 0.045   0.591   0.652   0.683   0.713   0.764 1.008   240
phi.a[7]   0.689 0.058   0.572   0.652   0.693   0.729   0.791 1.010   200
phi.a[8]   0.696 0.071   0.547   0.650   0.702   0.746   0.816 1.011   190
phi.a[9]   0.702 0.084   0.525   0.647   0.711   0.761   0.842 1.012   180
mu         0.575 0.178   0.233   0.448   0.571   0.691   0.936 1.019   150
gamma      0.039 0.070  -0.095  -0.008   0.042   0.086   0.170 1.012   160
mean.p     0.491 0.024   0.445   0.475   0.490   0.507   0.538 1.005   370
```

```
deviance 2942.084 48.102 2850.050 2912.000 2941.000 2973.000 3035.000 1.008   240
```

Here we used again a range restriction for the priors of `mu` and `gamma` to help with convergence. Another option that is often helpful for faster convergence is the standardisation of the covariable (in our case `u-1`). Thus, we could write, e.g., `beta[u] <- mu + gamma*(u-1-n.occasions/2)`.

Exercise 6

Task: 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.

Solution: We simulate capture-recapture data using the function `simul.cjs`. For the analysis using the state-space model, we construct matrix **x**, indicating the age of each individual at each occasion. Since we distinguish only two age classes here, **x** has entries of either 1 or 2. To fit the model with the state-space likelihood we index survival probability with **x** and time. We then apply a GLMM to model survival of the first age class with the random annual variation and a GLM for the survival of the second age class to constrain the estimates to be the same in each year. For the analysis using the multinomial likelihood, we first construct m-arrays. All individual that are recaptured are released as adults, and thus two m-arrays need to be produced (one for individuals released as young and one for individuals released as adults). We then define the multinomial likelihood for each m-array. Again we apply a GLMM to model survival of the first age class and a GLM for the survival of the second age class.

Data simulation

```
# Define the parameters
n.occasions <- 15                   # Number of recapture occasions
marked <- rep(100, n.occasions-1)   # Number of newly marked juveniles
phi.juv <- 0.4                      # Juvenile annual survival
var.phi <- 0.3
phi.ad <- 0.8                       # Adult annual survival
p <- rep(0.6, n.occasions-1)        # Recapture
phi.juv.t <- plogis(rnorm(n.occasions-1, qlogis(phi.juv), var.phi^0.5))
phi.j <- matrix(0, nrow = n.occasions-1, ncol = n.occasions-1)
for (t in 1 :(n.occasions-1)){
   phi.j[t,t:(n.occasions-1)] <- c(phi.juv.t[t], rep(phi.ad, n.occasions-1-
t))
   }

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

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

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

# Create matrices X indicating the age class
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] <- 2
      x[i,f[i]] <- 1
      } #t
   } #i
```

Data analysis

a) State-space likelihood

```
# Specify model in BUGS language
sink("cjs-age2.bug")
cat("
model {
# Priors and constraints
for (i in 1:nind){
   for (t in f[i]:(n.occasions-1)){
      logit(phi[i,t]) <- beta[x[i,t],t]
      p[i,t] <- mean.p
      } #t
   } #i
for (t in 1:(n.occasions-1)){
   beta[1,t] <- mu + epsilon[t]    # Juvenile survival
   epsilon[t] ~ dnorm(0, tau)      # Temporal variation of juv survival
   phi.j[t] <- 1/(1+exp(-beta[1,t]))
   beta[2,t] <- lphi.ad            # Constrain ad survival to be constant
   }

mu <- log(mean.phij / (1-mean.phij))
mean.phij ~ dunif(0, 1)            # Prior for mean juv survival
sigma ~ dunif(0, 10)               # Prior on sd of temp. var
tau <- pow(sigma, -2)
sigma2 <- pow(sigma, 2)
lphi.ad <- log(mean.phiad / (1-mean.phiad))
mean.phiad ~ dunif(0, 1)           # Prior for mean ad survival
mean.p ~ dunif(0, 1)               # Prior for mean recapture

# Define the likelihood
for (i in 1:nind){
   # Ensures that individuals enter the sample with probability 1
   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), mean.phij = runif(1, 0, 1),
mean.phiad = runif(1, 0, 1), sigma = runif(1, 0, 5), mean.p = runif(1, 0,
1))}

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

# MCMC settings
niter <- 2000
nthin <- 3
nburn <- 1000
nchains <- 3

# Call WinBUGS from R (BRT 9 min)
cjs.1 <- bugs(bugs.data, inits, parameters, "cjs-age2.bug", n.chains =
nchains, n.thin = nthin, n.iter = niter, n.burnin = nburn, debug = TRUE,
bugs.directory = bugs.dir, working.directory = getwd())


print(cjs.1, digits = 3)
Inference for Bugs model at "cjs-age2.bug", fit using WinBUGS,
 3 chains, each with 2000 iterations (first 1000 discarded), n.thin = 3
 n.sims = 1002 iterations saved
               mean     sd    2.5%      25%      50%      75%    97.5%  Rhat n.eff
mean.phij     0.418  0.058   0.301    0.382    0.419    0.455    0.537 1.042    52
phi.j[1]      0.234  0.045   0.153    0.203    0.233    0.262    0.327 1.002  1000
phi.j[2]      0.621  0.055   0.516    0.585    0.621    0.657    0.727 1.003   580
phi.j[3]      0.474  0.056   0.364    0.435    0.473    0.512    0.580 1.003   600
phi.j[4]      0.751  0.055   0.645    0.713    0.754    0.789    0.850 1.003   660
phi.j[5]      0.423  0.055   0.319    0.385    0.421    0.461    0.527 1.000  1000
phi.j[6]      0.392  0.050   0.295    0.358    0.393    0.425    0.496 1.000  1000
phi.j[7]      0.383  0.051   0.286    0.349    0.383    0.414    0.482 1.001  1000
phi.j[8]      0.277  0.047   0.188    0.244    0.276    0.309    0.372 1.005  1000
phi.j[9]      0.307  0.049   0.216    0.272    0.305    0.340    0.405 1.006   340
phi.j[10]     0.592  0.057   0.484    0.550    0.591    0.631    0.707 1.002  1000
phi.j[11]     0.346  0.051   0.248    0.310    0.344    0.383    0.444 1.002  1000
phi.j[12]     0.193  0.043   0.118    0.162    0.189    0.221    0.287 1.002   830
phi.j[13]     0.514  0.063   0.396    0.471    0.510    0.554    0.646 1.008   230
phi.j[14]     0.434  0.072   0.303    0.386    0.428    0.479    0.589 1.002   900
sigma2        0.722  0.385   0.252    0.467    0.650    0.865    1.700 1.004   440
mean.phiad    0.803  0.010   0.783    0.796    0.803    0.810    0.823 1.002  1000
mean.p        0.581  0.013   0.556    0.572    0.581    0.590    0.607 1.002   860
deviance   4414.788 55.997 4306.025 4378.000 4414.500 4449.000 4524.950 1.001  1000
```

b) Multinomial likelihood

```
# Create m-arrays
cap <- apply(CH, 1, sum)
ind <- which(cap >= 2)
CH.R <- CH[ind,]    # Juvenile CH recaptured at least once
CH.N <- CH[-ind,]   # Juvenile CH never recaptured
# Remove first capture
first <- numeric()
for (i in 1:dim(CH.R)[1]){
```

```
   first[i] <- min(which(CH.R[i,]==1))
   }
CH.R1 <- CH.R
for (i in 1:dim(CH.R)[1]){
   CH.R1[i,first[i]] <- 0
   }
# Create m-array of those recaptured at least once
CH.A.marray <- marray(CH.R1)
# Create CH matrix for juveniles, ignoring subsequent recaptures
second <- numeric()
for (i in 1:dim(CH.R1)[1]){
   second[i] <- min(which(CH.R1[i,]==1))
   }
CH.R2 <- matrix(0, nrow = dim(CH.R)[1], ncol = dim(CH.R)[2])
for (i in 1:dim(CH.R)[1]){
   CH.R2[i,first[i]] <- 1
   CH.R2[i,second[i]] <- 1
   }
# Create m-array for these
CH.R.marray <- marray(CH.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.R.marray[,dim(CH)[2]] <- 0
# Create the m-array for juveniles never recaptured and add it to the
previous m-array
CH.N.marray <- marray(CH.N)
CH.J.marray <- CH.R.marray + CH.N.marray


# Specify model in BUGS language
sink("cjs-mnl-2age.bug")
cat("
model {
# Priors and constraints
for (t in 1:(n.occasions-1)){
   logit(phi.juv[t]) <- mu + epsilon[t]
   epsilon[t] ~ dnorm(0, tau)I(-15,15)     # Range restriction
   phi.j[t] <- 1/(1+exp(-mu-epsilon[t]))
   phi.ad[t] <- mean.phiad
   p[t] <- mean.p
   }
mu <- log(mean.phij / (1-mean.phij))
mean.phij ~ dunif(0, 1)            # Prior for mean juv survival
sigma ~ dunif(0, 10)               # Prior on sd of temp. var
tau <- pow(sigma, -2)
sigma2 <- pow(sigma, 2)
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)){
   marrj[t,1:n.occasions] ~ dmulti(prj[t,], rj[t])
   marra[t,1:n.occasions] ~ dmulti(pra[t,], ra[t])
   }
# Calculate the number of birds released each year
for (t in 1:(n.occasions-1)){
   rj[t] <- sum(marrj[t,])
   ra[t] <- sum(marra[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
   prj[t,t] <- phi.juv[t]*p[t]
   pra[t,t] <- phi.ad[t]*p[t]
   # Above main diagonal
   for (j in (t+1):(n.occasions-1)){
      prj[t,j] <- phi.juv[t]*prod(phi.ad[(t+1):j])*prod(q[t:(j-1)])*p[j]
      pra[t,j] <- prod(phi.ad[t:j])*prod(q[t:(j-1)])*p[j]
      } # j
   # Below main diagonal
   for (j in 1:(t-1)){
      prj[t,j] <- 0
      pra[t,j] <- 0
      } # j
   } # t
# Last column: probability of non-recapture
for (t in 1:(n.occasions-1)){
   prj[t,n.occasions] <- 1-sum(prj[t,1:(n.occasions-1)])
   pra[t,n.occasions] <- 1-sum(pra[t,1:(n.occasions-1)])
   } # t
}
",fill = TRUE)
sink()


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

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

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

# MCMC settings
niter <- 5000
nthin <- 6
nburn <- 2500
nchains <- 3

# Call WinBUGS from R (BRT 1 min)
cjs.2 <- bugs(bugs.data, inits, parameters, "cjs-mnl-2age.bug", n.chains =
nchains, n.thin = nthin, n.iter = niter, n.burnin = nburn, debug = T,
bugs.directory = bugs.dir, working.directory = getwd())

print(cjs.2, 3)
Inference for Bugs model at "cjs-mnl-2age.bug", fit using WinBUGS,
 3 chains, each with 5000 iterations (first 2500 discarded), n.thin = 6
 n.sims = 1251 iterations saved
              mean    sd   2.5%    25%    50%    75%  97.5%  Rhat n.eff
mean.phij    0.425 0.057  0.314  0.389  0.424  0.458  0.545 1.017   140
phi.j[1]     0.235 0.045  0.155  0.203  0.234  0.266  0.326 1.001  1300
phi.j[2]     0.623 0.057  0.507  0.586  0.626  0.662  0.730 1.000  1300
phi.j[3]     0.479 0.054  0.381  0.442  0.476  0.514  0.586 1.000  1300
phi.j[4]     0.752 0.053  0.648  0.717  0.753  0.788  0.855 1.003   970
phi.j[5]     0.425 0.052  0.325  0.388  0.423  0.461  0.535 1.002  1300
phi.j[6]     0.391 0.053  0.295  0.353  0.390  0.425  0.500 1.000  1300
phi.j[7]     0.382 0.052  0.278  0.346  0.382  0.417  0.486 1.002  1300
phi.j[8]     0.275 0.048  0.187  0.240  0.274  0.306  0.372 1.002   940
phi.j[9]     0.306 0.049  0.218  0.270  0.303  0.338  0.405 1.003   670
phi.j[10]    0.595 0.057  0.485  0.558  0.594  0.635  0.706 1.005   420
phi.j[11]    0.343 0.050  0.246  0.310  0.342  0.376  0.446 1.000  1300
phi.j[12]    0.188 0.042  0.115  0.158  0.186  0.215  0.283 1.003   560
```

```
phi.j[13]     0.510 0.060   0.390   0.469   0.509   0.551   0.626 1.000  1300
phi.j[14]     0.434 0.072   0.300   0.385   0.434   0.482   0.581 1.000  1300
sigma2        0.726 0.401   0.253   0.469   0.632   0.877   1.812 1.000  1300
mean.phiad    0.803 0.010   0.783   0.796   0.803   0.810   0.822 1.002  1100
mean.p        0.581 0.013   0.555   0.572   0.580   0.590   0.606 1.000  1300
deviance    479.421 5.509 470.700 475.400 478.800 482.950 491.000 1.002  1300
```

Both models produce almost identical results, as we have expected.

Chapter 8

Exercise 1

Task: Simulate mark-recovery data of two groups, both groups have a survival probability of 0.5, the first group a recovery probability of 0.1, the second a recovery probability of 0.2. The study is conducted for 10 years and each year 50 individuals are marked in each group. Fit the model (s, r_g) using a) the multinomial and b) the state-space likelihood.

Solution: We simulate the data using function `simul.mr` and covert the obtained individual capture histories to the m-array format using function `marray.dead`. For the analysis of the data with the multinomial likelihood we use the data in the m-array format. We write separate likelihoods for the data sets of each group and then constraint the survival probabilities of both groups to be the same. For the analysis with the state-space likelihood we use the individual capture-histories and define a variable indicating the group membership of each individual. In the analysing model we then use this grouping variable as an index for the recovery probability, thus we apply a simple linear model. In fact, we also apply a simple linear model for the survival probabilities, in this case it is just an intercept model.

<u>*Data simulation*</u>

```
# Define the parameters
n.occasions <- 10                  # Number of occasions
marked <- rep(50, n.occasions)     # Annual number of newly marked
individuals
s <- rep(0.5, n.occasions)
r1 <- rep(0.1, n.occasions)
r2 <- rep(0.2, n.occasions)

# Define matrices with the survival and recovery probabilities
S <- matrix(rep(s, sum(marked)), ncol = n.occasions, nrow = sum(marked),
byrow = TRUE)
R1 <- matrix(rep(r1, sum(marked)), ncol = n.occasions, nrow = sum(marked),
byrow = TRUE)
R2 <- matrix(rep(r2, sum(marked)), ncol = n.occasions, nrow = sum(marked),
byrow = TRUE)

# Apply function
MR1 <- simul.mr(S, R1, marked)
MR2 <- simul.mr(S, R2, marked)

# Merge capture-histories
MR <- rbind(MR1, MR2)

# Create group variable
group <- c(rep(1, dim(MR1)[1]), rep(2, dim(MR2)[1]))

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

# Create m-arrays
marr1 <- marray.dead(MR1)
marr2 <- marray.dead(MR2)
```

Data analyses

a) Multinomial likelihood

```
# Specify model in BUGS language
sink("mr-mnl.bug")
cat("
model {
# Priors and constraints
for (t in 1:n.occasions){
   s[t] <- mean.s
   r1[t] <- mean.r1
   r2[t] <- mean.r2
   }
mean.s ~ dunif(0, 1)
mean.r1 ~ dunif(0, 1)
mean.r2 ~ dunif(0, 1)

# Define the multinomial likelihoods
for (t in 1:n.occasions){
   marr1[t,1:(n.occasions+1)] ~ dmulti(pr1[t,], rel1[t])
   marr2[t,1:(n.occasions+1)] ~ dmulti(pr2[t,], rel2[t])
   }
# Calculate the number of birds released each year
for (t in 1:n.occasions){
   rel1[t] <- sum(marr1[t,])
   rel2[t] <- sum(marr2[t,])
   }
# Define the cell probabilities of the m-array
# Main diagonal
for (t in 1:n.occasions){
   pr1[t,t] <- (1-s[t])*r1[t]
   pr2[t,t] <- (1-s[t])*r2[t]
   # Above main diagonal
   for (j in (t+1):n.occasions){
      pr1[t,j] <- prod(s[t:(j-1)])*(1-s[j])*r1[j]
      pr2[t,j] <- prod(s[t:(j-1)])*(1-s[j])*r2[j]
      } # j
   # Below main diagonal
   for (j in 1:(t-1)){
      pr1[t,j] <- 0
      pr2[t,j] <- 0
      } # j
   } # t
# Last column: probability of non-recovery
for (t in 1:n.occasions){
   pr1[t,n.occasions+1] <- 1-sum(pr1[t,1:n.occasions])
   pr2[t,n.occasions+1] <- 1-sum(pr2[t,1:n.occasions])
   } # t
}
",fill = TRUE)
sink()

# Bundle data
bugs.data <- list(marr1 = marr1, marr2 = marr2, n.occasions =
dim(marr1)[2]-1)

# Initial values
inits <- function(){list(mean.s = runif(1, 0, 1), mean.r1 = runif(1, 0, 1),
mean.r2 = runif(1, 0, 1))}

# Parameters monitored
parameters <- c("mean.s", "mean.r1", "mean.r2")
```

```
# MCMC settings
niter <- 5000
nthin <- 6
nburn <- 2000
nchains <- 3

# Call WinBUGS from R (BRT 0.1 min)
mr.age <- bugs(bugs.data, inits, parameters, "mr-mnl.bug", n.chains =
nchains, n.thin = nthin, n.iter = niter, n.burnin = nburn, debug = TRUE,
bugs.directory = bugs.dir)

# Inspect results
print(mr.age, 3)
Inference for Bugs model at "mr-mnl.bug", fit using WinBUGS,
 3 chains, each with 5000 iterations (first 2000 discarded), n.thin = 6
 n.sims = 1500 iterations saved
            mean    sd    2.5%     25%     50%     75%   97.5%  Rhat n.eff
mean.s     0.530 0.039   0.456   0.503   0.529   0.556   0.609 1.001  1500
mean.r1    0.099 0.014   0.074   0.089   0.098   0.108   0.130 1.000  1500
mean.r2    0.194 0.019   0.158   0.180   0.194   0.207   0.231 1.004  1100
deviance 244.077 2.430 241.300 242.200 243.450 245.200 250.352 1.005   750
```

b) State-space likelihood

```
# Specify model in BUGS language
sink("mr.ss.bug")
cat("
model {

# Priors and constraints
for (i in 1:nind){
   for (t in 1:n.occasions){
      s[i,t] <- mean.s
      r[i,t] <- mean.r[group[i]]
      } #t
   } #i

mean.s ~ dunif(0, 1)
for (u in 1:g){
   mean.r[u] ~ dunif(0, 1)
   }

# 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] <- s[i,t-1] * z[i,t-1]
      # Observation process
      y[i,t] ~ dbern(mu2[i,t])
      mu2[i,t] <- r[i,t-1] * (z[i,t-1] - z[i,t])
      } #t
   } #i
}
",fill = TRUE)
sink()

# Bundle data
```

```
bugs.data <- list(y = MR, f = f, group = group, g = length(unique(group)),
nind = dim(MR)[1], n.occasions = dim(MR)[2], z = known.state.mr(MR))

# Initial values
inits <- function(){list(z = mr.init.z(MR), mean.s = runif(1, 0, 1), mean.r
= runif(2, 0, 1))}


# Parameters monitored
parameters <- c("mean.s", "mean.r")

# MCMC settings
niter <- 7000
nthin <- 3
nburn <- 5000
nchains <- 3

# Call WinBUGS from R (BRT 11 min)
mr <- bugs(bugs.data, inits, parameters, "mr.ss.bug", n.chains = nchains,
n.thin = nthin, n.iter = niter, n.burnin = nburn, debug = TRUE,
working.directory = getwd(), bugs.directory = bugs.dir)


# Inspect results
print(mr, digits = 3)
Inference for Bugs model at "mr.ss.bug", fit using WinBUGS,
 3 chains, each with 7000 iterations (first 5000 discarded), n.thin = 3
 n.sims = 2001 iterations saved
              mean    sd     2.5%      25%      50%      75%    97.5%  Rhat n.eff
mean.s       0.533 0.036    0.459    0.509    0.533    0.558    0.603 1.010   210
mean.r[1]    0.100 0.014    0.073    0.089    0.099    0.109    0.129 1.001  2000
mean.r[2]    0.194 0.019    0.157    0.181    0.194    0.207    0.231 1.001  2000
deviance  1049.838 3.759 1043.000 1047.000 1050.000 1052.000 1058.000 1.002  1100
```

The results from both analyses are nearly identical, as expected.

Exercise 2

Task: It is quite typical for population studies that only nestlings are marked, but no adult individuals. This is because the capture of adults is often much more time consuming than the marking of nestlings, which can be easily marked in the nest. Simulate data from a study on a common tern population in which only nestlings are marked. The study duration is 15 years, in each year 200 nestlings are marked and the parameters are *sj* = 0.3, *sa* = 0.8, *rj* = 0.25, and *ra* = 0.15. Analyze these data with a) the data generating model, and b) using a model in which the recovery probability are the same in both age classes. Comment on the parameter estimates that you obtain from both models.

Solution: We simulate the data using function `simul.mr` and convert the generated individual capture-histories into the m-array format using function `marray.dead`. We did this last step because we intend to analyse the data with the multinomial likelihood. Of course, the analysis using the state-space likelihood is also possible, it would require in addition that we construct a matrix indicating the age of each individual at each time. The analysis with the multinomial likelihood does not pose any specific problems. In the model where the recovery probabilities of both age classes is the same, we have to apply a linear model (or in other words, a constraint such that both recovery probabilities are the same).

We plot the posterior density of the estimated parameters of both models in order to see whether the parameters behave well.

Data simulation

```
n.occasions <- 15                  # Number of occasions
marked.j <- rep(200, n.occasions)  # Annual number of newly marked young
sjuv <- 0.3                        # First year survival probability
sad <- 0.7                         # Adult survival probability
rjuv <- 0.25                       # First year recovery probability
rad <- 0.15                        # Adult recovery probability
sj <- c(sjuv, rep(sad, n.occasions-1))
rj <- c(rjuv, rep(rad, n.occasions-1))
sa <- rep(sad, n.occasions)
ra <- rep(rad, n.occasions)

# Define matrices with the survival and recovery probabilities
S <- matrix(0, ncol = n.occasions, nrow = sum(marked.j))
for (i in 1:length(marked.j)){
   S[(sum(marked.j[1:i])-marked.j[i]+1):sum(marked.j[1:i]),i:n.occasions]
<-  matrix(rep(sj[1:(n.occasions-i+1)],  marked.j[i]),  ncol  =  n.occasions-
i+1, byrow = TRUE)
   }
R <- matrix(0, ncol = n.occasions, nrow = sum(marked.j))
for (i in 1:length(marked.j)){
   R[(sum(marked.j[1:i])-marked.j[i]+1):sum(marked.j[1:i]),i:n.occasions]
<-  matrix(rep(rj[1:(n.occasions-i+1)],  marked.j[i]),  ncol  =  n.occasions-
i+1, byrow = TRUE)
   }

# Apply simulation function
MR <- simul.mr(S, R, marked.j)

# Create m-arrays
marr <- marray.dead(MR)
```

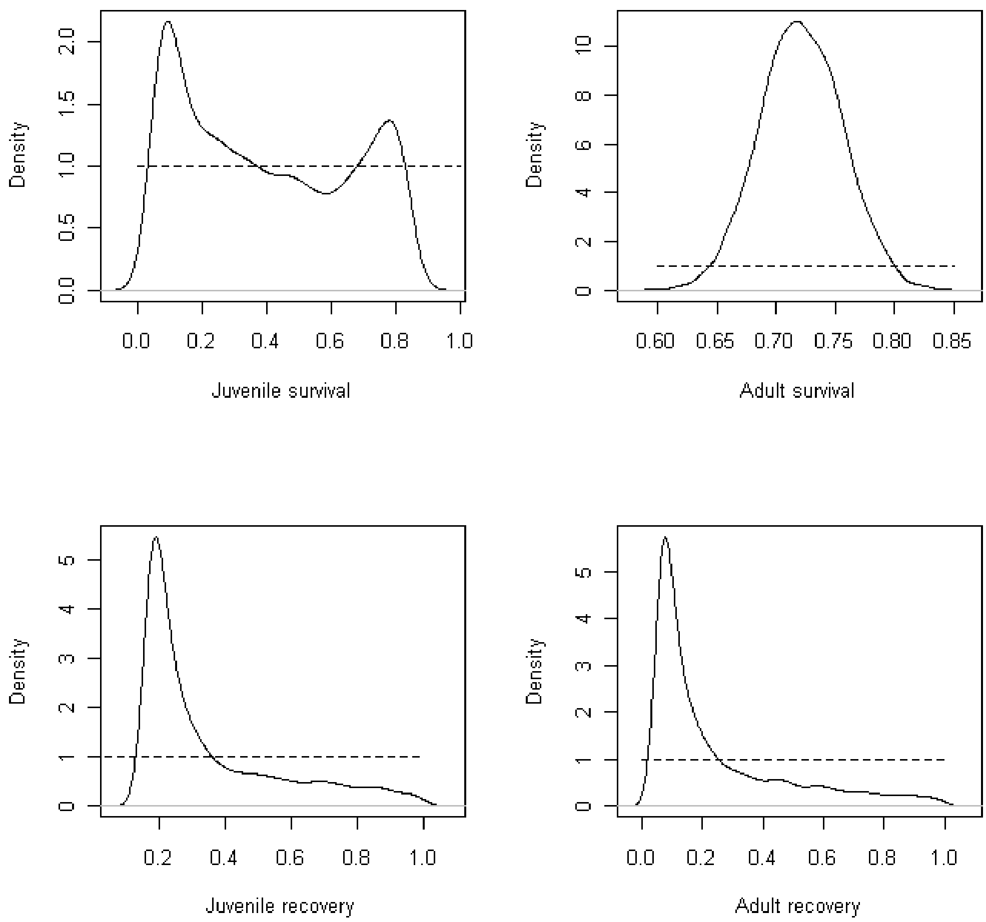
Data analysis

a) Using the data generating model $\{s_{a2}, r_{a2}\}$

```
# Specify model in BUGS language
sink("mr-mnl-age1.bug")
cat("
model {
# Priors and constraints
for (t in 1:n.occasions){
   sj[t] <- mean.sj
   sa[t] <- mean.sa
   rj[t] <- mean.rj
   ra[t] <- mean.ra
   }
mean.sj ~ dunif(0, 1)
mean.sa ~ dunif(0, 1)
mean.rj ~ dunif(0, 1)
mean.ra ~ dunif(0, 1)
# Define the multinomial likelihoods
for (t in 1:n.occasions){
   marr[t,1:(n.occasions+1)] ~ dmulti(pr[t,], rel[t])
   }
# Calculate the number of birds released each year
for (t in 1:n.occasions){
   rel[t] <- sum(marr[t,])
```

```
   }
# Define the cell probabilities of the m-array
# Main diagonal
for (t in 1:n.occasions){
   pr[t,t] <- (1-sj[t])*rj[t]
   # Further above main diagonal
   for (j in (t+2):n.occasions){
      pr[t,j] <- sj[t]*prod(sa[(t+1):(j-1)])*(1-sa[j])*ra[j]
      } # j
   # Below main diagonal
   for (j in 1:(t-1)){
      pr[t,j] <- 0
      } # j
   } # t
for (t in 1:(n.occasions-1)){
   # One above main diagonal
   pr[t,t+1] <- sj[t]*(1-sa[t+1])*ra[t+1]
   } # t
# Last column: probability of non-recovery
for (t in 1:n.occasions){
   pr[t,n.occasions+1] <- 1-sum(pr[t,1:n.occasions])
   } # t
}
",fill = TRUE)
sink()


# Bundle data
bugs.data <- list(marr = marr, n.occasions = dim(marr)[2]-1)

# Initial values
inits <- function(){list(mean.sj = runif(1, 0, 1), mean.sa = runif(1, 0,
1), mean.rj = runif(1, 0, 1), mean.ra = runif(1, 0, 1))}

# Parameters monitored
parameters <- c("mean.sj", "mean.sa", "mean.rj", "mean.ra")

# MCMC settings
niter <- 20000
nthin <- 6
nburn <- 10000
nchains <- 3

# Call WinBUGS from R (BRT 2 min)
mr.age1 <- bugs(bugs.data, inits, parameters, "mr-mnl-age1.bug", n.chains =
nchains, n.thin = nthin, n.iter = niter, n.burnin = nburn, debug = TRUE,
working.directory = getwd(), bugs.directory = bugs.dir)

# Inspect results
print(mr.age1, 3)
          mean    sd  2.5%   25%   50%   75% 97.5%  Rhat n.eff
mean.sj  0.355 0.248 0.045 0.117 0.312 0.573 0.804 1.125    21
mean.sa  0.725 0.035 0.658 0.701 0.724 0.748 0.796 1.001  5000
mean.rj  0.338 0.193 0.177 0.198 0.253 0.408 0.887 1.093    27
mean.ra  0.251 0.245 0.048 0.072 0.134 0.357 0.899 1.124    21


# Plot posterior distribution of the parameters
par(mfrow = c(2, 2))
plot(density(mr.age1$sims.list$mean.sj), xlab = "Juvenile survival", main =
"")
segments(0, 1, 1, 1, lty = 2)
```

```
plot(density(mr.age1$sims.list$mean.sa), xlab = "Adult survival", main =
"")
segments(0.6, 1, 0.85, 1, lty = 2)
plot(density(mr.age1$sims.list$mean.rj), xlab = "Juvenile recovery", main =
"")
segments(0, 1, 1, 1, lty = 2)
plot(density(mr.age1$sims.list$mean.ra), xlab = "Adult recovery", main =
"")
segments(0, 1, 1, 1, lty = 2)
```



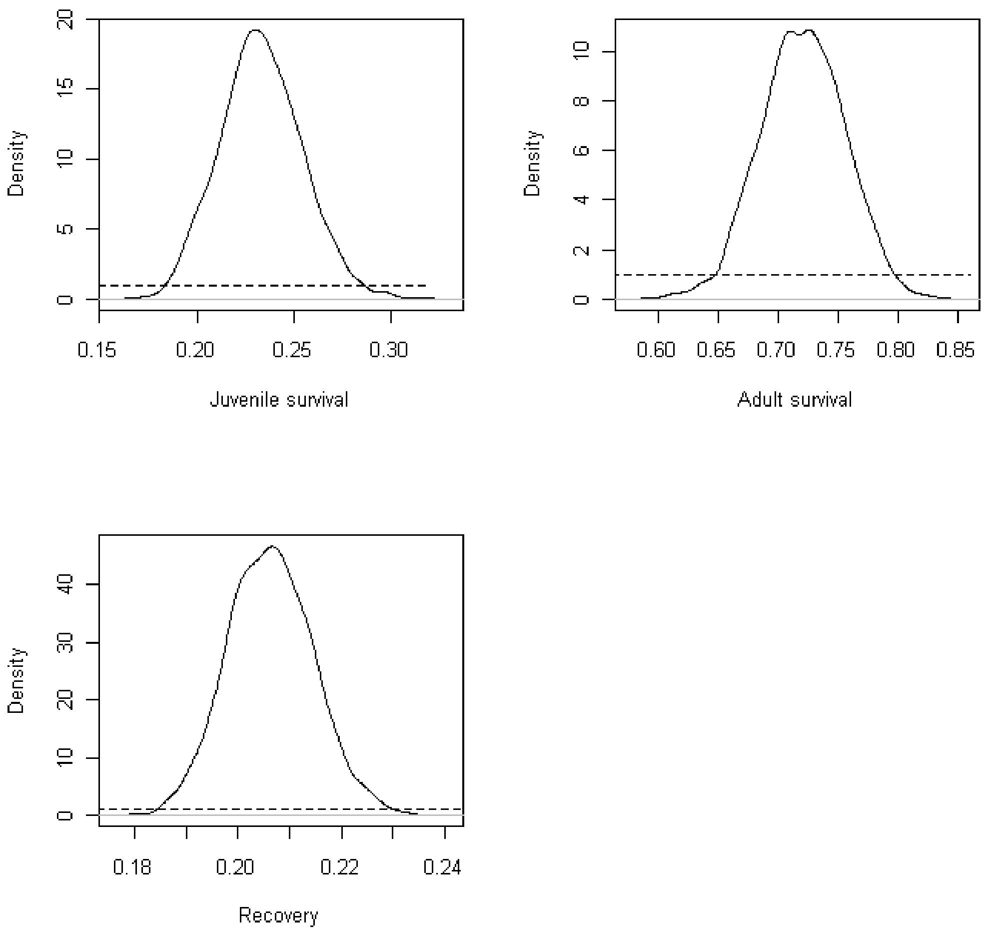
The posterior distributions of all parameters except for the adult survival do not look very nice. Indeed, it is well known that only adult survival is identifiable in this model (see e.g., Anderson et al. 1985, J. Anim. Ecol. 54: 89-98).

b) Using the model {s_{a2}, r}

```
# Specify model in BUGS language
sink("mr-mnl-age2.bug")
cat("
model {
# Priors and constraints
for (t in 1:n.occasions){
   sj[t] <- mean.sj
```

```
   sa[t] <- mean.sa
   r[t] <- mean.r
   }
mean.sj ~ dunif(0, 1)
mean.sa ~ dunif(0, 1)
mean.r ~ dunif(0, 1)
# Define the multinomial likelihoods
for (t in 1:n.occasions){
   marr[t,1:(n.occasions+1)] ~ dmulti(pr[t,], rel[t])
   }
# Calculate the number of birds released each year
for (t in 1:n.occasions){
   rel[t] <- sum(marr[t,])
   }
# Define the cell probabilities of the m-array
# Main diagonal
for (t in 1:n.occasions){
   pr[t,t] <- (1-sj[t])*r[t]
   # Further above main diagonal
   for (j in (t+2):n.occasions){
      pr[t,j] <- sj[t]*prod(sa[(t+1):(j-1)])*(1-sa[j])*r[j]
      } # j
   # Below main diagonal
   for (j in 1:(t-1)){
      pr[t,j] <- 0
      } # j
   } # t
for (t in 1:(n.occasions-1)){
   # One above main diagonal
   pr[t,t+1] <- sj[t]*(1-sa[t+1])*r[t+1]
   } # t
# Last column: probability of non-recovery
for (t in 1:n.occasions){
   pr[t,n.occasions+1] <- 1-sum(pr[t,1:n.occasions])
   } # t
}
",fill = TRUE)
sink()


# Bundle data
bugs.data <- list(marr = marr, n.occasions = dim(marr)[2]-1)

# Initial values
inits <- function(){list(mean.sj = runif(1, 0, 1), mean.sa = runif(1, 0,
1), mean.r = runif(1, 0, 1))}

# Parameters monitored
parameters <- c("mean.sj", "mean.sa", "mean.r")

# MCMC settings
niter <- 20000
nthin <- 6
nburn <- 10000
nchains <- 3

# Call WinBUGS from R (BRT 2 min)
mr.age2 <- bugs(bugs.data, inits, parameters, "mr-mnl-age2.bug", n.chains =
nchains, n.thin = nthin, n.iter = niter, n.burnin = nburn, debug = TRUE,
working.directory = getwd(), bugs.directory = bugs.dir)

# Inspect results
```

```
print(mr.age2, 3)
          mean    sd   2.5%    25%    50%    75%   97.5%  Rhat n.eff
mean.sj  0.193 0.019  0.159  0.180  0.192  0.205  0.232 1.001  5000
mean.sa  0.725 0.035  0.654  0.702  0.725  0.749  0.796 1.001  5000
mean.r   0.215 0.008  0.200  0.210  0.215  0.221  0.231 1.001  5000

# Plot posterior distribution of the parameters
par(mfrow = c(2, 2))
plot(density(mr.age2$sims.list$mean.sj), xlab = "Juvenile survival", main =
"")
segments(0.15, 1, 0.32, 1, lty = 2)
plot(density(mr.age2$sims.list$mean.sa), xlab = "Adult survival", main =
"")
segments(0.5, 1, 0.86, 1, lty = 2)
plot(density(mr.age2$sims.list$mean.r), xlab = "Recovery", main = "")
segments(0.1, 1, 0.27, 1, lty = 2)
```



The posterior distributions of all parameters look much better now and indeed, all are identifiable. However, the parameter estimates are biased, because the analysing model does not correspond to the data generating model. To explore the magnitude of this bias, the above exercise would have to be repeated many times. Another possibility to gauge the bias is to increase the number of released animals significantly (e.g. 2000 at each occasion; see also exercise 4 of chapter 7).

Exercise 3

Task: Simulate mark-recovery data with the following characteristics: one group, during each of the 20 study years 500 individuals are released, the survival probability declines linearly from 0.8 in the first year to 0.6 in the last study year, the recovery probability is constant at 0.05. Analyze these data with the multinomial model.

Solution: We simulate individual capture histories with function `simul.mr` and convert the data into the m-array format using function `marray.dead`. For the analysis, we have to apply a model for the survival probability, such that it is a linear function of time. Two parameters are needed, an intercept and a slope, and for both we have to specify prior distributions. This relationship between survival and time needs to be defined on an appropriate scale (e.g. logit) to ensure that all survival probabilities remain in the interval between 0 and 1.

Data simulation

```
n.occasions <- 20                    # Number of occasions
marked <- rep(500, n.occasions)      # Annual number of newly marked young
s <- seq(0.8, 0.6, length.out = n.occasions)    # Survival probability
r <- rep(0.05, n.occasions)                     # Recovery probability

# Define matrices with the survival and recovery probabilities
S <- matrix(rep(s, sum(marked)), ncol = n.occasions, nrow = sum(marked),
byrow = TRUE)
R <- matrix(rep(r, sum(marked)), ncol = n.occasions, nrow = sum(marked),
byrow = TRUE)

# Apply simulation function
MR <- simul.mr(S, R, marked)

# Create m-arrays
marr <- marray.dead(MR)
```

Data analysis

```
# Specify model in BUGS language
sink("mr-trend.bug")
cat("
model {
# Priors and constraints
for (t in 1:n.occasions){
   logit(s[t]) <- mu + slope*(t-n.occasions/2)  # standardise trend
   r[t] <- mean.r
   }
mu ~ dnorm(0, 0.001)
slope ~ dnorm(0, 0.001)
mean.r ~ dunif(0, 1)
# Define the multinomial likelihoods
for (t in 1:n.occasions){
   marr[t,1:(n.occasions+1)] ~ dmulti(pr[t,], rel[t])
   }
# Calculate the number of birds released each year
for (t in 1:n.occasions){
   rel[t] <- sum(marr[t,])
   }
```

```
# Define the cell probabilities of the m-array
# Main diagonal
for (t in 1:n.occasions){
   pr[t,t] <- (1-s[t])*r[t]
   # Above main diagonal
   for (j in (t+1):n.occasions){
      pr[t,j] <- prod(s[t:(j-1)])*(1-s[j])*r[j]
      } # j
   # Below main diagonal
   for (j in 1:(t-1)){
      pr[t,j] <- 0
      } # j
   } # t
# Last column: probability of non-recovery
for (t in 1:n.occasions){
   pr[t,n.occasions+1] <- 1-sum(pr[t,1:n.occasions])
   } # t
}
",fill = TRUE)
sink()


# Bundle data
bugs.data <- list(marr = marr, n.occasions = dim(marr)[2]-1)

# Initial values
inits <- function(){list(mu = rnorm(1), slope = rnorm(1), mean.r = runif(1,
0, 1))}

# Parameters monitored
parameters <- c("s", "mu", "slope", "mean.r")

# MCMC settings
niter <- 10000
nthin <- 6
nburn <- 5000
nchains <- 3

# Call WinBUGS from R (BRT 2 min)
mr.trend <- bugs(bugs.data, inits, parameters, "mr-trend.bug", n.chains =
nchains, n.thin = nthin, n.iter = niter, n.burnin = nburn, debug = TRUE,
working.directory = getwd(), bugs.directory = bugs.dir)

# Inspect results
print(mr.trend, digits=3)
             mean    sd   2.5%    25%    50%    75%  97.5%  Rhat n.eff
s[1]        0.797 0.023  0.753  0.782  0.798  0.812  0.841 1.005   510
s[2]        0.789 0.021  0.746  0.774  0.789  0.803  0.830 1.006   440
s[3]        0.779 0.020  0.741  0.766  0.779  0.793  0.819 1.007   370
s[4]        0.770 0.019  0.734  0.757  0.770  0.782  0.807 1.008   310
s[5]        0.760 0.017  0.727  0.748  0.760  0.771  0.794 1.009   250
s[6]        0.750 0.016  0.719  0.739  0.750  0.760  0.781 1.011   200
s[7]        0.740 0.015  0.711  0.729  0.739  0.749  0.769 1.013   160
s[8]        0.729 0.014  0.702  0.720  0.729  0.738  0.758 1.016   130
s[9]        0.718 0.013  0.693  0.709  0.718  0.726  0.745 1.020   110
s[10]       0.707 0.013  0.681  0.698  0.706  0.715  0.734 1.023   100
s[11]       0.695 0.014  0.669  0.686  0.695  0.704  0.723 1.024   100
s[12]       0.683 0.015  0.656  0.673  0.683  0.693  0.714 1.023   110
s[13]       0.671 0.016  0.640  0.660  0.671  0.682  0.706 1.020   120
s[14]       0.659 0.019  0.624  0.646  0.659  0.671  0.697 1.017   150
s[15]       0.646 0.021  0.606  0.632  0.646  0.660  0.688 1.014   170
s[16]       0.634 0.024  0.587  0.617  0.634  0.649  0.681 1.012   200
s[17]       0.621 0.027  0.568  0.602  0.621  0.638  0.673 1.010   230
```

```
s[18]       0.608 0.030   0.547   0.587   0.608   0.627   0.666 1.009   270
s[19]       0.594 0.033   0.527   0.572   0.595   0.616   0.659 1.007   300
s[20]       0.581 0.036   0.506   0.557   0.582   0.604   0.652 1.007   340
mu          0.880 0.063   0.760   0.836   0.878   0.919   1.017 1.022   100
slope      -0.055 0.014  -0.083  -0.064  -0.055  -0.046  -0.029 1.002  1700
mean.r      0.051 0.002   0.047   0.050   0.051   0.053   0.056 1.001  2500
```

Exercise 4

Task: Due to differential behavior, the recovery probability may show strong individual variation. Simulate mark-recovery data for a population with mean survival of 0.7 and a mean recovery probability of 0.2. The variance of the recovery probability among individuals is 0.7 (on the logit scale). Assume that the study lasts 10 years and that each year 100 individuals are released. Analyze the data with a) the data-generating model and b) with a model that assume a common recovery probability for all individuals. What is the impact on the estimate of the survival probability?

Solution: We simulate individual capture histories using function `simul.mr`. Because our analyzing model needs to include an individual random effect we have to analyze the data with the state-space likelihood. The model without individual heterogeneity in the recovery probability is very straightforward to write. For the model with individual heterogeneity, we specify that the recovery probability of each individual is generated from a normal distribution, whose mean and standard deviation we estimate.

<u>*Data simulation*</u>

```
n.occasions <- 10                  # Number of occasions
marked <- rep(100, n.occasions)    # Annual number of newly marked young
s <- rep(0.7, n.occasions)         # Survival probability
mean.r <- 0.2
v.ind <- 0.7
r <- plogis(rnorm(sum(marked), qlogis(mean.r), v.ind^0.5))

# Define matrices with the survival and recovery probabilities
S <- matrix(rep(s, sum(marked)), ncol = n.occasions, nrow = sum(marked),
byrow = TRUE)
R <- matrix(rep(r, n.occasions), ncol = n.occasions, nrow = sum(marked),
byrow = FALSE)

# Apply simulation function
MR <- simul.mr(S, R, marked)

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

<u>*Data analysis*</u>

a) Model with individual variation in recovery probability

```
# Specify model in BUGS language
sink("mr.indvar.bug")
cat("
model {

# Priors and constraints
for (i in 1:nind){
```

```
   for (t in 1:n.occasions){
      s[i,t] <- mean.s
      logit(r[i,t]) <- mu + epsilon[i]
      } #t
   epsilon[i] ~ dnorm(0, tau)I(-15,15)
   } #i

mean.s ~ dunif(0, 1)
mu <- log(mean.r / (1-mean.r))    # Logit transformation
mean.r ~ dunif(0, 1)
tau <- pow(sigma, -2)
sigma ~ dunif(0, 5)               # Prior on standard deviation
sigma2 <- pow(sigma, 2)

# 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] <- s[i,t-1] * z[i,t-1]
      # Observation process
      y[i,t] ~ dbern(mu2[i,t])
      mu2[i,t] <- r[i,t-1] * (z[i,t-1] - z[i,t])
      } #t
   } #i
}
",fill = TRUE)
sink()


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

# Initial values
inits <- function(){list(z = mr.init.z(MR), mean.s = runif(1, 0, 1), mean.r
= runif(1, 0, 1), sigma = runif(1, 0, 1))}


# Parameters monitored
parameters <- c("mean.s", "mean.r", "sigma2")

# MCMC settings
niter <- 50000
nthin <- 6
nburn <- 30000
nchains <- 3

# Call WinBUGS from R (BRT 115 min)
mr.indvar <- bugs(bugs.data, inits, parameters, "mr.indvar.bug", n.chains =
nchains, n.thin = nthin, n.iter = niter, n.burnin = nburn, debug = TRUE,
working.directory = getwd(), bugs.directory = bugs.dir)


# Inspect results
print(mr.indvar, digits = 3)
Inference for Bugs model at "mr.indvar.bug", fit using WinBUGS,
 3 chains, each with 50000 iterations (first 30000 discarded), n.thin = 6
 n.sims = 10002 iterations saved
            mean    sd  2.5%   25%   50%   75% 97.5%  Rhat n.eff
mean.s     0.723 0.030 0.663 0.703 0.723 0.742 0.782 1.033    67
```

```
mean.r      0.132   0.060   0.035    0.079    0.132    0.180    0.241 1.003  2300
sigma2      6.510   5.559   0.318    1.989    4.618   10.300   19.400 1.043   140
deviance 1191.993 140.558 963.200 1068.000 1189.000 1307.000 1450.000 1.006   560
```

Convergence is not so easily obtained. For a publication, we would probably run the chains for even longer.

b) Model without individual variation in recovery probability

```
# Specify model in BUGS language
sink("mr.bug")
cat("
model {

# Priors and constraints
for (i in 1:nind){
   for (t in 1:n.occasions){
      s[i,t] <- mean.s
      r[i,t] <- mean.r
      } #t
   } #i
   mean.s ~ dunif(0, 1)
   mean.r ~ dunif(0, 1)

# 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] <- s[i,t-1] * z[i,t-1]
      # Observation process
      y[i,t] ~ dbern(mu2[i,t])
      mu2[i,t] <- r[i,t-1] * (z[i,t-1] - z[i,t])
      } #t
   } #i
}
",fill = TRUE)
sink()

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

# Initial values
inits <- function(){list(z = mr.init.z(MR), mean.s = runif(1, 0, 1), mean.r
= runif(1, 0, 1))}

# Parameters monitored
parameters <- c("mean.s", "mean.r")

# MCMC settings
niter <- 20000
nthin <- 3
nburn <- 10000
nchains <- 3

# Call WinBUGS from R (BRT 25 min)
mr <- bugs(bugs.data, inits, parameters, "mr.bug", n.chains = nchains,
n.thin = nthin, n.iter = niter, n.burnin = nburn, debug = TRUE,
working.directory = getwd(), bugs.directory = bugs.dir)
```

```
# Inspect results
print(mr, digits = 3)
Inference for Bugs model at "mr.bug", fit using WinBUGS,
 3 chains, each with 20000 iterations (first 10000 discarded), n.thin = 3
 n.sims = 10002 iterations saved
             mean    sd     2.5%      25%      50%      75%    97.5%  Rhat n.eff
mean.s      0.721 0.028    0.666    0.702    0.720    0.740    0.775 1.011   200
mean.r      0.245 0.019    0.210    0.232    0.244    0.258    0.285 1.005   470
deviance 1486.163 6.899 1472.000 1482.000 1486.000 1491.000 1499.000 1.003   840
```

The estimates of the survival probabilities from the analyses with and without the individual random effect on recovery probability are very similar. Thus, it appears as if unmodeled individual heterogeneity in recovery probability has no effect on the survival estimate. However, to see whether this result generally holds, we would have to conduct a simulation study with many repetitions.

Chapter 9

Exercise 1

Task: Simulate multistate capture-recapture data for two sexes (m, f) in two populations (A, B) which are connected by dispersal. Assume that movement rates between populations are the same for both sexes, but that site-specific survival and recapture differ among populations. The simulation parameters are: $\phi_{A,m} = 0.5$, $\phi_{B,m} = 0.6$, $\phi_{A,f} = 0.7$, $\phi_{B,f} = 0.6$, $\psi_{AB} = 0.2$, $\psi_{BA} = 0.5$, $p_{A,m} = 0.3$, $p_{B,m} = 0.7$, $p_{A,f} = 0.4$, $p_{B,f} = 0.8$, 6 occasions, and 20 males and females are released at each population in each year. Simulate the data and analyze them.

Solution: We simulate individual multistate capture histories for males and for females using function `simul.ms`. We stag the two data sets and create a grouping variable indicating for each individual to which group it belongs. Next, we set up the multistate model. As always when data shall be analyzed with a multistate model, we should define all true and observed states as well as the state-transition and the observation matrix. These matrices are then included in the analyzing code. The required model is fairly standard (see Section 9.2.2 of the book). A minor difficulty may be that we have to apply a linear model for each parameter type in such a way that a separate estimate is obtained for the two sexes. This is done by using the grouping variable as an index.

Data simulation

```
# Define mean survival, transitions, recapture, as well as number of
occasions, states, observations and released individuals
phiAm <- 0.5
phiBm <- 0.6
pAm <- 0.3
pBm <- 0.7
phiAf <- 0.7
phiBf <- 0.6
pAf <- 0.4
pBf <- 0.8
psiAB <- 0.2
psiBA <- 0.5
n.occasions <- 6
n.states <- 3
n.obs <- 3
marked <- matrix(NA, ncol = n.states, nrow = n.occasions)
marked[,1] <- rep(20, n.occasions)
marked[,2] <- rep(20, n.occasions)
marked[,3] <- rep(0, n.occasions)


# Simulate male data
# Define arrays with survival, transition and recapture probabilities
# These are 4-dimensional arrays, with
   # Dimension 1: state of departure
   # Dimension 2: state of arrival
   # Dimension 3: individual
   # Dimension 4: time
# 1. State process array
totrel <- sum(marked)
PSI.STATE <- array(NA, dim = c(n.states, n.states, totrel, n.occasions-1))
for (i in 1:totrel){
```

```
   for (t in 1:(n.occasions-1)){
      PSI.STATE[,,i,t] <- matrix(c(
      phiAm*(1-psiAB), phiAm*psiAB,     1-phiAm,
      phiBm*psiBA,     phiBm*(1-psiBA), 1-phiBm,
      0,               0,               1       ), nrow = n.states, byrow = 
TRUE)
      } #t
   } #i

# 2. Observation array
PSI.OBS <- array(NA, dim = c(n.states, n.obs, totrel, n.occasions-1))
for (i in 1:totrel){
   for (t in 1:(n.occasions-1)){
      PSI.OBS[,,i,t] <- matrix(c(
      pAm, 0,   1-pAm,
      0,   pBm, 1-pBm,
      0,   0,   1       ), nrow = n.states, byrow = TRUE)
      } #t
   } #i


# Execute simulation function
sim <- simul.ms(PSI.STATE, PSI.OBS, marked)
CHm <- sim$CH

# Simulate female data
# Define arrays with survival, transition and recapture probabilities
# 1. State process array
totrel <- sum(marked)
PSI.STATE <- array(NA, dim = c(n.states, n.states, totrel, n.occasions-1))
for (i in 1:totrel){
   for (t in 1:(n.occasions-1)){
      PSI.STATE[,,i,t] <- matrix(c(
      phiAf*(1-psiAB), phiAf*psiAB,     1-phiAf,
      phiBf*psiBA,     phiBf*(1-psiBA), 1-phiBf,
      0,               0,               1       ), nrow = n.states, byrow = 
TRUE)
      } #t
   } #i

# 2. Observation array
PSI.OBS <- array(NA, dim = c(n.states, n.obs, totrel, n.occasions-1))
for (i in 1:totrel){
   for (t in 1:(n.occasions-1)){
      PSI.OBS[,,i,t] <- matrix(c(
      pAf, 0,   1-pAf,
      0,   pBf, 1-pBf,
      0,   0,   1       ), nrow = n.states, byrow = TRUE)
      } #t
   } #i

# Execute simulation function
sim <- simul.ms(PSI.STATE, PSI.OBS, marked)
CHf <- sim$CH

# Compute vector with occasion of first capture
get.first <- function(x) min(which(x!=0))
fm <- apply(CHm, 1, get.first)
ff <- apply(CHf, 1, get.first)

# Recode CH matrix: note, a 0 is not allowed!
# 1 = seen alive in A, 2 = seen alive in B, 3 = not seen
```

```
rCHm <- CHm  # recoded CH
rCHm[rCHm==0] <- 3
rCHf <- CHf  # recoded CH
rCHf[rCHf==0] <- 3

# Combine data sets
rCH <- rbind(rCHm, rCHf)

# Create group variable
group <- c(rep(1, nrow(rCHm)), rep(2, nrow(rCHf)))

# Compute vector with occasion of first capture
get.first <- function(x) min(which(x!=3))
f <- apply(rCH, 1, get.first)
```

Data analysis

```
# Specify model in BUGS language
sink("ms.bug")
cat("
model {
####################################################
# Parameters:
# phiA: survival probability at site A
# phiB: survival probability at site B
# psiAB: movement probability from site A to site B
# psiBA: movement probability from site B to site A
# pA: recapture probability at site A
# pB: recapture probability at site B
###################################################
# States (S)
# 1 alive at A
# 2 alive at B
# 3 dead
# Observations (O)
# 1 seen at A
# 2 seen at B
# 3 not seen
###################################################

# Priors and constraints
for (i in 1: nind){
   for (t in 1:(n.occasions-1)){
      phiA[i,t] <- mean.phiA[group[i]]
      phiB[i,t] <- mean.phiB[group[i]]
      psiAB[i,t] <- mean.psi[1]
      psiBA[i,t] <- mean.psi[2]
      pA[i,t] <- mean.pA[group[i]]
      pB[i,t] <- mean.pB[group[i]]
      } #t
   } #i

for (u in 1:2){
   mean.phiA[u] ~ dunif(0, 1) # Priors for mean state-spec. survival (at A)
   mean.phiB[u] ~ dunif(0, 1) # Priors for mean state-spec. survival (at B)
   mean.psi[u] ~ dunif(0, 1)  # Priors for mean transitions
   mean.pA[u] ~ dunif(0, 1)  # Priors for mean state-spec. recapture (at A)
   mean.pB[u] ~ dunif(0, 1)  # Priors for mean state-spec. recapture (at B)
   }

# Define state and observation matrices
for (i in 1:nind){
```

```
   # Define probabilities of state S(t+1) given S(t)
   for (t in f[i]:(n.occasions-1)){
      ps[1,i,t,1] <- phiA[i,t] * (1-psiAB[i,t])
      ps[1,i,t,2] <- phiA[i,t] * psiAB[i,t]
      ps[1,i,t,3] <- 1-phiA[i,t]
      ps[2,i,t,1] <- phiB[i,t] * psiBA[i,t]
      ps[2,i,t,2] <- phiB[i,t] * (1-psiBA[i,t])
      ps[2,i,t,3] <- 1-phiB[i,t]
      ps[3,i,t,1] <- 0
      ps[3,i,t,2] <- 0
      ps[3,i,t,3] <- 1

      # Define probabilities of O(t) given S(t)
      po[1,i,t,1] <- pA[i,t]
      po[1,i,t,2] <- 0
      po[1,i,t,3] <- 1-pA[i,t]
      po[2,i,t,1] <- 0
      po[2,i,t,2] <- pB[i,t]
      po[2,i,t,3] <- 1-pB[i,t]
      po[3,i,t,1] <- 0
      po[3,i,t,2] <- 0
      po[3,i,t,3] <- 1
      } #t
   } #i

# Likelihood
for (i in 1:nind){
   # Define latent state at first capture
   z[i,f[i]] <- y[i,f[i]]
   for (t in (f[i]+1):n.occasions){
      # State process: draw S(t) given S(t-1)
      z[i,t] ~ dcat(ps[z[i,t-1], i, t-1,])
      # Observation process: draw O(t) given S(t)
      y[i,t] ~ dcat(po[z[i,t], i, t-1,])
      } #t
   } #i
}
",fill = TRUE)
sink()


# Bundle data
bugs.data <- list(y = rCH, group = group, f = f, n.occasions = dim(rCH)[2],
nind = dim(rCH)[1], z = known.state.ms(rCH, 3))

# Initial values
inits <- function(){list(mean.phiA = runif(2, 0, 1), mean.phiB = runif(2,
0, 1), mean.psi = runif(2, 0, 1), mean.pA = runif(2, 0, 1), mean.pB =
runif(2, 0, 1), z = ms.init.z(rCH, f))}

# Parameters monitored
parameters <- c("mean.phiA", "mean.phiB", "mean.psi", "mean.pA", "mean.pB")

# MCMC settings
ni <- 10000
nt <- 1
nb <- 5000
nc <- 3

# Call WinBUGS from R (BRT 7 min)
```

```
ms <- bugs(bugs.data, inits, parameters, "ms.bug", n.chains = nc, n.thin =
nt, n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory = bugs.dir,
working.directory = getwd())

print(ms, 3)
Inference for Bugs model at "ms.bug", fit using WinBUGS,
 3 chains, each with 10000 iterations (first 5000 discarded)
 n.sims = 15000 iterations saved
                mean     sd    2.5%      25%      50%      75%    97.5%  Rhat n.eff
mean.phiA[1]    0.520  0.073  0.372    0.472    0.521    0.569    0.664 1.005  3300
mean.phiA[2]    0.689  0.055  0.581    0.652    0.688    0.724    0.803 1.003  1900
mean.phiB[1]    0.714  0.091  0.552    0.650    0.708    0.772    0.911 1.011   200
mean.phiB[2]    0.557  0.070  0.426    0.508    0.555    0.604    0.695 1.003   800
mean.psi[1]     0.266  0.098  0.139    0.195    0.243    0.313    0.526 1.017   240
mean.psi[2]     0.521  0.117  0.285    0.438    0.528    0.612    0.721 1.018   240
mean.pA[1]      0.302  0.114  0.160    0.225    0.276    0.349    0.609 1.020   310
mean.pA[2]      0.446  0.101  0.302    0.378    0.428    0.492    0.706 1.017   320
mean.pB[1]      0.633  0.156  0.370    0.513    0.619    0.746    0.950 1.007   470
mean.pB[2]      0.582  0.191  0.264    0.433    0.561    0.725    0.961 1.015   180
deviance     1050.783 32.126 981.300 1031.000 1053.000 1073.000 1108.000 1.006   480
```

Exercise 2

Task: Simulate multistate capture-recapture data from two populations observed over 8 years that exchange individuals with the following parameter values: ϕ_A = [0.5, 0.6, 0.3, 0.7, 0.5, 0.65, 0.55], ϕ_B = 0.6, ψ_{AB} = 0.2, ψ_{BA} = 0.5, p_A = 0.3, p_B = 0.7, at each population 20 individuals are released each year. Thus we assume that survival probabilities vary among years at location A, but not at B. Simulate data and analyze them, a) assuming fixed year effects and b) assuming random year effects.

Solution: We first simulate multistate capture histories using function `simul.ms`. The definitions of the true and observed states as well as the transition matrices are identical as in exercise 1. The specification of ϕ_A with fixed year effects requires that we define a prior distribution for each annual value, whereas the specification of ϕ_A with random year effects requires a prior of the mean and of the standard deviation of the normal distribution from which the annual values are generated.

Data simulation

```
# Define mean survival, transitions, recapture, as well as number of
occasions, states, observations and released individuals
phiA <- c(0.5, 0.6, 0.3, 0.7, 0.5, 0.65, 0.55)
phiB <- 0.6
pA <- 0.3
pB <- 0.7
psiAB <- 0.2
psiBA <- 0.5
n.occasions <- 8
n.states <- 3
n.obs <- 3
marked <- matrix(NA, ncol = n.states, nrow = n.occasions)
marked[,1] <- rep(20, n.occasions)
marked[,2] <- rep(20, n.occasions)
marked[,3] <- rep(0, n.occasions)


# Define arrays with survival, transition and recapture probabilities
# These are 4-dimensional arrays, with
   # Dimension 1: state of departure
```

```
   # Dimension 2: state of arrival
   # Dimension 3: individual
   # Dimension 4: time
# 1. State process array
totrel <- sum(marked)
PSI.STATE <- array(NA, dim = c(n.states, n.states, totrel, n.occasions-1))
for (i in 1:totrel){
   for (t in 1:(n.occasions-1)){
      PSI.STATE[,,i,t] <- matrix(c(
      phiA[t]*(1-psiAB), phiA[t]*psiAB,     1-phiA[t],
      phiB*psiBA,        phiB*(1-psiBA),    1-phiB,
      0,                 0,                 1       ), nrow = n.states,
byrow = TRUE)
      } #t
   } #i

# 2. Observation array
PSI.OBS <- array(NA, dim = c(n.states, n.obs, totrel, n.occasions-1))
for (i in 1:totrel){
   for (t in 1:(n.occasions-1)){
      PSI.OBS[,,i,t] <- matrix(c(
      pA, 0,  1-pA,
      0,  pB, 1-pB,
      0,  0,  1       ), nrow = n.states, byrow = TRUE)
      } #t
   } #i

# Execute simulation function
sim <- simul.ms(PSI.STATE, PSI.OBS, marked)
CH <- sim$CH

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

# Recode CH matrix: note, a 0 is not allowed!
# 1 = seen alive in A, 2 = seen alive in B, 3 = not seen
rCH <- CH  # recoded CH
rCH[rCH==0] <- 3
```

<u>Data analysis</u>

a) Fixed time effects

```
# Specify model in BUGS language
sink("ms.bug")
cat("
model {
####################################################
# Parameters:
# phiA: survival probability at site A
# phiB: survival probability at site B
# psiAB: movement probability from site A to site B
# psiBA: movement probability from site B to site A
# pA: recapture probability at site A
# pB: recapture probability at site B
###################################################
# States (S)
# 1 alive at A
# 2 alive at B
# 3 dead
# Observations (O)
# 1 seen at A
```

```
# 2 seen at B
# 3 not seen
####################################################

# Priors and constraints
for (t in 1:(n.occasions-1)){
   phiA[t] ~ dunif(0, 1)
   phiB[t] <- mean.phiB
   psiAB[t] <- mean.psi[1]
   psiBA[t] <- mean.psi[2]
   pA[t] <- mean.p[1]
   pB[t] <- mean.p[2]
   }

mean.phiB ~ dunif(0, 1)
for (u in 1:2){
   mean.psi[u] ~ dunif(0, 1)
   mean.p[u] ~ dunif(0, 1)
   }

# Define parameters
for (i in 1:nind){
   # Define probabilities of state S(t+1) given S(t)
   for (t in f[i]:(n.occasions-1)){  # loop over time
      # First index = states at time t-1, last index = states at time t
      ps[1,i,t,1] <- phiA[t] * (1-psiAB[t])
      ps[1,i,t,2] <- phiA[t] * psiAB[t]
      ps[1,i,t,3] <- 1-phiA[t]
      ps[2,i,t,1] <- phiB[t] * psiBA[t]
      ps[2,i,t,2] <- phiB[t] * (1-psiBA[t])
      ps[2,i,t,3] <- 1-phiB[t]
      ps[3,i,t,1] <- 0
      ps[3,i,t,2] <- 0
      ps[3,i,t,3] <- 1

      # Define probabilities of O(t) given S(t)
      # First index = states at time t, last index = observations at time t
      po[1,i,t,1] <- pA[t]
      po[1,i,t,2] <- 0
      po[1,i,t,3] <- 1-pA[t]
      po[2,i,t,1] <- 0
      po[2,i,t,2] <- pB[t]
      po[2,i,t,3] <- 1-pB[t]
      po[3,i,t,1] <- 0
      po[3,i,t,2] <- 0
      po[3,i,t,3] <- 1
      } #t
   } #i

# State-space model likelihood
for (i in 1:nind){
   z[i,f[i]] <- y[i,f[i]]
   for (t in (f[i]+1):n.occasions){  # loop over time
      # State process: draw S(t) given S(t-1)
      z[i,t] ~ dcat(ps[z[i,t-1], i, t-1,])
      # Observation process: draw O(t) given S(t)
      y[i,t] ~ dcat(po[z[i,t], i, t-1,])
      } #t
   } # i
}
",fill = TRUE)
sink()
```

```
# Bundle data
bugs.data <- list(y = rCH, f = f, n.occasions = dim(rCH)[2], nind =
dim(rCH)[1], z = known.state.ms(rCH, 3))

# Initial values
inits <- function(){list(phiA = runif(n.occasions-1, 0, 1), mean.phiB =
runif(1, 0, 1), mean.psi = runif(2, 0, 1), mean.p = runif(2, 0, 1), z =
ms.init.z(rCH, f))}

# Parameters monitored
parameters <- c("phiA", "mean.phiB", "mean.psi", "mean.p")

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


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

# Inspect results
print(msf, 3)
Inference for Bugs model at "ms.bug", fit using WinBUGS,
 3 chains, each with 1000 iterations (first 500 discarded), n.thin = 3
 n.sims = 501 iterations saved
                mean     sd    2.5%     25%     50%     75%   97.5%  Rhat n.eff
phiA[1]        0.684  0.170   0.345   0.567   0.685   0.814   0.971 1.003   610
phiA[2]        0.462  0.160   0.207   0.349   0.446   0.559   0.848 1.003   590
phiA[3]        0.403  0.134   0.178   0.303   0.390   0.487   0.689 1.012   160
phiA[4]        0.552  0.149   0.284   0.448   0.535   0.656   0.861 1.003   650
phiA[5]        0.677  0.154   0.384   0.569   0.678   0.787   0.967 1.015   130
phiA[6]        0.506  0.139   0.264   0.408   0.495   0.591   0.817 1.022    95
phiA[7]        0.488  0.176   0.207   0.363   0.473   0.594   0.913 1.005   560
mean.phiB      0.642  0.071   0.514   0.591   0.636   0.689   0.781 1.012   160
mean.psi[1]    0.197  0.059   0.106   0.155   0.187   0.230   0.338 1.013   160
mean.psi[2]    0.446  0.101   0.260   0.377   0.443   0.515   0.650 1.003   660
mean.p[1]      0.275  0.069   0.165   0.226   0.267   0.315   0.430 1.008   230
mean.p[2]      0.668  0.118   0.442   0.589   0.663   0.735   0.938 1.013   160
deviance     680.514 23.959 632.632 663.950 681.800 696.975 724.190 1.013   150
```

b) Random time effects

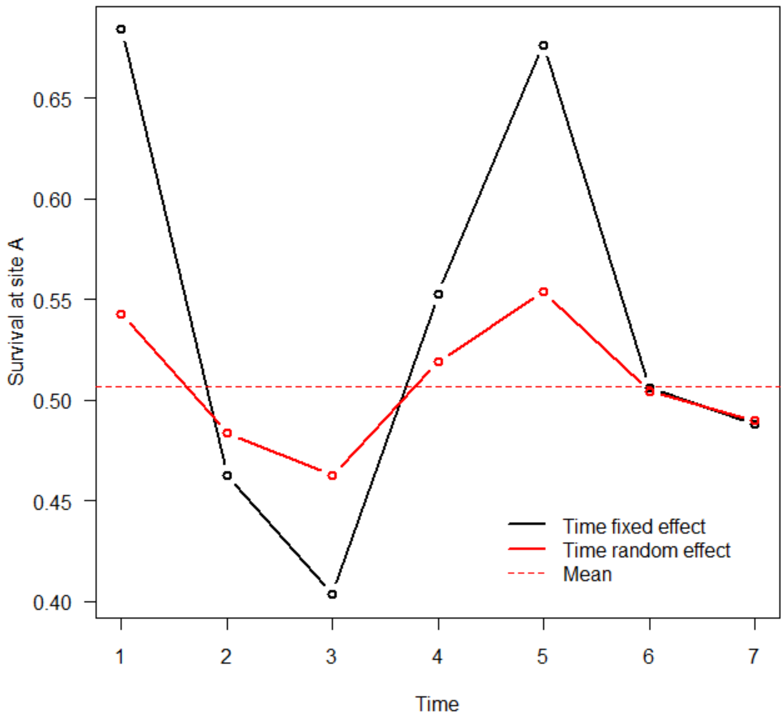
```
# Specify model in BUGS language
sink("msrand.bug")
cat("
model {
####################################################
# Parameters:
# phiA: survival probability at site A
# phiB: survival probability at site B
# psiAB: movement probability from site A to site B
# psiBA: movement probability from site B to site A
# pA: recapture probability at site A
# pB: recapture probability at site B
###################################################
# States (S)
# 1 alive at A
```

```
# 2 alive at B
# 3 dead
# Observations (O)
# 1 seen at A
# 2 seen at B
# 3 not seen
###################################################

# Priors and constraints
for (t in 1:(n.occasions-1)){
   logit(phiA[t]) <- mu + epsilon[t]
   epsilon[t] ~ dnorm(0, tau)I(-15,15)
   phiB[t] <- mean.phiB
   psiAB[t] <- mean.psi[1]
   psiBA[t] <- mean.psi[2]
   pA[t] <- mean.p[1]
   pB[t] <- mean.p[2]
   }

mean.phiB ~ dunif(0, 1)
for (u in 1:2){
   mean.psi[u] ~ dunif(0, 1)
   mean.p[u] ~ dunif(0, 1)
   }

mu <- log(mean.phiA / (1- mean.phiA ))  # Logit transformation
mean.phiA ~ dunif(0, 1)                 # Prior for mean survival
tau <- pow(sigma, -2)
sigma ~ dunif(0, 10)                    # Prior on standard deviation
sigma2 <- pow(sigma, 2)                 # Temporal variance

# Define parameters
for (i in 1:nind){
   # Define probabilities of state S(t+1) given S(t)
   for (t in f[i]:(n.occasions-1)){  # loop over time
      # First index = states at time t-1, last index = states at time t
      ps[1,i,t,1] <- phiA[t] * (1-psiAB[t])
      ps[1,i,t,2] <- phiA[t] * psiAB[t]
      ps[1,i,t,3] <- 1-phiA[t]
      ps[2,i,t,1] <- phiB[t] * psiBA[t]
      ps[2,i,t,2] <- phiB[t] * (1-psiBA[t])
      ps[2,i,t,3] <- 1-phiB[t]
      ps[3,i,t,1] <- 0
      ps[3,i,t,2] <- 0
      ps[3,i,t,3] <- 1

      # Define probabilities of O(t) given S(t)
      # First index = states at time t, last index = observations at time t
      po[1,i,t,1] <- pA[t]
      po[1,i,t,2] <- 0
      po[1,i,t,3] <- 1-pA[t]
      po[2,i,t,1] <- 0
      po[2,i,t,2] <- pB[t]
      po[2,i,t,3] <- 1-pB[t]
      po[3,i,t,1] <- 0
      po[3,i,t,2] <- 0
      po[3,i,t,3] <- 1
      } #t
   } #i

# State-space model likelihood
for (i in 1:nind){
```

```
    z[i,f[i]] <- y[i,f[i]]
    for (t in (f[i]+1):n.occasions){  # loop over time
       # State process: draw S(t) given S(t-1)
       z[i,t] ~ dcat(ps[z[i,t-1], i, t-1,])
       # Observation process: draw O(t) given S(t)
       y[i,t] ~ dcat(po[z[i,t], i, t-1,])
       } #t
    } # i
}
",fill = TRUE)
sink()


# Bundle data
bugs.data <- list(y = rCH, f = f, n.occasions = dim(rCH)[2], nind =
dim(rCH)[1], z = known.state.ms(rCH, 3))

# Initial values
inits <- function(){list(mean.phiA = runif(1, 0, 1), sigma = runif(1, 0,
1), mean.phiB = runif(1, 0, 1), mean.psi = runif(2, 0, 1), mean.p =
runif(2, 0, 1), z = ms.init.z(rCH, f))}

# Parameters monitored
parameters <- c("phiA", "mean.phiA", "sigma2", "mean.phiB", "mean.psi",
"mean.p")

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


# Call WinBUGS from R (BRT 8 min)
ms <- bugs(bugs.data, inits, parameters, "msrand.bug", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory =
bugs.dir, working.directory = getwd())

# Inspect results
print(ms, 3)
Inference for Bugs model at "msrand.bug", fit using WinBUGS,
 3 chains, each with 10000 iterations (first 5000 discarded), n.thin = 3
 n.sims = 5001 iterations saved
               mean     sd    2.5%     25%     50%     75%   97.5%  Rhat n.eff
phiA[1]       0.543  0.111   0.357   0.470   0.529   0.599   0.810 1.002  1700
phiA[2]       0.484  0.100   0.275   0.422   0.486   0.544   0.684 1.001  5000
phiA[3]       0.463  0.095   0.255   0.405   0.468   0.526   0.640 1.003   770
phiA[4]       0.519  0.091   0.351   0.460   0.514   0.570   0.719 1.001  3700
phiA[5]       0.554  0.100   0.391   0.486   0.540   0.609   0.789 1.001  2800
phiA[6]       0.504  0.090   0.328   0.449   0.504   0.558   0.696 1.003   960
phiA[7]       0.490  0.103   0.284   0.426   0.491   0.550   0.709 1.002  5000
mean.phiA     0.507  0.075   0.360   0.458   0.506   0.551   0.664 1.002  5000
sigma2        0.346  1.010   0.001   0.024   0.106   0.335   2.022 1.001  3800
mean.phiB     0.628  0.062   0.515   0.586   0.624   0.668   0.758 1.001  5000
mean.psi[1]   0.218  0.068   0.115   0.170   0.206   0.254   0.384 1.005   460
mean.psi[2]   0.425  0.096   0.255   0.358   0.419   0.488   0.625 1.007   310
mean.p[1]     0.304  0.076   0.182   0.250   0.296   0.347   0.477 1.003   960
mean.p[2]     0.653  0.113   0.455   0.573   0.646   0.723   0.900 1.009   260
deviance    675.103 21.997 630.700 660.900 675.900 689.700 716.800 1.001  3100
```

The parameter estimates from both analyses are similar. When we look at the estimates of the survival probabilities at site A, we can see the shrinkage of the individual estimates towards the mean in the analysis where time is treated as a random effect:

```
plot(msf$mean$phiA, type = "b", las = 1, ylab = "Survival at site A", xlab
= "Time", lwd = 2)
points(ms$mean$phiA, type="b", col="red", lwd = 2)
abline(h = ms$mean$mean.phiA, lty = 2, col = "red")
legend(x = 4.5, y = 0.45, legend = c("Time fixed effect", "Time random
effect", "Mean"), lty = c(1,1,2), col = c("black", "red", "red"), bty =
"n", lwd = c(2, 2, 1))
```



More discussion about shrinkage can be found on pages 80 and 377 of the BPA book.

Exercise 3

Task: In a population of salamanders there is non-random temporary emigration (with respect to one breeding site). In addition there is strong individual heterogeneity in capture probability. Assume a 10-years study and the following parameter values: survival = 0.7, ψ_{IO} = 0.4, ψ_{OI} = 0.8, mean recapture = 0.5, and the variance among individuals of the logit of recapture σ_i^2 = 0.4. Further assume that 100 salamanders are newly marked each year. Simulate data with these characteristics and analyze them.

Solution: We simulate multistate capture histories using function `simul.ms`. To anylse the data, we first define the true and the observed states as well as the state-transition and the observation matrices (see Section 9.3. of the BPA book for this model). We then define these matrices in BUGS with the corresponding parameters and use GLM or GLMM for each parameter to impose the structure of the model we want to fit.

Data simulation

```
# Define mean survival, transitions, recapture, as well as number of
occasions, states, observations and released individuals
phi <- 0.7
psiIO <- 0.4
psiOI <- 0.8
mean.p <- 0.5
v.ind <- 0.4
n.occasions <- 10
n.states <- 3
n.obs <- 2
marked <- matrix(NA, ncol = n.states, nrow = n.occasions)
marked[,1] <- rep(100, n.occasions) # present
marked[,2] <- rep(0, n.occasions)   # absent
marked[,3] <- rep(0, n.occasions)   # dead

# Draw individual recapture probabilities
logit.p <- rnorm(sum(marked), qlogis(mean.p), v.ind^0.5)
p <- plogis(logit.p)

# Define arrays with survival, transition and recapture probabilities
# These are 4-dimensional arrays, with
   # Dimension 1: state of departure
   # Dimension 2: state of arrival
   # Dimension 3: individual
   # Dimension 4: time
# 1. State process array
totrel <- sum(marked)
PSI.STATE <- array(NA, dim = c(n.states, n.states, totrel, n.occasions-1))
for (i in 1:totrel){
   for (t in 1:(n.occasions-1)){
      PSI.STATE[,,i,t] <- matrix(c(
      phi*(1-psiIO), phi*psiIO,     1-phi,
      phi*psiOI,     phi*(1-psiOI), 1-phi,
      0,               0,               1        ), nrow = n.states, byrow =
TRUE)
      } #t
   } #i

# 2.Observation array
PSI.OBS <- array(NA, dim = c(n.states, n.obs, totrel, n.occasions-1))
for (i in 1:totrel){
   for (t in 1:(n.occasions-1)){
      PSI.OBS[,,i,t] <- matrix(c(
      p[i], 1-p[i],
      0, 1,
      0, 1     ), nrow = n.states, byrow = TRUE)
      } #t
   } #i

# Execute simulation function
sim <- simul.ms(PSI.STATE, PSI.OBS, marked)
CH <- sim$CH

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

# Recode CH matrix: note, a 0 is not allowed!
# 1 = seen alive, 2 = not seen
```

```
rCH <- CH  # recoded CH
rCH[rCH==0] <- 2
```

Data analysis

```
# Specify model in BUGS language
sink("tempemi.bug")
cat("
model {
####################################################
# Parameters:
# phi: survival probability
# psiIO: probability to emigrate
# psiOI: probability to immigrate
# p: recapture probability
##################################
# States (S)
# 1 alive and present
# 2 alive and absent
# 3 dead
# Observations (O)
# 1 seen
# 2 not seen
##################################

# Priors and constraints
for (t in 1:(n.occasions-1)){
   phi[t] <- mean.phi
   psiIO[t] <- mean.psiIO
   psiOI[t] <- mean.psiOI
   }
mean.phi ~ dunif(0, 1)
mean.psiIO ~ dunif(0, 1)
mean.psiOI ~ dunif(0, 1)

for (i in 1:nind){
   for (t in 1:(n.occasions-1)){
      logit(p[i,t]) <- mu + epsilon[i]
      } #t
   epsilon[i] ~ dnorm(0, tau)I(-15,15)
   } # i

mu <- log(mean.p / (1-mean.p))    # Logit transformation
mean.p ~ dunif(0, 1)              # Prior for mean recapture
tau <- pow(sigma, -2)
sigma ~ dunif(0, 5)               # Prior for standard deviation
sigma2 <-pow(sigma, 2)


# Define parameters
for (i in 1:nind){
   # Define probabilities of state S(t+1) given S(t)
   for (t in f[i]:(n.occasions-1)){  # loop over time
      ps[1,i,t,1] <- phi[t] * (1-psiIO[t])
      ps[1,i,t,2] <- phi[t] * psiIO[t]
      ps[1,i,t,3] <- 1-phi[t]
      ps[2,i,t,1] <- phi[t] * psiOI[t]
      ps[2,i,t,2] <- phi[t] * (1-psiOI[t])
      ps[2,i,t,3] <- 1-phi[t]
      ps[3,i,t,1] <- 0
      ps[3,i,t,2] <- 0
      ps[3,i,t,3] <- 1
```

```
      # Define probabilities of O(t) given S(t)
      po[1,i,t,1] <- p[i,t]
      po[1,i,t,2] <- 1-p[i,t]
      po[2,i,t,1] <- 0
      po[2,i,t,2] <- 1
      po[3,i,t,1] <- 0
      po[3,i,t,2] <- 1
      } #t
   } #i

# State-space model likelihood
for (i in 1:nind){
   z[i,f[i]] <- y[i,f[i]]
   for (t in (f[i]+1):n.occasions){  # loop over time
      # State process: draw S(t) given S(t-1)
      z[i,t] ~ dcat(ps[z[i,t-1], i, t-1,])
      # Observation process: draw O(t) given S(t)
      y[i,t] ~ dcat(po[z[i,t], i, t-1,])
      } #t
   } # i
}
",fill = TRUE)
sink()


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

# Initial values
inits <- function(){list(mean.phi = runif(1, 0, 1), mean.psiIO = runif(1,
0, 1), mean.psiOI = runif(1, 0, 1), mean.p = runif(1, 0, 1), sigma =
runif(1, 0, 1), z = ms.init.z(rCH, f))}

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

# MCMC settings
ni <- 30000
nt <- 3
nb <- 20000
nc <- 3

# Call WinBUGS from R (BRT 113 min)
tempemi <- bugs(bugs.data, inits, parameters, "tempemi.bug", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory =
bugs.dir, working.directory = getwd())


print(tempemi, 3)
Inference for Bugs model at "tempemi.bug", fit using WinBUGS,
 3 chains, each with 30000 iterations (first 20000 discarded), n.thin = 3
 n.sims = 10002 iterations saved
               mean      sd     2.5%      25%      50%      75%    97.5%  Rhat n.eff
mean.phi      0.715   0.021    0.674    0.702    0.716    0.729    0.754 1.011   320
mean.psiIO    0.584   0.064    0.413    0.561    0.599    0.626    0.666 1.046    89
mean.psiOI    0.542   0.097    0.406    0.477    0.521    0.583    0.800 1.029   100
mean.p        0.784   0.142    0.464    0.689    0.818    0.898    0.972 1.032    81
sigma2        8.663   5.884    0.644    3.231    8.316   13.167   20.390 1.090    34
deviance   1701.742 160.841 1488.000 1580.000 1657.000 1796.000 2077.975 1.064    43
```

The model needs relatively long chains to reach convergence (in a real data analysis we would run the model even longer than here). While most parameter estimates are fairly close to the values of the data generating parameters, this is not the case for the individual variance. Variances in general are difficult to estimate and from the analysis of a single data set we cannot say anything about possible bias in this estimate. To formally study how well the model performs to estimate the parameters, we would have to conduct a proper simulation study.

Chapter 10

Exercise 1

Task: Simulate capture-recapture data of a species for males and females. The study is conducted for 8 years; the mean survival of males is 0.75 that of females 0.5 and capture is for both 0.4. The entry probability after the first occasion is 0.1 in both sexes. The size of the superpopulation is 300 in both sexes. Simulate corresponding one data set and analyze it with the model (ϕ_{sex}, b_t, p).

Solution: We simulate individual capture histories of males and females using the function `simul.js`. The entry probability at the first occasion must be calculated such that all entry probabilities sum to 1 (i.e. 1-7*0.1 = 0.3). We fit the JS model formulated as restricted occupancy model, but of course other choices are also possible. It requires that we augment the data with pseudo capture histories. We first augment the data for the males, then those for the females and finally stack them on top of each other using `rbind`. We define the indicator variable "group" for each sex. Note that the indicator variable needs to be defined for the complete (i.e. augmented) data set. In the analyzing model, we use GLM formulations to impose the model structure we would like to have, which is straightforwrad. Care must be taken for the computation of the derived parameters, since the first part (1 to the last male) of the latent variable *z* belongs to the first group and the second part (last male plus 1 to end) to the second group.

Data simulation

```
# Define the parameters
n.occasions <- 8
N <- 300
phi.m <- rep(0.75, n.occasions-1)
phi.f <- rep(0.5, n.occasions-1)
b <- c(0.3, rep(0.1, n.occasions-1))
p <- rep(0.4, n.occasions)

PHI.M <- matrix(rep(phi.m, (n.occasions-1)*N), ncol = n.occasions-1, nrow =
N, byrow = T)
PHI.F <- matrix(rep(phi.f, (n.occasions-1)*N), ncol = n.occasions-1, nrow =
N, byrow = T)
P <- matrix(rep(p, n.occasions*N), ncol = n.occasions, nrow = N, byrow = T)

# Apply simulation function
sim.m <- simul.js(PHI.M, P, b, N)
sim.f <- simul.js(PHI.F, P, b, N)
CH.m <- sim.m$CH
CH.f <- sim.f$CH
```

Data analysis

```
# Specify model in BUGS language
sink("js-rest.occ.bug")
cat("
model {
# Priors and constraints
for (i in 1:M){
   for (t in 1:(n.occasions-1)){
      phi[i,t] <- beta[group[i]]
```

```
      } # t
   for (t in 1:n.occasions){
      p[i,t] <- mean.p
      } #t
   } #i
for (i in 1:2){
   beta[i] ~ dunif(0, 1)
   }
mean.p ~ dunif(0, 1)

for (t in 1:n.occasions){
   gamma[t] ~ dunif(0, 1)
   } #t

# Define the likelihoods
for (i in 1:M){
   # First occasion
   # State process
   z[i,1] ~ dbern(gamma[1])
   mu1[i] <- z[i,1] * p[i,1]
   # Observation process
   y[i,1] ~ dbern(mu1[i])

   # Subsequent occasions
   for (t in 2:n.occasions){
      # State process
      q[i,t-1] <- 1-z[i,t-1]
      mu2[i,t] <- phi[i,t-1]*z[i,t-1] + gamma[t]*prod(q[i,1:(t-1)])
      z[i,t] ~ dbern(mu2[i,t])
      # Observation process
      mu3[i,t] <- z[i,t] * p[i,t]
      y[i,t] ~ dbern(mu3[i,t])
      } # t
   } # i

# Calculate derived population parameters
for (t in 1:n.occasions){
   qgamma[t] <- 1-gamma[t]
   }
cprob[1] <- gamma[1]
for (t in 2:n.occasions){
   cprob[t] <- gamma[t] * prod(qgamma[1:(t-1)])
   } # t
psi <- sum(cprob[])             # Inclusion probability
for (t in 1:n.occasions){
   b[t] <- cprob[t] / psi       # Entry probability
   } # t

for (i in 1:M){
   recruit[i,1] <- z[i,1]
   for (t in 2:n.occasions){
      recruit[i,t] <- (1-z[i,t-1]) * z[i,t]
      } # t
   } # i
for (t in 1:n.occasions){
   Nm[t] <- sum(z[1:mm,t])             # Actual population size of males
   Nf[t] <- sum(z[(mm+1):M,t])         # Actual population size of females
   Bm[t] <- sum(recruit[1:mm,t])       # Number of entries of males
   Bf[t] <- sum(recruit[(mm+1):M,t])   # Number of entries of females
   } # t
for (i in 1:M){
   Nind[i] <- sum(z[i,1:n.occasions])
```

```
   Nalive[i] <- 1-equals(Nind[i], 0)
   } # i
Nsuperm <- sum(Nalive[1:mm])           # Size of superpopulation of males
Nsuperf <- sum(Nalive[(mm+1):M])       # Size of superpopulation of females
}
",fill=TRUE)
sink()


# Augment the capture-histories by pseudo-individuals
nz <- 500
CHm.aug <- rbind(CH.m, matrix(0, ncol = dim(CH.m)[2], nrow = nz))
m <- rep(1, dim(CHm.aug)[1])
CHf.aug <- rbind(CH.f, matrix(0, ncol = dim(CH.f)[2], nrow = nz))
f <- rep(2, dim(CHf.aug)[1])
group <- c(m, f)
y <- rbind(CHm.aug, CHf.aug)

# Bundle data
bugs.data <- list(y = y, n.occasions = dim(y)[2], M = dim(y)[1], group =
group, mm = length(m))

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

# Parameters monitored
parameters <- c("psi", "mean.p", "beta", "b", "Nsuperm", "Nsuperf", "Nm",
"Nf", "Bm", "Bf")

# MCMC settings
niter <- 5000
nthin <- 3
nburn <- 3000
nchains <- 3

# Call WinBUGS from R (BRT 32 min)
js.occ <- bugs(bugs.data, inits, parameters, "js-rest.occ.bug", n.chains =
nchains, n.thin = nthin, n.iter = niter, n.burnin = nburn, debug = TRUE,
bugs.directory = bugs.dir, working.directory = getwd())

print(js.occ, 3)
Inference for Bugs model at "js-rest.occ.bug", fit using WinBUGS,
 3 chains, each with 5000 iterations (first 3000 discarded), n.thin = 3
 n.sims = 2001 iterations saved
             mean     sd    2.5%     25%     50%     75%   97.5%  Rhat n.eff
psi         0.440  0.028   0.388   0.420   0.439   0.459   0.496 1.006   510
mean.p      0.403  0.031   0.344   0.382   0.403   0.424   0.466 1.011   900
beta[1]     0.745  0.027   0.691   0.727   0.745   0.763   0.797 1.005   420
beta[2]     0.504  0.043   0.421   0.474   0.504   0.533   0.588 1.005   960
b[1]        0.304  0.034   0.243   0.281   0.303   0.326   0.379 1.001  2000
b[2]        0.105  0.031   0.046   0.083   0.104   0.125   0.167 1.008  2000
b[3]        0.077  0.027   0.025   0.059   0.075   0.094   0.133 1.021   570
b[4]        0.060  0.024   0.017   0.044   0.059   0.075   0.110 1.028   130
b[5]        0.145  0.028   0.092   0.124   0.144   0.164   0.201 1.003   650
b[6]        0.124  0.031   0.070   0.104   0.123   0.142   0.191 1.001  2000
b[7]        0.053  0.025   0.009   0.035   0.052   0.070   0.103 1.014   680
b[8]        0.132  0.026   0.082   0.115   0.131   0.149   0.185 1.006   380
Nsuperm   304.413 17.018 275.000 292.000 304.000 316.000 340.000 1.006   570
Nsuperf   286.309 18.953 252.000 273.000 286.000 298.000 327.000 1.008   310
Nm[1]      95.503 11.276  76.000  88.000  95.000 103.000 119.000 1.001  2000
Nm[2]     104.855 10.348  87.000  97.000 104.000 112.000 128.000 1.008  2000
Nm[3]     101.013  9.401  84.000  94.000 100.000 107.000 120.000 1.001  2000
Nm[4]      93.730  8.304  78.000  88.000  93.000  99.000 111.000 1.012   190
```

```
Nm[5]      111.000 11.107   91.000  103.000  111.000  118.000  134.000 1.009  1200
Nm[6]      123.369 10.993  104.000  116.000  123.000  131.000  145.000 1.006   490
Nm[7]      107.466 10.054   90.000  101.000  107.000  114.000  129.000 1.009   500
Nm[8]      117.738 12.814   96.000  109.000  117.000  126.000  145.000 1.003  1000
Nf[1]       85.056 11.681   65.000   77.000   84.000   92.000  110.000 1.006   650
Nf[2]       73.070 10.126   56.000   66.000   72.000   80.000   97.000 1.009  1400
Nf[3]       60.379  8.240   47.000   54.000   60.000   66.000   78.000 1.002  1100
Nf[4]       47.469  6.745   35.000   43.000   47.000   52.000   62.000 1.007   480
Nf[5]       64.641  8.653   49.000   59.000   64.000   70.000   83.000 1.002  2000
Nf[6]       65.020  9.198   49.000   59.000   64.000   71.000   86.000 1.004  1200
Nf[7]       48.491  7.706   35.000   43.000   48.000   53.000   65.000 1.005   520
Nf[8]       67.157  8.829   52.000   61.000   66.000   73.000   86.000 1.002  1100
Bm[1]       95.503 11.276   76.000   88.000   95.000  103.000  119.000 1.001  2000
Bm[2]       31.476  9.659   13.000   25.000   32.000   38.000   51.000 1.003  2000
Bm[3]       21.901  8.242    6.000   16.000   21.000   27.000   39.000 1.002   980
Bm[4]       18.720  7.358    5.000   14.000   19.000   23.000   34.000 1.014   150
Bm[5]       43.316  8.967   26.000   37.000   43.000   49.000   61.000 1.001  2000
Bm[6]       40.286  9.556   22.000   34.000   40.000   46.000   60.000 1.001  2000
Bm[7]       16.110  7.766    2.000   10.000   16.000   21.000   32.000 1.001  2000
Bm[8]       37.101  8.420   22.000   31.000   37.000   42.000   54.000 1.007   310
Bf[1]       85.056 11.681   65.000   77.000   84.000   92.000  110.000 1.006   650
Bf[2]       30.207  9.293   13.000   24.000   30.000   36.000   50.000 1.006  1800
Bf[3]       22.926  8.124    7.000   17.000   23.000   28.000   40.000 1.004   560
Bf[4]       16.246  6.414    5.000   12.000   16.000   20.000   30.000 1.017   130
Bf[5]       42.216  8.091   28.000   37.000   42.000   47.000   60.000 1.001  2000
Bf[6]       33.421  8.369   18.000   28.000   33.000   39.000   51.000 1.001  2000
Bf[7]       14.860  7.179    2.000   10.000   14.000   20.000   30.000 1.003  2000
Bf[8]       41.378  8.201   27.000   36.000   41.000   47.000   59.000 1.008   270
deviance  1836.392 98.454 1651.000 1767.000 1834.000 1906.000 2029.000 1.016  1100
```

Exercise 2

Task: Simulate capture-recapture data of a species collected over 7 years. Mean survival was 0.5, mean capture 0.6, and entry probability was 0.1 for all but the first occasion. The size of the superpopulation is assumed to be 500. Analyze the data with the model that explicitly uses constant entry probability for all occasions, but the first.

Solution: We simulate individual capture histories with function `simul.js`. Our goal is to model the entry probability directly, so we have to use the superpopulation approach to the JS model. To constrain the parameter *b* at and after the second occasion to the same value, we first give a prior for a variable lb_1 and lb_2, and specify that all lb_3 until lb_T are the same as lb_2. The sum of the *lb* certainly is different from one. We then define b_t as $lb_t/\Sigma lb$. The *b*'s then have the desired properties: b_2 until b_T have the same value and the sum of all *b* parameters is equal to 1.

Data simulation

```
# Define the parameters
n.occasions <- 7
N <- 500
phi <- rep(0.5, n.occasions-1)
b <- c(0.4, rep(0.1, n.occasions-1))
p <- rep(0.5, n.occasions)

PHI <- matrix(rep(phi, (n.occasions-1)*N), ncol = n.occasions-1, nrow = N,
byrow = T)
P <- matrix(rep(p, n.occasions*N), ncol = n.occasions, nrow = N, byrow = T)
```

```
# Apply simulation function
sim <- simul.js(PHI, P, b, N)
CH <- sim$CH
```

Data analysis

```
# Specify model in BUGS language
sink("js-super.bug")
cat("
model {
# Priors and constraints
for (i in 1:M){
   for (t in 1:(n.occasions-1)){
      phi[i,t] <- mean.phi
      } # t
   for (t in 1:n.occasions){
      p[i,t] <- mean.p
      } #t
   } #i

mean.phi ~ dunif(0, 1)          # Prior for mean survival
mean.p ~ dunif(0, 1)            # Prior for mean capture
psi ~ dunif(0, 1)               # Prior for inclusion probability

# Choose priors for the first two free b
lb[1] ~ dunif(0, 1)
lb[2] ~ dunif(0, 1)
for (t in 3:n.occasions){
   lb[t] <- lb[2]
   }
# Weigh the lb such that they sum to 1
for (t in 1:n.occasions){
   b[t] <- lb[t] / sum(lb[])
   }

# Convert entry probs to conditional entry probs
nu[1] <- b[1]
for (t in 2:n.occasions){
   nu[t] <- b[t] / (1-sum(b[1:(t-1)]))
   } # t

# Define the likelihood
for (i in 1:M){
   # First occasion
   # State process
   w[i] ~ dbern(psi)                     # Draw latent inclusion variable
   z[i,1] ~ dbern(nu[1])
   # Observation process
   mu1[i] <- z[i,1] * p[i,1] * w[i]
   y[i,1] ~ dbern(mu1[i])

   # Subsequent occasions
   for (t in 2:n.occasions){
      # State process
      q[i,t-1] <- 1-z[i,t-1]
      mu2[i,t] <- phi[i,t-1] * z[i,t-1] + nu[t] * prod(q[i,1:(t-1)])
      z[i,t] ~ dbern(mu2[i,t])
      # Observation process
      mu3[i,t] <- z[i,t] * p[i,t] * w[i]
      y[i,t] ~ dbern(mu3[i,t])
      } #t
   } #i
```

```
# Calculate derived population parameters
for (i in 1:M){
   for (t in 1:n.occasions){
      u[i,t] <- z[i,t]*w[i]     # Deflated latent state (u)
      }
   }
for (i in 1:M){
   recruit[i,1] <- u[i,1]
   for (t in 2:n.occasions){
      recruit[i,t] <- (1-u[i,t-1]) * u[i,t]
      } #t
   } #i
for (t in 1:n.occasions){
   N[t] <- sum(u[1:M,t])        # Actual population size
   B[t] <- sum(recruit[1:M,t])  # Number of entries
   } #t
for (i in 1:M){
   Nind[i] <- sum(u[i,1:n.occasions])
   Nalive[i] <- 1-equals(Nind[i], 0)
   } #i
Nsuper <- sum(Nalive[])         # Size of superpopulation
}
",fill=TRUE)
sink()


# Augment the capture-histories by nz pseudo-individuals
nz <- 500
CH.aug <- rbind(CH, matrix(0, ncol = dim(CH)[2], nrow = nz))

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

# Initial values
inits <- function(){list(mean.phi = runif(1, 0, 1), mean.p = runif(1, 0,
1), psi = runif(1, 0, 1), lb = c(runif(2, 0, 0.1), rep(NA, n.occasions-2)),
z = CH.aug)}

# Parameters monitored
parameters <- c("psi", "mean.p", "mean.phi", "b", "Nsuper", "N", "B")

# MCMC settings
niter <- 5000
nthin <- 3
nburn <- 3000
nchains <- 3

# Call WinBUGS from R (BRT 24 min)
js.super <- bugs(bugs.data, inits, parameters, "js-super.bug", n.chains =
nchains, n.thin = nthin, n.iter = niter, n.burnin = nburn, debug = TRUE,
bugs.directory = bugs.dir, working.directory = getwd())

print(js.super, digits = 3)
Inference for Bugs model at "js-super.bug", fit using WinBUGS,
 3 chains, each with 5000 iterations (first 3000 discarded), n.thin = 3
 n.sims = 2001 iterations saved
             mean     sd    2.5%     25%     50%     75%    97.5%  Rhat n.eff
psi         0.652  0.055   0.552   0.614   0.648   0.689   0.769 1.006   630
mean.p      0.454  0.052   0.360   0.418   0.454   0.489   0.555 1.007   380
mean.phi    0.485  0.037   0.416   0.460   0.484   0.510   0.561 1.013   180
```

```
b[1]         0.361   0.037    0.294    0.336    0.358    0.385    0.442 1.020   130
b[2]         0.107   0.006    0.093    0.103    0.107    0.111    0.118 1.029   120
b[3]         0.107   0.006    0.093    0.103    0.107    0.111    0.118 1.029   120
b[4]         0.107   0.006    0.093    0.103    0.107    0.111    0.118 1.029   120
b[5]         0.107   0.006    0.093    0.103    0.107    0.111    0.118 1.029   120
b[6]         0.107   0.006    0.093    0.103    0.107    0.111    0.118 1.029   120
b[7]         0.107   0.006    0.093    0.103    0.107    0.111    0.118 1.029   120
Nsuper     531.667  43.296  455.000  501.000  528.000  559.000  623.000 1.005   690
N[1]       192.504  25.973  149.000  174.000  189.000  207.000  250.000 1.013   240
N[2]       151.999  18.422  122.000  139.000  150.000  164.000  192.000 1.006   500
N[3]       124.310  16.278   97.000  112.000  123.000  135.000  159.000 1.004   640
N[4]       113.771  14.545   89.000  103.000  113.000  123.000  145.000 1.004   480
N[5]       114.418  14.139   91.000  104.000  113.000  123.000  145.000 1.004   690
N[6]       121.464  13.672   98.000  112.000  120.000  131.000  150.000 1.005   610
N[7]       112.839  14.569   89.000  102.000  112.000  122.000  145.000 1.005   880
B[1]       192.504  25.973  149.000  174.000  189.000  207.000  250.000 1.013   240
B[2]        58.179   7.750   44.000   53.000   58.000   63.000   74.000 1.001  2000
B[3]        52.001   7.259   39.000   47.000   52.000   57.000   67.000 1.001  2000
B[4]        55.300   7.187   42.000   50.000   55.000   60.000   70.000 1.002  1200
B[5]        57.360   7.209   44.000   52.000   57.000   62.000   72.000 1.001  2000
B[6]        62.591   7.314   50.000   57.000   62.000   67.000   78.000 1.001  2000
B[7]        53.732   7.397   40.000   49.000   53.000   58.000   70.000 1.004  2000
deviance 1273.020 122.398 1051.000 1186.000 1271.000 1355.000 1511.000 1.005   590
```

Exercise 3

Task: Simulate data for a species for which capture-recapture data are sampled and recoveries of dead individuals are available. The study runs for 10 years, mean survival was 0.5, mean capture 0.6, mean recovery was 0.2 and entry probability was 0.1 for all but the first occasion. The size of the superpopulation is assumed to be 500. Analyze the simulated data with an appropriate model.

Solution: This exercise is relatively complicated, since it requires the adaptation of the simulation function to create data and the specification of the JS model with a multistate model. The necessary steps are explained below.

<u>*Data simulation*</u>

To simulate the data, we adapt the function `simul.js` in such a way that dead recoveries can also be obtained. Basically we have to record the occasion when individuals die and evaluate whether they were recovered. Finally we remove individuals that were never marked, but found dead. We name the modified simulation function `simul.jsrecov`. The necessary changes made to the original function are highlighted in bold red font.

```
# Function to simulate capture-recapture data for JS analysis that also
includes dead recoveries
# Dead recoveries are coded with a 2 in the resulting capture histories
simul.jsrecov <- function(PHI, P, R, b, N){
   B <- rmultinom(1, N, b) # Generate no. of entering ind. per occasion
   n.occasions <- dim(PHI)[2] + 1
   CH.sur <- CH.p <- CH.r <- matrix(0, ncol = n.occasions, nrow = N)
   # Define a vector with the occasion of entering the population
   ent.occ <- numeric()
   for (t in 1:n.occasions){
      ent.occ <- c(ent.occ, rep(t, B[t]))
      }
   # Modeling survival
   for (i in 1:N){
```

```
      CH.sur[i, ent.occ[i]] <- 1   # Write 1 when ind. enters the pop.
      if (ent.occ[i] == n.occasions) next
      for (t in (ent.occ[i]+1):n.occasions){
         # Bernoulli trial: has individual survived occasion?
         sur <- rbinom(1, 1, PHI[i,t-1])
         if (sur==1) CH.sur[i,t] <- 1
         if (sur==0) CH.sur[i,t] <- 2
         if (sur==0) break
         } #t
      } #i
   # Modeling capture
   for (i in 1:N){
      CH.p[i,] <- rbinom(n.occasions, 1, P[i,])
      } #i
   # Modeling recovery
   for (i in 1:N){
      CH.r[i,] <- rbinom(n.occasions, 1, R[i,])
      } #i
   # Full capture-recapture matrix
   CH <- CH.sur * CH.p
   CH.2 <- CH.sur * CH.r
   CH[CH==2] <- 0
   CH[CH.2==2] <- 2
   # Remove individuals never captured
   cap.sum <- rowSums(CH)
   never <- which(cap.sum == 0)
   CH <- CH[-never,]
   # Remove individuals that were found dead, but never marked
   cap.sum <- rowSums(CH)
   two <- which(cap.sum==2)
   CH.help <- CH[two,]
   CH.help[CH.help==0] <- 1
   w <- apply(CH.help, 1, prod)
   rem <- two[which(w==2)]
   CH <- CH[-rem,]
   # Actual population size
   CH.pop <- CH.sur
   CH.pop[CH.pop==2] <- 0
   Nt <- colSums(CH.pop)
   return(list(CH=CH, B=B, N=Nt))
   }
```

Next we define the data generating parameters and simulate a data set.

```
# Define the parameters
n.occasions <- 10
N <- 500
phi <- 0.5
b <- rep(0.1, 10)
p <- 0.6
r <- 0.2

PHI <- matrix(rep(phi, (n.occasions-1)*N), ncol = n.occasions-1, nrow = N,
byrow = T)
P <- matrix(rep(p, n.occasions*N), ncol = n.occasions, nrow = N, byrow = T)
R <- matrix(rep(r, n.occasions*N), ncol = n.occasions, nrow = N, byrow = T)

# Apply simulation function
sim <- simul.jsrecov(PHI, P, R, b, N)
CH <- sim$CH
```

Data analysis
To be able to include dead recoveries in a JS model, we use the multistate formulation of the JS model. The state transition matrix of the classical formulation of the JS model (section 10.3.2 in the BPA book) has to be enlarged by an additional state "recently dead" (see section 9.5 in the BPA book). This is necessary to ensure that individuals that die at a certain time can only be recovered within the next time interval. Thus, the state transition matrix is this:

$$\begin{array}{c c} & \begin{array}{cccc} \text{not yet entered} & \text{alive} & \text{recently dead} & \text{dead} \end{array} \\ \begin{array}{c} \text{not yet entered} \\ \text{alive} \\ \text{recently dead} \\ \text{dead} \end{array} & \begin{bmatrix} 1-\gamma & \gamma & 0 & 0 \\ 0 & \phi & 1-\phi & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array}.$$

The observation matrix also needs some changes. The recovery process has to be included, and the matrix is:

$$\begin{array}{c c} & \begin{array}{ccc} \text{seen alive} & \text{recovered dead} & \text{not seen or recovered} \end{array} \\ \begin{array}{c} \text{not yet recruited} \\ \text{alive} \\ \text{recently dead} \\ \text{dead} \end{array} & \begin{bmatrix} 0 & 0 & 1 \\ p & 0 & 1-p \\ 0 & r & 1-r \\ 0 & 0 & 1 \end{bmatrix} \end{array}.$$

Strangely, for reasons unknown to us, WinBUGS produces the wrong parameter estimates when this parameterisation of the model is fitted. In contrast, areparameterized version of the model, where the recovery process is included in the state transition process, works well (see also discussion in section 9.5 of the BPA book). Thus, instead of the state "recently dead" we define the state "recently dead and recovered" and another state "dead". The latter includes individuals that are recently dead, but have not been recovered along with individuals that have been dead for a longer time. The state transition matrix is then:

$$\begin{array}{c c} & \begin{array}{cccc} \text{not yet entered} & \text{alive} & \text{recently dead, recovered} & \text{dead} \end{array} \\ \begin{array}{c} \text{not yet entered} \\ \text{alive} \\ \text{recently dead, recovered} \\ \text{dead} \end{array} & \begin{bmatrix} 1-\gamma & \gamma & 0 & 0 \\ 0 & \phi & (1-\phi)r & (1-\phi)(1-r) \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

The observation matrix also needs a slight adaptation – the recovery parameter r is not included anymore.

$$
\begin{array}{r c}
 & \begin{array}{ccc} \text{seen alive} & \text{recovered dead} & \text{not seen or recovered} \end{array} \\
\begin{array}{r} \text{not yet recruited} \\ \text{alive} \\ \text{recently dead, recovered} \\ \text{dead} \end{array} &
\begin{bmatrix} 0 & 0 & 1 \\ p & 0 & 1-p \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}
\end{array}
$$

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

#--------------------------------------
# Parameters:
# phi: survival probability
# gamma: removal entry probability
# p: capture probability
#--------------------------------------
# States (S):
# 1 not yet entered
# 2 alive
# 3 recently dead and recovered
# 4 dead or recently dead, but not recovered
# Observations (O):
# 1 seen
# 2 recovered
# 3 neither seen nor recovered
#--------------------------------------

# Priors and constraints
for (t in 1:(n.occasions-1)){
   phi[t] <- mean.phi
   gamma[t] ~ dunif(0, 1) # Prior for entry probabilities
   p[t] <- mean.p
   r[t] <- mean.r
   }

mean.phi ~ dunif(0, 1)    # Prior for mean survival
mean.p ~ dunif(0, 1)      # Prior for mean capture
mean.r ~ dunif(0, 1)      # Prior for mean recovery

# Define state-transition and observation matrices
for (i in 1:M){
   # Define probabilities of state S(t+1) given S(t)
   for (t in 1:(n.occasions-1)){
      ps[1,i,t,1] <- 1-gamma[t]
      ps[1,i,t,2] <- gamma[t]
      ps[1,i,t,3] <- 0
      ps[1,i,t,4] <- 0
      ps[2,i,t,1] <- 0
      ps[2,i,t,2] <- phi[t]
      ps[2,i,t,3] <- (1-phi[t])*r[t]
      ps[2,i,t,4] <- (1-phi[t])*(1-r[t])
      ps[3,i,t,1] <- 0
      ps[3,i,t,2] <- 0
      ps[3,i,t,3] <- 0
      ps[3,i,t,4] <- 1
      ps[4,i,t,1] <- 0
```

```
      ps[4,i,t,2] <- 0
      ps[4,i,t,3] <- 0
      ps[4,i,t,4] <- 1

      # Define probabilities of O(t) given S(t)
      po[1,i,t,1] <- 0
      po[1,i,t,2] <- 0
      po[1,i,t,3] <- 1
      po[2,i,t,1] <- p[t]
      po[2,i,t,2] <- 0
      po[2,i,t,3] <- 1-p[t]
      po[3,i,t,1] <- 0
      po[3,i,t,2] <- 1
      po[3,i,t,3] <- 0
      po[4,i,t,1] <- 0
      po[4,i,t,2] <- 0
      po[4,i,t,3] <- 1

      } #t
   } #i

# Likelihood
for (i in 1:M){
   # Define latent state at first occasion
   z[i,1] <- 1   # Make sure that all M individuals are in state 1 at t=1
   for (t in 2:n.occasions){
      # State process: draw S(t) given S(t-1)
      z[i,t] ~ dcat(ps[z[i,t-1], i, t-1,])
      # Observation process: draw O(t) given S(t)
      y[i,t] ~ dcat(po[z[i,t], i, t-1,])
      } #t
   } #i

# Calculate derived population parameters
for (t in 1:(n.occasions-1)){
   qgamma[t] <- 1-gamma[t]
   }
cprob[1] <- gamma[1]
for (t in 2:(n.occasions-1)){
   cprob[t] <- gamma[t] * prod(qgamma[1:(t-1)])
   } #t
psi <- sum(cprob[])            # Inclusion probability
for (t in 1:(n.occasions-1)){
   b[t] <- cprob[t] / psi      # Entry probability
   } #t

for (i in 1:M){
   for (t in 2:n.occasions){
      al[i,t-1] <- equals(z[i,t], 2)
      } #t
   for (t in 1:(n.occasions-1)){
      d[i,t] <- equals(z[i,t]-al[i,t],0)
      } #t
   alive[i] <- sum(al[i,])
   } #i

for (t in 1:(n.occasions-1)){
   N[t] <- sum(al[,t])        # Actual population size
   B[t] <- sum(d[,t])         # Number of entries
   } #t
for (i in 1:M){
   w[i] <- 1-equals(alive[i],0)
```

```
   } #i
Nsuper <- sum(w[])            # Superpopulation size
}
",fill = TRUE)
sink()


# Add dummy occasion
CH.du <- cbind(rep(0, dim(CH)[1]), CH)

# Augment data
nz <- 500
CH.ms <- rbind(CH.du, matrix(0, ncol = dim(CH.du)[2], nrow = nz))

# Recode CH matrix: a 0 is not allowed in WinBUGS!
CH.ms[CH.ms==0] <- 3                # Not seen = 3, seen = 1, recovered = 3
```

Then we run the analysis.

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

# Initial values
inits <- function(){list(mean.phi = runif(1, 0, 1), mean.p = runif(1, 0,
1), mean.r = runif(1, 0, 0.5), z = cbind(rep(NA, dim(CH.ms)[1]), CH.ms[,-
1]))}

# Parameters monitored
parameters <- c("mean.p", "mean.r", "mean.phi", "b", "Nsuper", "N", "B")

# MCMC settings
ni <- 50000
nt <- 3
nb <- 25000
nc <- 3

# Call WinBUGS from R (BRT 203 min)
js.ms <- bugs(bugs.data, inits, parameters, "js-ms.bug", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory =
bugs.dir, working.directory = getwd())

print(js.ms, digits = 3)
Inference for Bugs model at "js-ms.bug", fit using WinBUGS,
 3 chains, each with 50000 iterations (first 25000 discarded), n.thin = 3
 n.sims = 25002 iterations saved
             mean     sd    2.5%      25%      50%      75%    97.5%  Rhat n.eff
mean.p      0.636  0.035   0.568    0.613    0.637    0.660    0.704 1.004   960
mean.r      0.159  0.019   0.124    0.146    0.158    0.172    0.199 1.001  7000
mean.phi    0.471  0.026   0.422    0.454    0.471    0.488    0.522 1.002  2700
b[1]        0.110  0.018   0.078    0.097    0.109    0.121    0.147 1.002  3300
b[2]        0.090  0.018   0.057    0.078    0.090    0.102    0.128 1.001  6500
b[3]        0.119  0.020   0.081    0.105    0.118    0.132    0.160 1.001 25000
b[4]        0.092  0.019   0.057    0.078    0.091    0.104    0.130 1.001  5200
b[5]        0.096  0.019   0.061    0.082    0.095    0.108    0.134 1.001 12000
b[6]        0.111  0.020   0.074    0.097    0.110    0.123    0.152 1.001 21000
b[7]        0.101  0.019   0.066    0.088    0.100    0.114    0.141 1.001  4300
b[8]        0.068  0.017   0.038    0.057    0.068    0.079    0.103 1.001 14000
b[9]        0.129  0.021   0.091    0.115    0.129    0.143    0.172 1.001 14000
b[10]       0.085  0.018   0.051    0.072    0.084    0.097    0.123 1.001  9800
Nsuper    474.958 17.073 444.000  463.000  474.000  485.000  511.000 1.008   830
N[1]       51.861  5.735  42.000   48.000   51.000   55.000   64.000 1.003  1300
N[2]       66.365  6.139  56.000   62.000   66.000   70.000   80.000 1.002  1600
```

```
N[3]        89.518  7.147  77.000   84.000   89.000   94.000  105.000 1.002  2100
N[4]        87.259  6.733  75.000   83.000   87.000   91.000  102.000 1.002  2500
N[5]        90.131  6.814  78.000   85.000   90.000   94.000  105.000 1.003  1200
N[6]        97.555  7.295  85.000   92.000   97.000  102.000  113.000 1.002  1800
N[7]        98.070  6.880  86.000   93.000   98.000  102.000  113.000 1.002  2100
N[8]        74.223  6.214  63.000   70.000   74.000   78.000   88.000 1.003   890
N[9]        91.217  7.920  77.000   86.000   91.000   96.000  108.000 1.003  1100
N[10]       80.584  7.917  67.000   75.000   80.000   85.000   98.000 1.002  2400
B[1]        51.861  5.735  42.000   48.000   51.000   55.000   64.000 1.003  1300
B[2]        42.600  6.281  31.000   38.000   42.000   47.000   56.000 1.001  5500
B[3]        56.347  7.003  43.000   52.000   56.000   61.000   71.000 1.001 10000
B[4]        43.356  6.596  31.000   39.000   43.000   48.000   57.000 1.001 25000
B[5]        45.205  6.508  33.000   41.000   45.000   49.000   59.000 1.002  2300
B[6]        52.589  6.862  40.000   48.000   52.000   57.000   67.000 1.001  5600
B[7]        48.227  6.767  36.000   44.000   48.000   53.000   62.000 1.001 11000
B[8]        32.250  6.002  21.000   28.000   32.000   36.000   44.000 1.002  3400
B[9]        62.146  7.166  49.000   57.000   62.000   67.000   77.000 1.001  3800
B[10]       40.378  6.954  27.000   36.000   40.000   45.000   55.000 1.001 19000
deviance  1082.786 80.513 929.000 1028.000 1080.000 1135.000 1248.000 1.006   610
```

Long Markov chains are required for this model to reach convergence. The parameter estimates are fairly close to the values of the data-generating parameters.

An interesting issue would also be to understand whether the dead recoveries provide much additional information, i.e. to analyze the data set when the dead recoveries are excluded. This is done below:

```
# Remove recoveries
CH.msalt <- CH.ms
CH.msalt[CH.msalt==2] <- 3
CH.msalt[CH.msalt==3] <- 2           # 1: seen alive, 2: not seen
```

The multistate model also needs some changes – it is the same model as the one presented in section 10.3.2 of the BPA book.

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

#--------------------------------------
# Parameters:
# phi: survival probability
# gamma: removal entry probability
# p: capture probability
#--------------------------------------
# States (S):
# 1 not yet entered
# 2 alive
# 3 dead
# Observations (O):
# 1 seen
# 2 not seen
#--------------------------------------

# Priors and constraints
for (t in 1:(n.occasions-1)){
   phi[t] <- mean.phi
   gamma[t] ~ dunif(0, 1) # Prior for entry probabilities
   p[t] <- mean.p
```

```
   }

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

# Define state-transition and observation matrices
for (i in 1:M){
   # Define probabilities of state S(t+1) given S(t)
   for (t in 1:(n.occasions-1)){
      ps[1,i,t,1] <- 1-gamma[t]
      ps[1,i,t,2] <- gamma[t]
      ps[1,i,t,3] <- 0
      ps[2,i,t,1] <- 0
      ps[2,i,t,2] <- phi[t]
      ps[2,i,t,3] <- 1-phi[t]
      ps[3,i,t,1] <- 0
      ps[3,i,t,2] <- 0
      ps[3,i,t,3] <- 1

      # Define probabilities of O(t) given S(t)
      po[1,i,t,1] <- 0
      po[1,i,t,2] <- 1
      po[2,i,t,1] <- p[t]
      po[2,i,t,2] <- 1-p[t]
      po[3,i,t,1] <- 0
      po[3,i,t,2] <- 1
      } #t
   } #i

# Likelihood
for (i in 1:M){
   # Define latent state at first occasion
   z[i,1] <- 1   # Make sure that all M individuals are in state 1 at t=1
   for (t in 2:n.occasions){
      # State process: draw S(t) given S(t-1)
      z[i,t] ~ dcat(ps[z[i,t-1], i, t-1,])
      # Observation process: draw O(t) given S(t)
      y[i,t] ~ dcat(po[z[i,t], i, t-1,])
      } #t
   } #i

# Calculate derived population parameters
for (t in 1:(n.occasions-1)){
   qgamma[t] <- 1-gamma[t]
   }
cprob[1] <- gamma[1]
for (t in 2:(n.occasions-1)){
   cprob[t] <- gamma[t] * prod(qgamma[1:(t-1)])
   } #t
psi <- sum(cprob[])            # Inclusion probability
for (t in 1:(n.occasions-1)){
   b[t] <- cprob[t] / psi      # Entry probability
   } #t

for (i in 1:M){
   for (t in 2:n.occasions){
      al[i,t-1] <- equals(z[i,t], 2)
      } #t
   for (t in 1:(n.occasions-1)){
      d[i,t] <- equals(z[i,t]-al[i,t],0)
      } #t
   alive[i] <- sum(al[i,])
```

```
   } #i

for (t in 1:(n.occasions-1)){
   N[t] <- sum(al[,t])        # Actual population size
   B[t] <- sum(d[,t])         # Number of entries
   } #t
for (i in 1:M){
   w[i] <- 1-equals(alive[i],0)
   } #i
Nsuper <- sum(w[])            # Superpopulation size
}
",fill = TRUE)
sink()


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

# Initial values
inits <- function(){list(mean.phi = runif(1, 0, 1), mean.p = runif(1, 0,
1), z = cbind(rep(NA, dim(CH.msalt)[1]), CH.msalt[,-1]))}

# Parameters monitored
parameters <- c("mean.p", "mean.phi", "b", "Nsuper", "N", "B")

# MCMC settings
ni <- 50000
nt <- 3
nb <- 25000
nc <- 3

# Call WinBUGS from R (BRT 240 min)
js.msalt <- bugs(bugs.data, inits, parameters, "js-ms.bug", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory =
bugs.dir, working.directory = getwd())

print(js.msalt, 3)
Inference for Bugs model at "js-ms.bug", fit using WinBUGS,
 3 chains, each with 20000 iterations (first 5000 discarded), n.thin = 3
 n.sims = 15000 iterations saved
            mean     sd    2.5%     25%     50%     75%   97.5%  Rhat n.eff
mean.p     0.541  0.052   0.441   0.507   0.540   0.574   0.650 1.011   530
mean.phi   0.472  0.030   0.415   0.452   0.471   0.492   0.531 1.001  8300
b[1]       0.116  0.019   0.082   0.103   0.115   0.129   0.157 1.002  1400
b[2]       0.091  0.020   0.054   0.077   0.090   0.103   0.132 1.001 25000
b[3]       0.120  0.022   0.080   0.106   0.120   0.134   0.165 1.001  8300
b[4]       0.090  0.020   0.053   0.076   0.090   0.103   0.132 1.001  7500
b[5]       0.095  0.020   0.058   0.081   0.095   0.108   0.136 1.001 25000
b[6]       0.111  0.021   0.072   0.097   0.110   0.125   0.154 1.001 13000
b[7]       0.101  0.021   0.063   0.087   0.100   0.114   0.144 1.001  7000
b[8]       0.066  0.018   0.033   0.054   0.065   0.078   0.103 1.002  2300
b[9]       0.132  0.022   0.092   0.117   0.132   0.147   0.177 1.001 18000
b[10]      0.077  0.019   0.041   0.064   0.076   0.089   0.115 1.001 25000
Nsuper   559.855 39.119 486.000 535.000 557.000 582.000 646.000 1.015   360
N[1]      64.929 10.087  48.000  58.000  64.000  71.000  88.000 1.010   360
N[2]      80.214 10.956  62.000  73.000  79.000  87.000 105.000 1.009   510
N[3]     107.488 12.860  86.000  99.000 106.000 115.000 136.000 1.007   470
N[4]     103.110 12.045  82.000  95.000 102.000 110.000 130.000 1.007   780
N[5]     104.300 11.950  84.000  96.000 103.000 111.000 131.000 1.004   990
N[6]     113.837 13.299  91.000 105.000 113.000 122.000 144.000 1.004   990
N[7]     115.004 12.598  93.000 106.000 114.000 123.000 143.000 1.004  1100
N[8]      85.104 10.913  66.000  78.000  84.000  92.000 109.000 1.004  2900
N[9]     111.071 14.437  86.000 101.000 110.000 120.000 143.000 1.005   990
```

```
N[10]       93.607  12.348   73.000   85.000   93.000  101.000  121.000 1.004  1100
B[1]        64.929  10.087   48.000   58.000   64.000   71.000   88.000 1.010   360
B[2]        50.446   9.570   33.000   44.000   50.000   56.000   71.000 1.002  1500
B[3]        67.293  10.781   49.000   60.000   67.000   74.000   90.000 1.003   980
B[4]        50.308   9.812   32.000   44.000   50.000   56.000   71.000 1.002  2600
B[5]        53.149   9.628   36.000   47.000   53.000   59.000   74.000 1.001  5700
B[6]        62.469  10.567   44.000   55.000   62.000   69.000   85.000 1.002  3500
B[7]        56.486   9.968   39.000   50.000   56.000   63.000   78.000 1.001  9000
B[8]        36.758   8.560   21.000   31.000   36.000   42.000   55.000 1.001 18000
B[9]        74.949  11.390   56.000   67.000   74.000   82.000  100.000 1.002  1600
B[10]       43.067   9.407   26.000   37.000   43.000   49.000   63.000 1.001 14000
deviance  1342.787 139.504 1059.000 1255.000 1342.000 1431.000 1623.000 1.012   810
```

The estimates of the survival probabilities are in both cases nearly identical, but the precision is a bit better when dead recoveries were included. This is to be expected, since dead recoveries provide additional information about survival. The estimated population sizes are a bit different in both cases, but their confidence intervals are still widely overlapping.

Chapter 11

Exercise 1

Task: Predict the hoopoe population size in 3 years. How large is the extinction probability (assume an extinction threshold of 5 pairs)?

Solution: In order to predict the population size in the three years after the study ended, we simply extend the loop of the state model (system process) by 3 years. The demographic parameters in these three years are generated from the estimated mean and temporal variance of the corresponding parameter, and they are the used for the calculation of the predicted population sizes. Uncertainty is directly accounted for, which is a very handy property! The extinction probability can easily be computed after running the model.

Load data

```
nyears <- 9   # Number of years

# Capture recapture data: m-array of juveniles and adults (these are males
and females together)
marray.j <- matrix (c(15, 3, 0, 0, 0, 0, 0, 0, 198, 0, 34, 9, 1, 0, 0, 0,
0, 287, 0, 0, 56, 8, 1, 0, 0, 0, 455, 0, 0, 0, 48, 3, 1, 0, 0, 518, 0, 0,
0, 0, 45, 13, 2, 0, 463, 0, 0, 0, 0, 0, 27, 7, 0, 493, 0, 0, 0, 0, 0, 0,
37, 3, 434, 0, 0, 0, 0, 0, 0, 0, 39, 405), nrow = 8, ncol = 9, byrow =
TRUE)
marray.a <- matrix(c(14, 2, 0, 0, 0, 0, 0, 0, 43, 0, 22, 4, 0, 0, 0, 0, 0,
44, 0, 0, 34, 2, 0, 0, 0, 0, 79, 0, 0, 0, 51, 3, 0, 0, 0, 94, 0, 0, 0, 0,
45, 3, 0, 0, 118, 0, 0, 0, 0, 0, 44, 3, 0, 113, 0, 0, 0, 0, 0, 0, 48, 2,
99, 0, 0, 0, 0, 0, 0, 0, 51, 90), nrow = 8, ncol = 9, byrow = TRUE)

# Population population count data
popcount <- c(32, 42, 64, 85, 82, 78, 73, 69, 79)

# Reproductive success
J <- c(189, 274, 398, 538, 520, 476, 463, 438, 507) # number offspring
R <- c(28, 36, 57, 77, 81, 83, 77, 72, 85)     # number of surveyed broods

# Number of years with predictions
t.pred <- 3
```

Data analysis

```
# Specify model in BUGS language
sink("ipm.hoopoe.bug")
cat("
model {
##########################################################################
#  Integrated population model
#  - Age structured model with 2 age classes:
#          1-year old and at least 2 years old
#  - Age at first breeding = 1 year
#  - Pre-breeding census, female-based
#  - All vital rates are assumed to be time-dependent (random)
#  - Explicit estimate of immigration
##########################################################################

############################################################
```

```
# 1. Define the priors for the parameters
############################################################

# Initial population sizes
N1[1] ~ dnorm(10, 0.001)I(0,)          # 1-year old individuals
NadSurv[1] ~ dnorm(10, 0.001)I(0,)     # Adults >= 2 years
Nadimm[1] ~ dnorm(10, 0.001)I(0,)      # Immigrants

# Mean demographic parameters
mphij ~ dunif(0, 1)
mphia ~ dunif(0, 1)
mfec ~ dunif(0, 15)
mim ~ dunif(0, 3)
mp ~ dunif(0, 1)

# Precision of standard deviations of temporal variability
sig.phij ~ dunif(0, 10)
tau.phij <- pow(sig.phij, -2)
sig.phia ~ dunif(0, 10)
tau.phia <- pow(sig.phia, -2)
sig.fec ~ dunif(0, 10)
tau.fec <- pow(sig.fec, -2)
sig.im ~ dunif(0, 10)
tau.im <- pow(sig.im, -2)

# Distribution of error terms (Bounded to help with convergence)
for (t in 1:(nyears-1+t.pred)){
   epsilon.phij[t] ~ dnorm(0, tau.phij)I(-15,15)
   epsilon.phia[t] ~ dnorm(0, tau.phia)I(-15,15)
   epsilon.fec[t] ~ dnorm(0, tau.fec)I(-15,15)
   epsilon.im[t] ~ dnorm(0, tau.im)I(-15,15)
   }

############################################################
# 2. Constrain parameters
############################################################
for (t in 1:(nyears-1+t.pred)){
   logit(phij[t]) <- l.mphij + epsilon.phij[t]  # Juv. apparent survival
   logit(phia[t]) <- l.mphia + epsilon.phia[t]  # Adult apparent survival
   log(f[t]) <- l.mfec + epsilon.fec[t]         # Fecundity
   log(omega[t]) <- l.mim + epsilon.im[t]       # Immigration
   p[t] <- mp                                   # Recapture probability
   }

############################################################
# 3. Derived parameters
############################################################

l.mphij <- log(mphij / (1-mphij))     # Logit mean juv. survival
l.mphia <- log(mphia / (1-mphia))     # Logit mean adult survival
l.mfec <- log(mfec)                   # Log mean fecundity
l.mim <- log(mim)                     # Log mean immigration rate

# Population growth rate
for (t in 1:(nyears-1+t.pred)){
   lambda[t] <- Ntot[t+1] / Ntot[t]
   logla[t] <- log(lambda[t])
   }
mlam <- exp((1/(nyears-1))*sum(logla[1:(nyears-1)]))   # Geometric mean

######################################################################
# 4. The likelihoods of the single data sets
```

```
######################################################################

###################################################################
# 4.1. Likelihood for population count data (state-space model)
###################################################################

   #############################
   # 4.1.1 System process
   #############################
   for (t in 2:nyears+t.pred){
      mean1[t] <- 0.5 * f[t-1] * phij[t-1] * Ntot[t-1]
      N1[t] ~ dpois(mean1[t])
      NadSurv[t] ~ dbin(phia[t-1], Ntot[t-1])
      mpo[t] <- Ntot[t-1] * omega[t-1]
      Nadimm[t] ~ dpois(mpo[t])
      }
   for (t in 1:nyears+t.pred){
      Ntot[t] <- NadSurv[t] + Nadimm[t] + N1[t]
      }

   #############################
   # 4.1.2 Observation process
   #############################
   for (t in 1:nyears){
      y[t] ~ dpois(Ntot[t])
      }

######################################################################
# 4.2 Likelihood for capture-recapture data: CJS model (2 age classes)
######################################################################

# Multinomial likelihood
for (t in 1:(nyears-1)){
   marray.j[t,1:nyears] ~ dmulti(pr.j[t,], r.j[t])
   marray.a[t,1:nyears] ~ dmulti(pr.a[t,], r.a[t])
   }

# Calculate number of released individuals
for (t in 1:(nyears-1)){
   r.j[t] <- sum(marray.j[t,])
   r.a[t] <- sum(marray.a[t,])
   }

# m-array cell probabilities for juveniles
for (t in 1:(nyears-1)){
   q[t] <- 1-p[t]
   # Main diagonal
   pr.j[t,t] <- phij[t]*p[t]
   # Above main diagonal
   for (j in (t+1):(nyears-1)){
      pr.j[t,j] <- phij[t]*prod(phia[(t+1):j])*prod(q[t:(j-1)])*p[j]
      } # j
   # Below main diagonal
   for (j in 1:(t-1)){
      pr.j[t,j] <- 0
      } # j
   # Last column
   pr.j[t,nyears] <- 1-sum(pr.j[t,1:(nyears-1)])
   } # t

# m-array cell probabilities for adults
for (t in 1:(nyears-1)){
```

```
   # Main diagonal
   pr.a[t,t] <- phia[t]*p[t]
   # above main diagonal
   for (j in (t+1):(nyears-1)){
      pr.a[t,j] <- prod(phia[t:j])*prod(q[t:(j-1)])*p[j]
      } # j
   # Below main diagonal
   for (j in 1:(t-1)){
      pr.a[t,j] <- 0
      } # j
   # Last column
   pr.a[t,nyears] <- 1-sum(pr.a[t,1:(nyears-1)])
   } # t

############################################################
# 4.3. Likelihood for reproductive data: Poisson regression
############################################################
for (t in 1:nyears){
   J[t] ~ dpois(rho[t])
   rho[t] <- R[t] * f[t]
   }

} # End Model
",fill = TRUE)
sink()


# Bundle data
bugs.data <- list(nyears = nyears, marray.j = marray.j, marray.a =
marray.a, y = popcount, J = J, R = R, t.pred = t.pred)

# Initial values
inits <- function(){list(mphij = runif(1, 0.05, 0.2), mphia = runif(1, 0.2,
0.5), mfec = runif(1, 4, 6), mim = runif(1, 0, 0.3), mp = runif(1, 0.5,
0.8), sig.phij = runif(1, 0.1, 1), sig.phia = runif(1, 0.1, 1), sig.fec =
runif(1, 0.1, 1), sig.im = runif(1, 0.1, 1), N1 =
round(runif(nyears+t.pred, 10, 20), 0), NadSurv =
round(runif(nyears+t.pred, 10, 30), 0), Nadimm = round(runif(nyears+t.pred,
1, 20), 0))}

# Parameters monitored
parameters <- c("phij", "phia", "f", "omega", "p", "lambda", "mphij",
"mphia", "mfec", "mim", "mlam", "sig.phij", "sig.phia", "sig.fec",
"sig.im", "N1", "NadSurv", "Nadimm", "Ntot")

# MCMC settings
niter <- 10000
nthin <- 3
nburn <- 5000
nchains <- 3

# Call WinBUGS from R (BRT 4 min)
ipm.hoopoe.1 <- bugs(bugs.data, inits, parameters, "ipm.hoopoe.bug",
n.chains = nchains, n.thin = nthin, n.iter = niter, n.burnin = nburn, debug
= TRUE, bugs.directory = bugs.dir, working.directory = getwd())
```

It may happen that WinBUGS produces an error message when the model is run. This has to do with the initial values that are generated randomly (within the limits specified above). A solution to this problem would be to specify the initial values in a different way, so that they are generated only in such a way that MCMC sampling could start properly. This is, however, not an easy task, and much trial and error may be required. The other, more practical option is to simply start WinBUGS again, until initial values are generated such that WinBUGS can start updating properly.

```
# Inspect results
print(ipm.hoopoe1, 3)
Inference for Bugs model at "ipm.hoopoe.bug", fit using WinBUGS,
 3 chains, each with 10000 iterations (first 5000 discarded), n.thin = 3
 n.sims = 5001 iterations saved
              mean      sd   2.5%    25%    50%    75%  97.5%  Rhat n.eff
phij[1]      0.116   0.019  0.080  0.103  0.115  0.128  0.155 1.002  1700
phij[2]      0.155   0.022  0.117  0.139  0.154  0.169  0.200 1.004   620
phij[3]      0.148   0.017  0.118  0.136  0.147  0.160  0.184 1.004   640
phij[4]      0.115   0.013  0.090  0.106  0.115  0.124  0.142 1.001  5000
phij[5]      0.135   0.015  0.107  0.124  0.134  0.144  0.166 1.003  1000
phij[6]      0.093   0.014  0.066  0.083  0.092  0.102  0.123 1.004   660
phij[7]      0.110   0.014  0.083  0.100  0.109  0.119  0.138 1.002  1800
phij[8]      0.124   0.016  0.094  0.113  0.123  0.134  0.158 1.001  5000
phij[9]      0.127   0.042  0.062  0.104  0.122  0.144  0.217 1.001  5000
phij[10]     0.129   0.041  0.064  0.105  0.124  0.145  0.230 1.001  3000
phij[11]     0.127   0.041  0.062  0.104  0.123  0.144  0.216 1.001  5000
phia[1]      0.400   0.037  0.314  0.382  0.403  0.421  0.465 1.005   840
phia[2]      0.422   0.036  0.362  0.399  0.417  0.440  0.509 1.001  5000
phia[3]      0.416   0.030  0.360  0.397  0.414  0.434  0.484 1.002  3200
phia[4]      0.419   0.029  0.367  0.400  0.417  0.436  0.485 1.001  5000
phia[5]      0.386   0.032  0.312  0.366  0.390  0.409  0.440 1.002  1300
phia[6]      0.403   0.027  0.346  0.386  0.404  0.420  0.454 1.001  5000
phia[7]      0.411   0.028  0.356  0.393  0.410  0.428  0.469 1.002  3400
phia[8]      0.424   0.033  0.368  0.402  0.420  0.443  0.502 1.001  5000
phia[9]      0.410   0.045  0.317  0.389  0.409  0.431  0.506 1.003  5000
phia[10]     0.410   0.045  0.316  0.388  0.409  0.432  0.506 1.002  5000
phia[11]     0.410   0.045  0.316  0.389  0.410  0.432  0.506 1.006  1600
f[1]         6.651   0.392  5.905  6.386  6.635  6.906  7.473 1.001  5000
f[2]         7.238   0.423  6.473  6.933  7.218  7.517  8.111 1.002  1900
f[3]         6.854   0.313  6.278  6.637  6.845  7.057  7.492 1.001  5000
f[4]         6.860   0.280  6.320  6.669  6.851  7.044  7.440 1.001  5000
f[5]         6.408   0.255  5.923  6.235  6.402  6.577  6.925 1.001  5000
f[6]         5.891   0.254  5.392  5.716  5.893  6.064  6.375 1.002  1600
f[7]         6.108   0.255  5.620  5.932  6.108  6.281  6.621 1.001  4500
f[8]         6.173   0.261  5.661  6.002  6.170  6.344  6.689 1.002  1800
f[9]         6.070   0.250  5.586  5.897  6.065  6.241  6.561 1.001  3200
f[10]        6.500   0.754  5.150  6.049  6.449  6.879  8.182 1.001  5000
f[11]        6.489   0.721  5.135  6.064  6.449  6.872  8.096 1.001  5000
omega[1]     0.324   0.174  0.091  0.223  0.289  0.375  0.767 1.003  1300
omega[2]     0.359   0.198  0.109  0.240  0.307  0.420  0.906 1.003  2900
omega[3]     0.313   0.143  0.095  0.226  0.288  0.368  0.685 1.021   320
omega[4]     0.237   0.098  0.048  0.172  0.239  0.298  0.438 1.022   370
omega[5]     0.231   0.098  0.045  0.165  0.233  0.292  0.425 1.006   710
omega[6]     0.248   0.099  0.055  0.186  0.247  0.306  0.456 1.044   270
omega[7]     0.247   0.101  0.058  0.181  0.246  0.306  0.467 1.011  1500
omega[8]     0.282   0.122  0.074  0.206  0.269  0.340  0.583 1.009   900
omega[9]     0.814  12.433  0.050  0.197  0.270  0.353  1.632 1.015  5000
omega[10]    1.361  23.960  0.052  0.198  0.269  0.354  1.510 1.013  5000
omega[11]    1.892  91.301  0.055  0.200  0.272  0.360  1.530 1.007  2300
p[1]         0.715   0.026  0.663  0.697  0.715  0.732  0.764 1.001  5000
p[2]         0.715   0.026  0.663  0.697  0.715  0.732  0.764 1.001  5000
p[3]         0.715   0.026  0.663  0.697  0.715  0.732  0.764 1.001  5000
p[4]         0.715   0.026  0.663  0.697  0.715  0.732  0.764 1.001  5000
p[5]         0.715   0.026  0.663  0.697  0.715  0.732  0.764 1.001  5000
p[6]         0.715   0.026  0.663  0.697  0.715  0.732  0.764 1.001  5000
p[7]         0.715   0.026  0.663  0.697  0.715  0.732  0.764 1.001  5000
p[8]         0.715   0.026  0.663  0.697  0.715  0.732  0.764 1.001  5000
p[9]         0.715   0.026  0.663  0.697  0.715  0.732  0.764 1.001  5000
p[10]        0.715   0.026  0.663  0.697  0.715  0.732  0.764 1.001  5000
p[11]        0.715   0.026  0.663  0.697  0.715  0.732  0.764 1.001  5000
lambda[1]    1.180   0.184  0.870  1.055  1.161  1.284  1.600 1.001  4700
lambda[2]    1.437   0.196  1.119  1.303  1.415  1.540  1.905 1.002  2500
lambda[3]    1.280   0.142  1.035  1.183  1.270  1.365  1.597 1.002  1200
lambda[4]    1.021   0.104  0.829  0.952  1.018  1.085  1.239 1.003  1000
```

lambda[5]	1.008	0.102	0.819	0.938	1.003	1.072	1.225	1.001	2500
lambda[6]	0.911	0.095	0.743	0.846	0.904	0.970	1.117	1.001	5000
lambda[7]	0.980	0.104	0.791	0.908	0.973	1.043	1.194	1.002	2300
lambda[8]	1.100	0.124	0.883	1.015	1.094	1.178	1.363	1.002	1500
lambda[9]	1.605	12.481	0.713	0.933	1.067	1.218	2.468	1.023	1600
lambda[10]	2.186	23.950	0.735	0.960	1.099	1.262	2.412	1.046	1000
lambda[11]	2.712	91.275	0.746	0.964	1.097	1.262	2.350	1.016	1900
mphij	0.123	0.015	0.094	0.114	0.123	0.131	0.154	1.003	1700
mphia	0.409	0.022	0.367	0.396	0.410	0.424	0.452	1.001	5000
mfec	6.463	0.250	5.992	6.307	6.453	6.612	6.981	1.001	5000
mim	0.293	0.159	0.137	0.225	0.271	0.324	0.550	1.005	2100
mlam	1.096	0.024	1.051	1.079	1.096	1.112	1.146	1.001	5000
sig.phij	0.291	0.153	0.067	0.196	0.265	0.352	0.658	1.040	180
sig.phia	0.134	0.110	0.002	0.054	0.111	0.188	0.396	1.188	45
sig.fec	0.097	0.040	0.039	0.071	0.091	0.117	0.194	1.007	460
sig.im	0.529	0.574	0.008	0.146	0.388	0.705	2.046	1.010	330
N1[1]	12.818	9.081	0.505	5.256	11.250	18.980	32.880	1.002	1200
N1[2]	15.660	4.446	7.283	12.600	15.540	18.690	24.540	1.002	2400
N1[3]	26.572	6.092	14.980	22.520	26.450	30.600	38.360	1.002	1600
N1[4]	32.942	6.633	20.580	28.380	32.690	37.150	47.110	1.003	780
N1[5]	30.300	6.309	18.590	25.920	30.150	34.390	43.350	1.002	1900
N1[6]	33.280	6.482	21.570	28.800	33.050	37.530	46.880	1.004	630
N1[7]	21.702	5.400	11.720	17.940	21.480	25.240	32.630	1.002	1700
N1[8]	24.073	5.469	13.950	20.210	23.840	27.790	34.970	1.001	3800
N1[9]	27.720	6.110	16.380	23.430	27.500	31.580	40.240	1.001	3200
N1[10]	29.895	11.630	12.000	22.000	28.000	36.000	56.000	1.001	5000
N1[11]	51.523	382.006	12.000	25.000	33.000	44.000	102.000	1.005	2300
N1[12]	181.270	2556.199	12.000	26.000	37.000	51.000	176.000	1.018	1600
NadSurv[1]	12.273	8.876	0.419	4.890	10.580	18.230	32.310	1.002	1500
NadSurv[2]	15.767	3.641	9.000	13.000	16.000	18.000	23.000	1.001	5000
NadSurv[3]	19.420	3.966	12.000	17.000	19.000	22.000	28.000	1.001	3800
NadSurv[4]	26.592	4.653	18.000	23.000	26.000	30.000	36.000	1.001	5000
NadSurv[5]	32.675	5.239	23.000	29.000	33.000	36.000	43.000	1.001	5000
NadSurv[6]	30.099	5.175	20.000	27.000	30.000	34.000	41.000	1.001	4600
NadSurv[7]	32.251	5.159	23.000	29.000	32.000	36.000	43.000	1.001	5000
NadSurv[8]	29.895	4.775	21.000	27.000	30.000	33.000	39.000	1.002	2100
NadSurv[9]	30.495	4.980	21.000	27.000	30.000	34.000	41.000	1.001	5000
NadSurv[10]	31.774	6.298	20.000	28.000	32.000	36.000	45.000	1.001	5000
NadSurv[11]	51.050	392.139	18.000	28.000	34.000	41.000	78.000	1.014	1600
NadSurv[12]	193.513	2983.928	17.000	29.000	37.000	49.000	161.000	1.029	760
Nadimm[1]	12.636	8.957	0.495	5.213	11.040	18.680	32.580	1.002	2400
Nadimm[2]	12.427	5.639	2.953	8.550	11.800	15.560	25.580	1.001	3000
Nadimm[3]	16.363	7.677	4.339	11.070	15.130	20.280	35.220	1.003	2400
Nadimm[4]	19.741	8.519	5.563	14.030	18.780	24.020	40.450	1.017	390
Nadimm[5]	17.549	7.360	3.237	12.600	17.560	22.370	32.990	1.028	340
Nadimm[6]	17.368	7.413	3.423	12.270	17.260	22.260	32.610	1.006	570
Nadimm[7]	19.216	7.645	4.052	14.240	19.250	23.990	34.850	1.033	260
Nadimm[8]	17.325	7.131	3.917	12.520	17.170	21.870	32.440	1.009	910
Nadimm[9]	19.823	8.281	4.910	14.220	19.100	24.580	38.710	1.012	580
Nadimm[10]	62.349	932.780	3.000	14.000	21.000	29.000	119.000	1.245	2300
Nadimm[11]	369.909	7016.466	3.000	15.000	22.000	33.000	205.000	1.188	1200
Nadimm[12]	3240.962	153229.811	4.000	16.000	25.000	40.000	323.000	1.285	5000
Ntot[1]	37.727	5.363	27.950	34.090	37.440	41.200	48.680	1.001	5000
Ntot[2]	43.854	5.054	34.100	40.530	43.820	47.160	54.100	1.001	5000
Ntot[3]	62.354	6.048	51.070	58.130	62.120	66.300	74.770	1.001	3600
Ntot[4]	79.275	7.045	66.950	74.210	78.890	83.770	94.020	1.001	2700
Ntot[5]	80.524	6.714	67.970	75.860	80.450	84.990	93.910	1.001	3000
Ntot[6]	80.746	6.824	67.950	76.000	80.570	85.260	94.480	1.002	1500
Ntot[7]	73.169	6.394	61.130	68.770	72.890	77.350	86.040	1.002	1500
Ntot[8]	71.294	6.352	59.220	66.960	71.070	75.330	84.410	1.001	3100
Ntot[9]	78.038	7.761	63.780	72.680	77.680	83.200	94.230	1.001	5000
Ntot[10]	124.019	932.929	51.000	71.000	83.000	97.000	191.000	1.019	1800
Ntot[11]	472.474	7336.588	46.000	74.000	92.000	116.000	377.000	1.033	830
Ntot[12]	3615.385	153757.183	44.000	77.000	101.000	137.000	705.000	1.021	740
deviance	332.414	7.325	319.600	327.200	331.900	336.900	348.500	1.003	870

From the parameter estimates we see that the population is predicted to increase. We see that the uncertainty in the population projections is very large (and increases with increasing projection distance); this is mainly due to great uncertainty in immigration.
To compute the extinction probability, we count how in many of the MCMC samples was the estimated population size in the last year below the defined extinction threshold, and this is done by the following line of code:

```
# Compute extinction probability
mean(ipm.hoopoe.1$sims.list$Ntot[,12]<5)
[1] 0
```

Exercise 2

Task: Assume that population count data from years 3 and 5 are missing in the hoopoe example. Use an integrated population model to estimate these missing data. What do you observe?

Solution: In the data was replace the population counts in year 3 and 5 by NA. No changes in the analysing code are required.

Load data

```
nyears <- 9   # Number of years

# Capture recapture data: m-array of juveniles and adults (these are males
and females together)
marray.j <- matrix (c(15, 3, 0, 0, 0, 0, 0, 0, 198, 0, 34, 9, 1, 0, 0, 0,
0, 287, 0, 0, 56, 8, 1, 0, 0, 0, 455, 0, 0, 0, 48, 3, 1, 0, 0, 518, 0, 0,
0, 0, 45, 13, 2, 0, 463, 0, 0, 0, 0, 0, 27, 7, 0, 493, 0, 0, 0, 0, 0, 0,
37, 3, 434, 0, 0, 0, 0, 0, 0, 0, 39, 405), nrow = 8, ncol = 9, byrow =
TRUE)
marray.a <- matrix(c(14, 2, 0, 0, 0, 0, 0, 0, 43, 0, 22, 4, 0, 0, 0, 0, 0,
44, 0, 0, 34, 2, 0, 0, 0, 0, 79, 0, 0, 0, 51, 3, 0, 0, 0, 94, 0, 0, 0, 0,
45, 3, 0, 0, 118, 0, 0, 0, 0, 0, 44, 3, 0, 113, 0, 0, 0, 0, 0, 0, 48, 2,
99, 0, 0, 0, 0, 0, 0, 0, 51, 90), nrow = 8, ncol = 9, byrow = TRUE)

# Population population count data
popcount <- c(32, 42, NA, 85, NA, 78, 73, 69, 79)

# Reproductive success
J <- c(189, 274, 398, 538, 520, 476, 463, 438, 507) # number offspring
R <- c(28, 36, 57, 77, 81, 83, 77, 72, 85)          # number of surveyed
broods
```

Data analysis

```
# Specify model in BUGS language
sink("ipm.hoopoe.bug")
cat("
model {
##########################################################################
#  Integrated population model
#  - Age structured model with 2 age classes:
#          1-year old and at least 2 years old
#  - Age at first breeding = 1 year
#  - Pre-breeding census, female-based
#  - All vital rates are assumed to be time-dependent (random)
```

```
#  - Explicit estimate of immigration
###########################################################################

############################################################
# 1. Define the priors for the parameters
############################################################

# Initial population sizes
N1[1] ~ dnorm(10, 0.001)I(0,)          # 1-year old individuals
NadSurv[1] ~ dnorm(10, 0.001)I(0,)     # Adults >= 2 years
Nadimm[1] ~ dnorm(10, 0.001)I(0,)      # Immigrants

# Mean demographic parameters
mphij ~ dunif(0, 1)
mphia ~ dunif(0, 1)
mfec ~ dunif(0, 15)
mim ~ dunif(0, 3)
mp ~ dunif(0, 1)

# Precision of standard deviations of temporal variability
sig.phij ~ dunif(0, 10)
tau.phij <- pow(sig.phij, -2)
sig.phia ~ dunif(0, 10)
tau.phia <- pow(sig.phia, -2)
sig.fec ~ dunif(0, 10)
tau.fec <- pow(sig.fec, -2)
sig.im ~ dunif(0, 10)
tau.im <- pow(sig.im, -2)

# Distribution of error terms (Bounded to help with convergence)
for (t in 1:(nyears-1)){
   epsilon.phij[t] ~ dnorm(0, tau.phij)I(-15,15)
   epsilon.phia[t] ~ dnorm(0, tau.phia)I(-15,15)
   epsilon.im[t] ~ dnorm(0, tau.im)I(-15,15)
   }

for (t in 1:nyears){
   epsilon.fec[t] ~ dnorm(0, tau.fec)I(-15,15)
   }

############################################################
# 2. Constrain parameters
############################################################
for (t in 1:(nyears-1)){
   logit(phij[t]) <- l.mphij + epsilon.phij[t]  # Juv. apparent survival
   logit(phia[t]) <- l.mphia + epsilon.phia[t]  # Adult apparent survival
   log(omega[t]) <- l.mim + epsilon.im[t]       # Immigration
   p[t] <- mp                                   # Recapture probability
   }

# Fecundity: note data from an additional year compared to the other vital
rates is available
for (t in 1:nyears){
   log(f[t]) <- l.mfec + epsilon.fec[t]
   }

############################################################
# 3. Derived parameters
############################################################

l.mphij <- log(mphij / (1-mphij))     # Logit mean juv. survival
l.mphia <- log(mphia / (1-mphia))     # Logit mean adult survival
```

```
l.mfec <- log(mfec)                      # Log mean fecundity
l.mim <- log(mim)                        # Log mean immigration rate

# Population growth rate
for (t in 1:(nyears-1)){
   lambda[t] <- Ntot[t+1] / Ntot[t]
   logla[t] <- log(lambda[t])
   }
mlam <- exp((1/(nyears-1))*sum(logla[1:(nyears-1)]))   # Geometric mean


######################################################################
# 4. The likelihoods of the single data sets
######################################################################

###################################################################
# 4.1. Likelihood for population count data (state-space model)
###################################################################

   #############################
   # 4.1.1 System process
   #############################
   for (t in 2:nyears){
      mean1[t] <- 0.5 * f[t-1] * phij[t-1] * Ntot[t-1]
      N1[t] ~ dpois(mean1[t])
      NadSurv[t] ~ dbin(phia[t-1], Ntot[t-1])
      mpo[t] <- Ntot[t-1] * omega[t-1]
      Nadimm[t] ~ dpois(mpo[t])
      }

   #############################
   # 4.1.2 Observation process
   #############################
   for (t in 1:nyears){
      Ntot[t] <- NadSurv[t] + Nadimm[t] + N1[t]
      y[t] ~ dpois(Ntot[t])
      }

######################################################################
# 4.2 Likelihood for capture-recapture data: CJS model (2 age classes)
######################################################################

# Multinomial likelihood
for (t in 1:(nyears-1)){
   marray.j[t,1:nyears] ~ dmulti(pr.j[t,], r.j[t])
   marray.a[t,1:nyears] ~ dmulti(pr.a[t,], r.a[t])
   }

# Calculate number of released individuals
for (t in 1:(nyears-1)){
   r.j[t] <- sum(marray.j[t,])
   r.a[t] <- sum(marray.a[t,])
   }

# m-array cell probabilities for juveniles
for (t in 1:(nyears-1)){
   q[t] <- 1-p[t]
   # Main diagonal
   pr.j[t,t] <- phij[t]*p[t]
   # Above main diagonal
   for (j in (t+1):(nyears-1)){
      pr.j[t,j] <- phij[t]*prod(phia[(t+1):j])*prod(q[t:(j-1)])*p[j]
```

```
      } # j
   # Below main diagonal
   for (j in 1:(t-1)){
      pr.j[t,j] <- 0
      } # j
   # Last column
   pr.j[t,nyears] <- 1-sum(pr.j[t,1:(nyears-1)])
   } # t

# m-array cell probabilities for adults
for (t in 1:(nyears-1)){
   # Main diagonal
   pr.a[t,t] <- phia[t]*p[t]
   # above main diagonal
   for (j in (t+1):(nyears-1)){
      pr.a[t,j] <- prod(phia[t:j])*prod(q[t:(j-1)])*p[j]
      } # j
   # Below main diagonal
   for (j in 1:(t-1)){
      pr.a[t,j] <- 0
      } # j
   # Last column
   pr.a[t,nyears] <- 1-sum(pr.a[t,1:(nyears-1)])
   } # t

############################################################
# 4.3. Likelihood for reproductive data: Poisson regression
############################################################
for (t in 1:nyears){
   J[t] ~ dpois(rho[t])
   rho[t] <- R[t] * f[t]
   }
} # End Model
",fill = TRUE)
sink()


# Bundle data
bugs.data <- list(nyears = nyears, marray.j = marray.j, marray.a =
marray.a, y = popcount, J = J, R = R)

# Initial values
inits <- function(){list(mphij = runif(1, 0.05, 0.2), mphia = runif(1, 0.2,
0.5), mfec = runif(1, 4, 6), mim = runif(1, 0, 0.3), mp = runif(1, 0.5,
0.8), sig.phij = runif(1, 0.1, 1), sig.phia = runif(1, 0.1, 1), sig.fec =
runif(1, 0.1, 1), sig.im = runif(1, 0.1, 1), N1 = round(runif(nyears, 10,
20), 0), NadSurv = round(runif(nyears, 10, 30), 0), Nadimm =
round(runif(nyears, 1, 20), 0))}


# Parameters monitored
parameters <- c("phij", "phia", "f", "omega", "p", "lambda", "mphij",
"mphia", "mfec", "mim", "mlam", "sig.phij", "sig.phia", "sig.fec",
"sig.im", "N1", "NadSurv", "Nadimm", "Ntot")

# MCMC settings
niter <- 10000
nthin <- 3
nburn <- 5000
nchains <- 3

# Call WinBUGS from R (BRT 4 min)
```

```
ipm.hoopoe.2 <- bugs(bugs.data, inits, parameters, "ipm.hoopoe.bug",
n.chains = nchains, n.thin = nthin, n.iter = niter, n.burnin = nburn, debug
= TRUE, bugs.directory = bugs.dir, working.directory = getwd())

# Inspect results
print(imp.hoopoe.2, 3)
Inference for Bugs model at "ipm.hoopoe.bug", fit using WinBUGS,
 3 chains, each with 10000 iterations (first 5000 discarded), n.thin = 3
 n.sims = 5001 iterations saved
             mean    sd   2.5%    25%    50%    75%  97.5% Rhat n.eff
phij[1]     0.115 0.019  0.079  0.103  0.115  0.128  0.154 1.002  2500
phij[2]     0.152 0.021  0.116  0.137  0.150  0.165  0.196 1.002  1800
phij[3]     0.147 0.017  0.117  0.135  0.146  0.158  0.182 1.001  5000
phij[4]     0.115 0.014  0.090  0.106  0.115  0.124  0.143 1.001  4200
phij[5]     0.135 0.015  0.108  0.125  0.134  0.145  0.166 1.001  5000
phij[6]     0.094 0.015  0.067  0.084  0.094  0.104  0.125 1.001  5000
phij[7]     0.111 0.015  0.084  0.101  0.111  0.121  0.142 1.001  4000
phij[8]     0.125 0.016  0.095  0.114  0.124  0.135  0.159 1.001  5000
phia[1]     0.399 0.037  0.313  0.380  0.403  0.422  0.468 1.003   880
phia[2]     0.422 0.036  0.361  0.399  0.418  0.441  0.507 1.002  3000
phia[3]     0.416 0.031  0.358  0.397  0.414  0.434  0.485 1.003  2800
phia[4]     0.421 0.030  0.367  0.401  0.418  0.438  0.490 1.004  1100
phia[5]     0.386 0.034  0.310  0.365  0.390  0.410  0.441 1.006   410
phia[6]     0.405 0.027  0.350  0.388  0.405  0.422  0.458 1.001  5000
phia[7]     0.411 0.028  0.357  0.394  0.411  0.428  0.470 1.002  5000
phia[8]     0.425 0.033  0.370  0.403  0.421  0.444  0.502 1.003  1200
f[1]        6.650 0.388  5.936  6.383  6.628  6.894  7.470 1.001  2700
f[2]        7.227 0.428  6.431  6.921  7.208  7.511  8.099 1.001  3400
f[3]        6.854 0.317  6.264  6.637  6.843  7.058  7.504 1.002  1500
f[4]        6.851 0.273  6.340  6.661  6.843  7.027  7.419 1.001  3000
f[5]        6.422 0.255  5.930  6.251  6.420  6.587  6.947 1.001  5000
f[6]        5.892 0.253  5.401  5.721  5.888  6.067  6.384 1.002  1200
f[7]        6.114 0.253  5.610  5.946  6.113  6.287  6.604 1.001  5000
f[8]        6.181 0.259  5.689  6.005  6.179  6.354  6.695 1.001  3000
f[9]        6.073 0.249  5.581  5.904  6.077  6.246  6.554 1.001  5000
omega[1]    0.325 0.195  0.045  0.204  0.286  0.397  0.858 1.004  5000
omega[2]    0.383 0.295  0.054  0.216  0.303  0.447  1.210 1.019   140
omega[3]    0.387 0.280  0.046  0.222  0.311  0.468  1.175 1.011   290
omega[4]    0.225 0.120  0.022  0.141  0.221  0.296  0.488 1.003  2100
omega[5]    0.223 0.119  0.020  0.139  0.219  0.293  0.493 1.009   630
omega[6]    0.238 0.113  0.032  0.162  0.233  0.304  0.487 1.011  3100
omega[7]    0.239 0.120  0.024  0.159  0.233  0.305  0.508 1.003  1100
omega[8]    0.283 0.143  0.056  0.191  0.266  0.350  0.626 1.006   810
p[1]        0.714 0.027  0.661  0.696  0.715  0.733  0.766 1.002  2400
p[2]        0.714 0.027  0.661  0.696  0.715  0.733  0.766 1.002  2400
p[3]        0.714 0.027  0.661  0.696  0.715  0.733  0.766 1.002  2400
p[4]        0.714 0.027  0.661  0.696  0.715  0.733  0.766 1.002  2400
p[5]        0.714 0.027  0.661  0.696  0.715  0.733  0.766 1.002  2400
p[6]        0.714 0.027  0.661  0.696  0.715  0.733  0.766 1.002  2400
p[7]        0.714 0.027  0.661  0.696  0.715  0.733  0.766 1.002  2400
p[8]        0.714 0.027  0.661  0.696  0.715  0.733  0.766 1.002  2400
lambda[1]   1.165 0.195  0.845  1.032  1.143  1.274  1.612 1.001  3600
lambda[2]   1.418 0.297  0.973  1.235  1.372  1.535  2.176 1.011   190
lambda[3]   1.364 0.288  0.941  1.185  1.314  1.477  2.101 1.012   190
lambda[4]   1.013 0.136  0.766  0.921  1.007  1.093  1.312 1.004   690
lambda[5]   1.018 0.137  0.779  0.922  1.008  1.099  1.312 1.002  1600
lambda[6]   0.913 0.104  0.730  0.841  0.906  0.976  1.135 1.001  2700
lambda[7]   0.983 0.113  0.786  0.904  0.975  1.051  1.228 1.002  1800
lambda[8]   1.103 0.135  0.876  1.009  1.090  1.184  1.406 1.002  1200
mphij       0.126 0.020  0.100  0.115  0.123  0.132  0.161 1.048   330
mphia       0.411 0.023  0.369  0.397  0.410  0.425  0.458 1.002  2900
mfec        6.481 0.257  6.014  6.320  6.466  6.620  7.020 1.008   270
mim         0.291 0.182  0.102  0.213  0.265  0.325  0.632 1.007  1900
mlam        1.098 0.025  1.051  1.081  1.097  1.114  1.150 1.001  5000
sig.phij    0.282 0.164  0.071  0.182  0.253  0.343  0.674 1.002  1300
sig.phia    0.139 0.112  0.009  0.055  0.114  0.193  0.416 1.018   130
sig.fec     0.096 0.040  0.038  0.069  0.090  0.115  0.193 1.002  2000
sig.im      0.758 0.743  0.031  0.274  0.552  0.973  2.936 1.005   600
```

```
N1[1]        12.757  9.011   0.516   5.212  11.400  18.680  32.560 1.001  5000
N1[2]        15.137  4.573   6.729  11.890  14.990  18.200  24.550 1.002  1700
N1[3]        24.903  6.461  13.060  20.390  24.420  29.040  38.300 1.001  2600
N1[4]        31.673  7.800  17.080  26.250  31.460  36.780  47.480 1.008   280
N1[5]        30.297  6.925  17.810  25.480  29.990  34.790  44.850 1.001  5000
N1[6]        33.573  7.029  20.770  28.720  33.280  38.050  47.980 1.001  4000
N1[7]        22.154  5.742  11.960  18.050  21.810  25.970  34.070 1.001  5000
N1[8]        24.676  5.739  14.080  20.770  24.440  28.370  36.610 1.002  1400
N1[9]        27.717  6.250  16.180  23.270  27.530  31.860  40.710 1.001  4500
NadSurv[1]   12.125  8.593   0.472   5.076  10.630  17.650  31.320 1.001  2600
NadSurv[2]   15.486  3.742   8.000  13.000  15.000  18.000  23.000 1.001  4000
NadSurv[3]   18.542  4.162  11.000  16.000  18.000  21.000  27.000 1.001  5000
NadSurv[4]   25.610  5.908  15.000  22.000  25.000  29.000  38.000 1.007   300
NadSurv[5]   32.837  5.825  22.000  29.000  33.000  37.000  45.000 1.001  5000
NadSurv[6]   30.208  5.760  20.000  26.000  30.000  34.000  42.000 1.001  5000
NadSurv[7]   32.380  5.276  22.000  29.000  32.000  36.000  43.000 1.001  5000
NadSurv[8]   29.738  4.863  21.000  26.000  30.000  33.000  40.000 1.001  4900
NadSurv[9]   30.509  5.192  21.000  27.000  30.000  34.000  41.000 1.003   800
Nadimm[1]    12.379  8.865   0.541   5.125  10.790  18.120  32.410 1.003   840
Nadimm[2]    12.100  6.272   1.453   7.732  11.470  15.880  26.240 1.002  4300
Nadimm[3]    16.594 11.383   1.904   9.198  14.030  20.480  46.960 1.023   120
Nadimm[4]    21.888 11.804   3.440  13.830  19.710  28.390  50.770 1.009   360
Nadimm[5]    16.751  9.267   1.487  10.390  16.040  22.180  37.730 1.002  2400
Nadimm[6]    16.341  8.342   1.306  10.440  16.280  21.780  33.900 1.009   710
Nadimm[7]    18.165  8.390   2.232  12.350  18.110  23.610  35.260 1.011  4000
Nadimm[8]    16.618  8.040   1.651  11.370  16.350  21.730  33.120 1.003  1100
Nadimm[9]    19.660  9.172   3.821  13.370  18.940  25.150  40.730 1.005  1100
Ntot[1]      37.261  5.401  27.130  33.520  37.080  40.750  48.510 1.001  5000
Ntot[2]      42.724  5.431  32.450  39.020  42.520  46.200  53.920 1.001  5000
Ntot[3]      60.039 11.862  39.460  52.420  58.880  66.240  86.500 1.013   160
Ntot[4]      79.171  7.965  65.130  73.600  78.650  84.250  96.390 1.002  1300
Ntot[5]      79.886 11.255  60.260  72.110  79.090  86.780 103.700 1.001  3400
Ntot[6]      80.121  7.337  66.200  75.050  79.860  85.010  94.540 1.001  5000
Ntot[7]      72.699  6.610  60.300  68.180  72.560  77.030  86.200 1.001  3100
Ntot[8]      71.033  6.643  58.380  66.600  70.920  75.260  84.620 1.003   790
Ntot[9]      77.887  8.021  62.950  72.490  77.630  82.950  94.720 1.001  5000
deviance    319.726  7.446 306.900 314.400 319.100 324.400 335.900 1.001  5000
```

The only difference to the analysis of the complete data set is that the precision of the population size estimates of the third and fifth year are lower. The difference is small, because the population size of a given year is a function of the population size of the previous year and the demographic rates. Thus, although the count in one year is missing, there is plenty of information about population size in that year. If two or more years in a row are missing, the uncertainty increases.

Exercise 3

Task: Fit an integrated population model with time-dependent parameters to the ortolan bunting date (section 11.3 of the BPA book). Compare the population size estimates with those from a model with constant demographic parameters. Explain.

Solution: The only change in the model code compared to the one presented in section 11.3 of the BPA book is that we give a prior distribution to each of the year-specific parameters. To visualize the differences in the estimated population sizes under the time-constant and the time-dependent model, we produce a figure.

Load data

```
# Population count data
y <- c(45, 48, 44, 59, 62, 62, 55, 51, 46, 42)
```

Capture-recapture data (in m-array format)

```
m <- matrix(c(11,  0,  0,  0,  0,  0,  0,  0,  0,  70,
               0, 12,  0,  1,  0,  0,  0,  0,  0,  52,
               0,  0, 15,  5,  1,  0,  0,  0,  0,  42,
               0,  0,  0,  8,  3,  0,  0,  0,  0,  51,
               0,  0,  0,  0,  4,  3,  0,  0,  0,  61,
               0,  0,  0,  0,  0, 12,  2,  3,  0,  66,
               0,  0,  0,  0,  0,  0, 16,  5,  0,  44,
               0,  0,  0,  0,  0,  0,  0, 12,  0,  46,
               0,  0,  0,  0,  0,  0,  0,  0, 11,  71,
              10,  2,  0,  0,  0,  0,  0,  0,  0,  13,
               0,  7,  0,  1,  0,  0,  0,  0,  0,  27,
               0,  0, 13,  2,  1,  1,  0,  0,  0,  14,
               0,  0,  0, 12,  2,  0,  0,  0,  0,  20,
               0,  0,  0,  0, 10,  2,  0,  0,  0,  21,
               0,  0,  0,  0,  0, 11,  2,  1,  1,  14,
               0,  0,  0,  0,  0,  0, 12,  0,  0,  18,
               0,  0,  0,  0,  0,  0,  0, 11,  1,  21,
               0,  0,  0,  0,  0,  0,  0,  0, 10,  26), ncol = 10, byrow =
TRUE)
```

```
# Data on productivity
J <- c(64, 132,  86, 154, 156, 134, 116, 106, 110, 144)
R <- c(21, 28, 26, 38, 35, 33, 31, 30, 33, 34)
```

Data analysis

```
# Specify model in BUGS language
sink("ipm.bug")
cat("
model {
##########################################################################
#  Integrated population model
#  - Age structured model with 2 age classes:
#         1-year old and at least 2 years old
#  - Age at first breeding = 1 year
#  - Pre-breeding census, female-based
#  - All vital rates assumed to be constant
##########################################################################

############################################################
# 1. Define the priors for the parameters
############################################################

# Observation error
tauy <- pow(sigma.y, -2)
sigma.y ~ dunif(0, 50)
sigma2.y <- pow(sigma.y, 2)

# Initial population sizes
N1[1] ~ dnorm(100, 0.0001)I(0,)     # 1-year
Nad[1] ~ dnorm(100, 0.0001)I(0,)    # Adults

# Survival and recapture probabilities, as well as productivity
for (t in 1:(nyears-1)){
   sjuv[t] ~ dunif(0, 1)
   sad[t] ~ dunif(0, 1)
   p[t] ~ dunif(0, 1)
   f[t] ~ dunif(0, 20)
   }
```

```
###################################################################
# 2. Derived parameters
###################################################################
# Population growth rate
for (t in 1:(nyears-1)){
   lambda[t] <- Ntot[t+1] / Ntot[t]
   }

######################################################################
# 3. The likelihoods of the single data sets
######################################################################

###################################################################
# 3.1. Likelihood for population count data (state-space model)
###################################################################

   #############################
   # 3.1.1 System process
   #############################
   for (t in 2:nyears){
      mean1[t] <- f[t-1] / 2 * sjuv[t-1] * Ntot[t-1]
      N1[t] ~ dpois(mean1[t])
      Nad[t] ~ dbin(sad[t-1], Ntot[t-1])
      }
   for (t in 1:nyears){
      Ntot[t] <- Nad[t] + N1[t]
      }
   ###############################
   # 3.1.2 Observation process
   ###############################
   for (t in 1:nyears){
      y[t] ~ dnorm(Ntot[t], tauy)
      }

######################################################################
# 3.2 Likelihood for capture-recapture data: CJS model (2 age classes)
######################################################################

# Multinomial likelihood
for (t in 1:2*(nyears-1)){
   m[t,1:nyears] ~ dmulti(pr[t,], r[t])
   }

# Calculate the number of released individuals
for (t in 1:2*(nyears-1)){
   r[t] <- sum(m[t,])
   }

# m-array cell probabilities for juveniles
for (t in 1:(nyears-1)){
   # Main diagonal
   q[t] <- 1-p[t]
   pr[t,t] <- sjuv[t] * p[t]
   # Above main diagonal
   for (j in (t+1):(nyears-1)){
      pr[t,j] <- sjuv[t]*prod(sad[(t+1):j])*prod(q[t:(j-1)])*p[j]
      } # j
   # Below main diagonal
   for (j in 1:(t-1)){
      pr[t,j] <- 0
      } # j
   # Last column: probability of non-recapture
```

```
   pr[t,nyears] <- 1-sum(pr[t,1:(nyears-1)])
   } # t

# m-array cell probabilities for adults
for (t in 1:(nyears-1)){
   # Main diagonal
   pr[t+nyears-1,t] <- sad[t] * p[t]
   # Above main diagonal
   for (j in (t+1):(nyears-1)){
      pr[t+nyears-1,j] <- prod(sad[t:j])*prod(q[t:(j-1)])*p[j]
      }
   # Below main diagonal
   for (j in 1:(t-1)){
      pr[t+nyears-1,j] <- 0
      } # j
   # Last column
   pr[t+nyears-1,nyears] <- 1 - sum(pr[t+nyears-1,1:(nyears-1)])
   } # t

############################################################
# 3.3. Likelihood for reproductive data: Poisson regression
############################################################
for (t in 1:(nyears-1)){
   J[t] ~ dpois(rho[t])
   rho[t] <- R[t]*f[t]
   }
} # End Model
",fill = TRUE)
sink()


# Bundle data
bugs.data <- list(m = m, y = y, J = J, R = R, nyears = dim(m)[2])

# Initial values
inits <- function(){list(sjuv = runif(dim(m)[2]-1, 0, 1), sad =
runif(dim(m)[2]-1, 0, 1), p = runif(dim(m)[2]-1, 0, 1), f =
runif(dim(m)[2]-1, 0, 10), N1 = rpois(dim(m)[2], 30), Nad =
rpois(dim(m)[2], 30), sigma.y = runif(1 ,0, 10))}

# Parameters monitored
parameters <- c("sjuv", "sad", "p", "f", "N1", "Nad", "Ntot", "lambda",
"sigma2.y")

# MCMC settings
niter <- 20000
nthin <- 6
nburn <- 5000
nchains <- 3

# Call WinBUGS from R (BRT 3.5 min)
ipm.t <- bugs(bugs.data, inits, parameters, "ipm.bug", n.chains = nchains,
n.thin = nthin, n.iter = niter, n.burnin = nburn, debug = TRUE,
bugs.directory = bugs.dir, working.directory = getwd())

# Inspect results
print(ipm.t, digits = 3)
             mean     sd   2.5%    25%    50%    75%  97.5%  Rhat n.eff
sjuv[1]     0.206  0.056  0.112  0.166  0.200  0.239  0.334 1.001  7500
sjuv[2]     0.243  0.049  0.152  0.210  0.241  0.274  0.342 1.001  7500
sjuv[3]     0.462  0.085  0.312  0.403  0.455  0.514  0.646 1.001  7500
sjuv[4]     0.247  0.058  0.148  0.206  0.243  0.282  0.373 1.001  5300
sjuv[5]     0.159  0.046  0.078  0.126  0.157  0.190  0.257 1.001  7500
```

sjuv[6]	0.246	0.053	0.155	0.208	0.241	0.277	0.366	1.001	6200
sjuv[7]	0.369	0.072	0.245	0.319	0.363	0.411	0.526	1.001	7500
sjuv[8]	0.240	0.055	0.141	0.202	0.237	0.274	0.359	1.001	5500
sjuv[9]	0.259	0.081	0.133	0.203	0.250	0.302	0.441	1.001	7500
sad[1]	0.687	0.125	0.445	0.601	0.687	0.775	0.930	1.002	1500
sad[2]	0.264	0.072	0.139	0.212	0.259	0.308	0.420	1.001	7500
sad[3]	0.718	0.120	0.487	0.634	0.717	0.805	0.947	1.002	2100
sad[4]	0.579	0.105	0.387	0.506	0.574	0.647	0.799	1.001	7500
sad[5]	0.521	0.110	0.318	0.444	0.519	0.594	0.747	1.002	1800
sad[6]	0.570	0.112	0.375	0.493	0.562	0.638	0.818	1.001	4100
sad[7]	0.500	0.101	0.326	0.429	0.492	0.561	0.728	1.001	5600
sad[8]	0.481	0.090	0.316	0.419	0.477	0.540	0.664	1.001	7500
sad[9]	0.505	0.140	0.265	0.406	0.491	0.591	0.821	1.001	7500
p[1]	0.645	0.123	0.411	0.559	0.646	0.731	0.882	1.001	5200
p[2]	0.787	0.101	0.561	0.722	0.798	0.862	0.948	1.001	7500
p[3]	0.505	0.086	0.343	0.445	0.503	0.562	0.680	1.001	7200
p[4]	0.573	0.093	0.391	0.507	0.573	0.638	0.752	1.001	7500
p[5]	0.564	0.108	0.356	0.487	0.563	0.638	0.775	1.002	2400
p[6]	0.585	0.091	0.409	0.522	0.586	0.649	0.761	1.001	7500
p[7]	0.599	0.086	0.424	0.541	0.601	0.660	0.761	1.002	2600
p[8]	0.780	0.098	0.571	0.717	0.788	0.854	0.944	1.001	7500
p[9]	0.557	0.152	0.303	0.446	0.541	0.653	0.905	1.001	7500
f[1]	3.107	0.383	2.404	2.836	3.089	3.346	3.919	1.001	5200
f[2]	4.775	0.413	4.002	4.490	4.765	5.050	5.620	1.001	7500
f[3]	3.290	0.343	2.655	3.050	3.274	3.514	3.990	1.002	2800
f[4]	4.057	0.322	3.461	3.837	4.050	4.270	4.706	1.001	7200
f[5]	4.492	0.356	3.848	4.248	4.486	4.725	5.225	1.001	7300
f[6]	3.995	0.344	3.343	3.764	3.987	4.224	4.697	1.001	3200
f[7]	3.620	0.334	2.986	3.392	3.613	3.838	4.293	1.001	7500
f[8]	3.527	0.339	2.891	3.294	3.518	3.748	4.220	1.001	7500
f[9]	3.347	0.317	2.748	3.130	3.339	3.554	4.002	1.001	7500
N1[1]	23.569	14.034	1.203	11.660	23.130	34.632	50.095	1.011	300
N1[2]	15.206	4.950	6.558	11.650	14.850	18.462	25.685	1.002	2100
N1[3]	28.182	5.585	16.919	24.497	28.360	31.990	38.700	1.001	7500
N1[4]	29.622	5.852	18.770	25.647	29.440	33.332	41.826	1.002	2200
N1[5]	28.506	6.690	16.305	23.840	28.370	32.900	42.500	1.001	3500
N1[6]	22.440	6.987	9.526	17.440	22.240	27.170	36.700	1.001	7500
N1[7]	23.429	5.930	12.350	19.250	23.240	27.452	35.356	1.001	7500
N1[8]	30.070	6.589	17.950	25.707	29.690	34.060	44.370	1.001	4700
N1[9]	21.631	5.814	11.180	17.650	21.380	25.182	34.075	1.002	2900
N1[10]	19.523	6.064	9.149	15.427	19.190	23.020	32.670	1.001	6700
Nad[1]	23.699	13.880	1.149	12.097	23.380	34.662	48.925	1.007	380
Nad[2]	32.616	5.774	21.000	29.000	33.000	36.000	44.000	1.003	920
Nad[3]	12.864	4.243	5.000	10.000	13.000	16.000	22.000	1.001	7500
Nad[4]	29.113	5.455	18.000	25.000	29.000	33.000	40.000	1.002	1500
Nad[5]	33.741	6.406	21.000	29.000	34.000	38.000	46.000	1.001	7500
Nad[6]	32.777	7.337	18.000	28.000	33.000	38.000	47.000	1.002	1500
Nad[7]	29.457	5.847	18.000	25.000	29.000	33.000	41.000	1.002	1900
Nad[8]	24.299	5.517	14.000	20.000	24.000	28.000	36.000	1.001	7500
Nad[9]	25.341	5.389	15.000	22.000	25.000	29.000	36.000	1.001	3100
Nad[10]	23.354	6.139	12.000	19.000	23.000	27.000	36.000	1.001	7500
Ntot[1]	47.267	6.583	36.519	43.770	45.885	49.900	63.636	1.001	7500
Ntot[2]	47.823	5.773	36.005	45.037	47.800	50.360	60.476	1.002	1900
Ntot[3]	41.046	5.028	29.579	38.210	41.950	44.170	49.666	1.001	4100
Ntot[4]	58.735	5.930	44.674	56.150	58.990	61.540	70.805	1.003	7500
Ntot[5]	62.247	6.128	48.608	59.560	62.165	65.120	75.301	1.001	6500
Ntot[6]	55.217	8.032	35.343	50.590	57.440	61.402	65.626	1.002	2300
Ntot[7]	52.887	5.952	38.379	50.067	53.940	56.050	63.980	1.002	1900
Ntot[8]	54.369	6.449	43.785	50.680	52.950	57.312	69.872	1.001	4100
Ntot[9]	46.973	6.063	35.379	43.930	46.340	49.482	61.680	1.001	4700
Ntot[10]	42.877	7.319	29.399	39.490	42.190	45.570	60.121	1.001	4200
lambda[1]	1.024	0.145	0.719	0.940	1.038	1.101	1.327	1.002	2100
lambda[2]	0.867	0.127	0.597	0.795	0.880	0.937	1.123	1.001	4700
lambda[3]	1.447	0.195	1.133	1.327	1.407	1.546	1.928	1.001	7500
lambda[4]	1.068	0.131	0.826	1.000	1.055	1.124	1.383	1.002	4000
lambda[5]	0.892	0.137	0.583	0.806	0.920	0.993	1.107	1.002	1300
lambda[6]	0.974	0.155	0.754	0.878	0.932	1.043	1.370	1.001	6600
lambda[7]	1.041	0.176	0.814	0.926	0.991	1.121	1.490	1.002	1400
lambda[8]	0.871	0.118	0.625	0.803	0.880	0.932	1.116	1.001	7500

```
lambda[9]   0.925   0.183   0.589   0.831   0.911   0.994   1.382 1.001  7500
sigma2.y   69.895 114.422   0.169   8.769  32.050  83.250 366.210 1.005   940
```

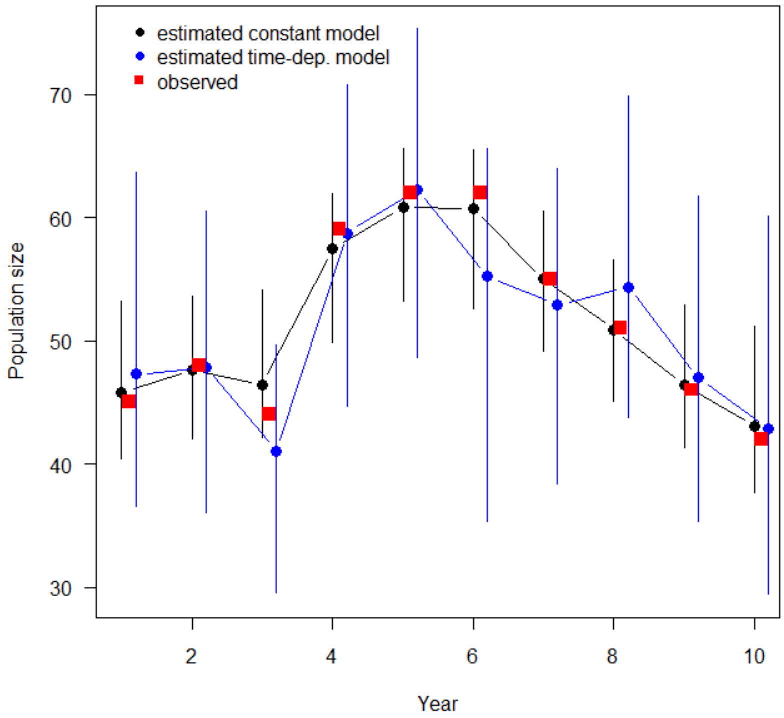
Here are the results of the model with constant parameters:

```
             mean     sd   2.5%     25%     50%     75%   97.5%  Rhat n.eff
mean.sjuv   0.257  0.018   0.223   0.245   0.257   0.269   0.294 1.001  7500
mean.sad    0.519  0.026   0.468   0.501   0.519   0.536   0.570 1.001  7500
mean.p      0.619  0.037   0.548   0.594   0.618   0.644   0.691 1.001  7500
mean.fec    3.829  0.115   3.603   3.752   3.826   3.907   4.055 1.003  1000
N1[1]      22.757 13.264   1.127  11.440  22.600  34.052  45.310 1.009   320
N1[2]      23.472  3.475  16.660  21.120  23.450  25.830  30.310 1.001  7500
N1[3]      22.205  3.590  15.660  19.690  22.070  24.482  29.840 1.002  2800
N1[4]      30.080  3.882  22.240  27.520  30.160  32.650  37.445 1.001  7500
N1[5]      29.896  3.909  22.245  27.280  29.905  32.470  37.630 1.001  3800
N1[6]      29.221  3.994  21.409  26.510  29.180  31.952  37.195 1.001  7500
N1[7]      25.621  3.744  18.589  23.010  25.570  28.140  32.996 1.001  7500
N1[8]      23.847  3.556  17.045  21.510  23.790  26.130  30.970 1.001  7500
N1[9]      21.700  3.414  15.170  19.340  21.670  23.920  28.580 1.001  7500
N1[10]     20.237  3.563  13.630  17.880  20.050  22.490  27.565 1.001  4900
Nad[1]     22.955 13.209   1.359  11.330  23.050  34.052  45.180 1.005   540
Nad[2]     24.250  3.268  18.000  22.000  24.000  26.000  31.000 1.001  7500
Nad[3]     24.195  3.284  18.000  22.000  24.000  26.000  31.000 1.001  3800
Nad[4]     27.347  3.376  21.000  25.000  27.000  30.000  34.000 1.001  7500
Nad[5]     30.923  3.641  24.000  29.000  31.000  33.000  38.000 1.001  6100
Nad[6]     31.424  3.751  24.000  29.000  31.000  34.000  39.000 1.001  7500
Nad[7]     29.327  3.565  22.000  27.000  29.000  32.000  36.000 1.001  7500
Nad[8]     27.028  3.389  20.000  25.000  27.000  29.000  34.000 1.001  7500
Nad[9]     24.699  3.215  18.000  23.000  25.000  27.000  31.000 1.001  7500
Nad[10]    22.831  3.230  17.000  21.000  23.000  25.000  29.000 1.002  1400
Ntot[1]    45.711  3.101  39.990  44.260  45.220  46.870  53.137 1.001  5100
Ntot[2]    47.722  2.622  42.260  46.460  47.830  48.792  53.565 1.001  7500
Ntot[3]    46.399  3.154  42.205  44.160  45.560  47.960  54.560 1.001  6500
Ntot[4]    57.426  2.976  49.905  56.140  58.140  59.130  62.145 1.001  7500
Ntot[5]    60.819  3.044  53.199  59.680  61.540  62.420  65.495 1.001  7500
Ntot[6]    60.645  3.081  52.779  59.400  61.390  62.340  65.480 1.001  7500
Ntot[7]    54.948  2.787  48.780  53.780  55.000  56.220  60.590 1.002  7500
Ntot[8]    50.875  2.662  45.015  49.720  50.950  52.030  56.495 1.002  6800
Ntot[9]    46.399  2.691  41.199  45.130  46.100  47.512  52.765 1.001  5300
Ntot[10]   43.068  3.240  37.540  41.480  42.400  44.310  51.241 1.001  3000
lambda[1]   1.047  0.071   0.893   1.008   1.055   1.082   1.192 1.001  7500
lambda[2]   0.974  0.073   0.874   0.921   0.957   1.012   1.154 1.001  6500
lambda[3]   1.243  0.102   1.011   1.178   1.266   1.327   1.381 1.001  7500
lambda[4]   1.061  0.060   0.942   1.032   1.055   1.089   1.194 1.001  7500
lambda[5]   0.999  0.054   0.887   0.972   0.999   1.022   1.119 1.001  7500
lambda[6]   0.908  0.055   0.814   0.877   0.896   0.933   1.048 1.001  7500
lambda[7]   0.928  0.057   0.812   0.900   0.927   0.953   1.057 1.001  7500
lambda[8]   0.914  0.060   0.802   0.882   0.906   0.942   1.057 1.001  7500
lambda[9]   0.930  0.068   0.804   0.893   0.920   0.960   1.096 1.001  7500
sigma2.y   14.580 28.382   0.016   1.550   6.087  16.480  79.205 1.010   740
```

To visualize the difference between the two models, we best produce a figure (this requires the results from the model with constant parameters).

```
lower <- upper <- lower.t <- upper.t <- numeric()
for (i in 1:10){
   lower[i] <- quantile(ipm$sims.list$Ntot[,i], 0.025)
   upper[i] <- quantile(ipm$sims.list$Ntot[,i], 0.975)
   lower.t[i] <- quantile(ipm.t$sims.list$Ntot[,i], 0.025)
   upper.t[i] <- quantile(ipm.t$sims.list$Ntot[,i], 0.975)
   }
```

```
plot(ipm$mean$Ntot, type = "b", ylim = c(min(c(lower, lower.t)),
max(c(upper, upper.t))), ylab = "Population size", xlab = "Year", las = 1,
cex = 1.2, pch = 16)
segments(1:10, lower, 1:10, upper)
points((1:10)+0.2, ipm.t$mean$Ntot, type = "b", cex = 1.2, pch = 19, col =
"blue")
segments((1:10)+0.2, lower.t, (1:10)+0.2, upper.t, col = "blue")
points((1:10)+0.1, y, type = "p", col = "red", pch = 15, cex = 1.5)
legend(x = 1, y = 77, legend = c("estimated constant model", "estimated
time-dep. model", "observed"), pch = c(16, 19, 15), col = c("black",
"blue", "red"), bty = "n")
```



As expected, the estimates under a time-dependent model are less precise than those under a constant model. Moreover, the fit of the constant model to the observed counts appears to be slightly better than that of the time-dependent model (This is shown more formally also by the smaller observation variance in the constant model). Given that the data generating parameters were constant this is not a very surprising result.

Exercise 4

Task: Use an integrated population model to study population dynamics of British lapwings (Besbeas et al., 2002; Brooks et al., 2004). The data consist of a national population index (1965-1998) and of mark-recoveries from individuals marked as hatchings (1963-1997). No data on productivity is available. Construct a model with two age classes for survival and where a) first breeding occurs at age 2 years and b) where it occurs at age 3 years. Further, c) make a model where survival is a function of the number of frost days. The data (population index, m-array, normalized number of frost days; data from Brooks et al. 2004) can be found on the book website (www.vogelwarte.ch/bpa).

Solution: The assumption about the age at which lapwings start to reproduce affects the way how the state model is written. When we assume that the age of first reproduction is with 2 years, it is enough to distinguish between 2 age classes. Yet, if we assume an age of 3 years for the first reproduction, we need to consider 3 age classes in the model. The remaining parts of the intergared model are quite straightforward, they pose no particular difficulty. We have chosen a model with random year effects on all demographic parameters. This has the advantage that the full length of the national population index could be used. Recall that we have the population index until 1998, but the recovery data only until 1997. Thus, in principle, we cannot use the data from 1998, because we do not have information about survival from 1997 to 1998. This can be overcome by considering year as a random effect. We can then generate survival probabilities from 1997 to 1998 from the estimated distributions.

Load data

```
# Lapwing data taken from Brooks et al. 2004 (Animal Biodiversity and
Conservation 27.1: 515-529)

# National population index (1965-1998)
y = c(NA,NA, 1092.23,1100.01, 1234.32, 1460.85, 1570.38, 1819.79,1391.27,1507.60,
1541.44,1631.21,1628.60,1609.33,1801.68,1809.08,1754.74,1779.48,1699.13,
1681.39,1610.46,1918.45,1717.07,1415.69, 1229.02,1082.02,1096.61,1045.84, 1137.03,
981.1, 647.67, 992.65, 968.62, 926.83, 952.96, 865.64)

# Mark-recoveries from individuals marked as hatchings (1963-1997) in m-
array format
dead.recov <- matrix(c(
13, 4, 1, 2, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1124,
0, 16, 4, 3, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1259,
0, 0, 11, 1, 1, 1, 0, 2, 1, 1, 1, 1, 2, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1082,
0, 0, 0, 10, 4, 2, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1595,
0, 0, 0, 0, 11, 1, 5, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1596,
0, 0, 0, 0, 0, 9, 5, 4, 0, 2, 2, 2, 1, 2, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 2091,
0, 0, 0, 0, 0, 0, 11, 9, 4, 3, 1, 1, 1, 3, 2, 2, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1964,
0, 0, 0, 0, 0, 0, 0, 8, 4, 2, 0, 0, 0, 1, 2, 3, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1942,
0, 0, 0, 0, 0, 0, 0, 0, 4, 1, 1, 2, 2, 1, 3, 3, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 2444,
0, 0, 0, 0, 0, 0, 0, 0, 0, 8, 2, 2, 2, 6, 1, 5, 2, 1, 3, 1, 1, 1, 2, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 3055,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 16, 1, 1, 1, 2, 3, 2, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 3412,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 13, 4, 4, 7, 3, 1, 1, 1, 1, 0, 0, 2, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 3907,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 11, 4, 0, 2, 1, 1, 2, 2, 0, 3, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 2538,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 11, 3, 5, 1, 3, 3, 2, 3, 0, 1, 0, 1, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 3270,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12, 5, 0, 5, 4, 2, 1, 2, 3, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, 3443,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 15, 5, 2, 2, 0, 5, 3, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 3132,
```

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 4, 6, 1, 3, 3, 2, 0, 1, 0, 0, 1,
0, 1, 0, 0, 0, 0, 0, 3275,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 13, 8, 1, 2, 4, 5, 3, 0, 1, 2,
0, 0, 1, 0, 0, 0, 0, 0, 3447,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 23, 2, 2, 3, 3, 3, 1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 3902,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10, 0, 6, 2, 0, 1, 1, 0,
0, 1, 0, 0, 0, 0, 0, 0, 2860,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 19, 7, 6, 4, 0, 0, 2,
0, 0, 0, 1, 2, 0, 0, 1, 4077,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12, 3, 2, 0, 0, 0,
0, 1, 0, 1, 0, 0, 0, 0, 4017,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 25, 2, 5, 2, 0,
2, 2, 2, 0, 0, 0, 0, 0, 4827,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 14, 4, 3, 4,
4, 2, 2, 1, 0, 2, 0, 1, 4732,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 14, 2, 1,
2, 2, 3, 0, 0, 3, 0, 0, 5000,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 18, 4,
4, 3, 0, 2, 1, 0, 2, 1, 4769,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10,
4, 2, 4, 2, 2, 3, 1, 1, 3603,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
12, 3, 3, 2, 1, 0, 2, 0, 4147,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
9, 4, 6, 1, 0, 1, 0, 4293,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 18, 3, 1, 2, 0, 1, 3455,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 6, 5, 2, 2, 1, 3673,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 12, 4, 6, 0, 3900,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 7, 5, 1, 3578,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 7, 0, 4481,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 5, 4334), ncol= 36, nrow=35, byrow = T)


# Normalised number of frost days from 1963-1997
f = c(0.1922,0.3082,0.3082,-0.9676,0.5401,0.3082,1.1995,0.1921,-0.8526,-1.0835,-
0.6196,-1.1995,-0.5037,-0.1557,0.0762,2.628,-0.3877,-0.968,1.9318,-0.6196,-
0.3877,1.700, 2.2797,0.6561,-0.8516,-1.0835,-1.0835,0.1922,0.1922,-0.1557,-0.5037,-
0.8516,0.8880,-0.0398,-1.1995)
```

Data analysis

a) Age at first breeding is 2 years

```
# Specify model in BUGS language
sink("ipm-lapwing1.bug")
cat("
model {
#-------------------------------------------------
#  Integrated population model
#  - Age structured model with 2 age classes:
#          1-year old and adult (at least 2 years old)
#  - Age at first breeding = 2 years
#  - Prebreeding census, female-based
#  - All vital rates assumed to be constant
#-------------------------------------------------

#-------------------------------------------------
# 1. Define the priors for the parameters
#-------------------------------------------------
# Observation error
```

```
tauy <- pow(sigma.y, -2)
sigma.y ~ dunif(0, 50)
sigma2.y <- pow(sigma.y, 2)

# Initial population sizes
N1[1] ~ dnorm(200, 0.0001)I(0,)      # 1-year
Nad[1] ~ dnorm(1000, 0.0001)I(0,)    # Adults

# Survival and recapture probabilities, as well as productivity
for (t in 1:(n.occasions+1)){
   logit(sj[t]) <- mu.sj + ep.sj[t]
   logit(sa[t]) <- mu.sa + ep.sa[t]
   logit(rj[t]) <- mu.rj + ep.rj[t]
   ra[t] <- rj[t]
   log(fec[t]) <- mu.fec + ep.fec[t]

   ep.sj[t] ~ dnorm(0, tau.sj)I(-10,10)
   ep.sa[t] ~ dnorm(0, tau.sa)I(-10,10)
   ep.rj[t] ~ dnorm(0, tau.rj)I(-10,10)
   ep.fec[t] ~ dnorm(0, tau.fec)I(-10,10)
   }

mean.sj ~ dunif(0, 1)
mu.sj <- log(mean.sj / (1-mean.sj))
mean.sa ~ dunif(0, 1)
mu.sa <- log(mean.sa / (1-mean.sa))
mean.rj ~ dunif(0, 1)
mu.rj <- log(mean.rj / (1-mean.rj))
mean.fec ~ dunif(0, 5)
mu.fec <- log(mean.fec)

sigma.sj ~ dunif(0, 10)
tau.sj <- pow(sigma.sj, -2)
sigma2.sj <- pow(sigma.sj, 2)
sigma.sa ~ dunif(0, 10)
tau.sa <- pow(sigma.sa, -2)
sigma2.sa <- pow(sigma.sa, 2)
sigma.rj ~ dunif(0, 10)
tau.rj <- pow(sigma.rj, -2)
sigma2.rj <- pow(sigma.rj, 2)
sigma.fec ~ dunif(0, 10)
tau.fec <- pow(sigma.fec, -2)
sigma2.fec <- pow(sigma.fec, 2)


#-------------------------------------------------
# 2. The likelihoods of the single data sets
#-------------------------------------------------
# 2.1. Likelihood for population count data (state-space model)
   # 3.1.1 System process
   for (t in 2:(n.occasions+1)){
      mean1[t] <- fec[t-1] / 2 * sj[t-1] * Nad[t-1]
      N1[t] ~ dpois(mean1[t])
      Nad[t] ~ dbin(sa[t-1], Ntot[t-1])
      }
   for (t in 1:(n.occasions+1)){
      Ntot[t] <- Nad[t] + N1[t]   # only breeding birds are counted
      }

   # 3.1.2 Observation process
   for (t in 3:(n.occasions+1)){
      y[t] ~ dnorm(Nad[t], tauy)
```

```
      }

# Define the multinomial likelihoods
for (t in 1:n.occasions){
   marr.j[t,1:(n.occasions+1)] ~ dmulti(pr.j[t,], rel.j[t])
   }
# Calculate the number of birds released each year
for (t in 1:n.occasions){
   rel.j[t] <- sum(marr.j[t,])
   }
# Define the cell probabilities of the juvenile m-array
# Main diagonal
for (t in 1:n.occasions){
   pr.j[t,t] <- (1-sj[t])*rj[t]
   # Further above main diagonal
   for (j in (t+2):n.occasions){
      pr.j[t,j] <- sj[t]*prod(sa[(t+1):(j-1)])*(1-sa[j])*ra[j]
      } #j
   # Below main diagonal
   for (j in 1:(t-1)){
      pr.j[t,j] <- 0
      } #j
   } #t
for (t in 1:(n.occasions-1)){
   # One above main diagonal
   pr.j[t,t+1] <- sj[t]*(1-sa[t+1])*ra[t+1]
   } #t
# Last column: probability of non-recovery
for (t in 1:n.occasions){
   pr.j[t,n.occasions+1] <- 1-sum(pr.j[t,1:n.occasions])
   } #t
}
",fill = TRUE)
sink()


# Bundle data
bugs.data <- list(marr.j = dead.recov, y = y, n.occasions =
dim(dead.recov)[2]-1)  # last year of census not used, because no
demographic data available (but, since we have

# Initial values
inits <- function(){list(mean.sj = runif(1, 0.4, 0.6), mean.sa = runif(1,
0.7, 0.9), mean.rj = runif(1, 0, 0.2), mean.fec = runif(1, 0, 2), sigma.sj
= runif(1, 0, 1), sigma.sa = runif(1, 0, 1), sigma.rj = runif(1, 0, 1),
sigma.fec = runif(1, 0, 1), N1 = rpois(36, 400), Nad = rpois(36, 1000),
sigma.y = runif(1, 0, 10))}


# Parameters monitored
parameters <- c("sj", "mean.sj", "sigma2.sj", "sa", "mean.sa", "sigma2.sa",
"rj", "mean.rj", "sigma2.rj", "fec", "mean.fec", "sigma2.fec", "sigma2.y",
"N1", "Nad", "Ntot")

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

# Call WinBUGS from R (BRT 362 min)
```

```
ipm.lapwing1 <- bugs(bugs.data, inits, parameters, "ipm-lapwing1.bug",
n.chains = nc, n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE,
bugs.directory = bugs.dir, working.directory = getwd())

save(ipm.lapwing1a, file="ipm.lapwing1a.Rdata")

# Inspect results
print(ipm.lapwing1, 3)
Inference for Bugs model at "ipm-lapwing1.bug", fit using WinBUGS,
 3 chains, each with 10000 iterations (first 5000 discarded), n.thin = 3
 n.sims = 5001 iterations saved
              mean    sd   2.5%    25%    50%    75%  97.5%  Rhat n.eff
sj[1]        0.583 0.065 0.435  0.546  0.593  0.625  0.693 1.002  2200
sj[2]        0.580 0.066 0.429  0.543  0.590  0.624  0.692 1.006   450
sj[3]        0.626 0.056 0.509  0.592  0.625  0.661  0.740 1.002  1500
sj[4]        0.618 0.057 0.499  0.585  0.618  0.651  0.732 1.006   470
sj[5]        0.611 0.058 0.487  0.578  0.614  0.647  0.721 1.003  1300
sj[6]        0.649 0.056 0.542  0.611  0.644  0.684  0.765 1.001  3100
sj[7]        0.671 0.058 0.576  0.626  0.664  0.711  0.793 1.001  5000
sj[8]        0.631 0.058 0.506  0.598  0.630  0.667  0.746 1.006   370
sj[9]        0.652 0.063 0.540  0.611  0.644  0.691  0.786 1.001  4400
sj[10]       0.672 0.060 0.573  0.628  0.666  0.711  0.798 1.002  5000
sj[11]       0.560 0.070 0.398  0.516  0.572  0.611  0.670 1.004   790
sj[12]       0.617 0.054 0.500  0.586  0.618  0.650  0.723 1.002  1300
sj[13]       0.617 0.057 0.497  0.584  0.617  0.651  0.729 1.004   690
sj[14]       0.650 0.055 0.552  0.613  0.644  0.686  0.768 1.002  5000
sj[15]       0.639 0.055 0.531  0.604  0.635  0.672  0.760 1.001  5000
sj[16]       0.630 0.055 0.518  0.598  0.629  0.663  0.741 1.001  5000
sj[17]       0.645 0.058 0.538  0.607  0.640  0.681  0.771 1.002  2500
sj[18]       0.645 0.053 0.547  0.610  0.640  0.679  0.758 1.001  5000
sj[19]       0.577 0.063 0.433  0.538  0.586  0.620  0.681 1.007   400
sj[20]       0.601 0.058 0.475  0.568  0.605  0.636  0.709 1.004   890
sj[21]       0.618 0.055 0.502  0.587  0.617  0.650  0.730 1.002  1400
sj[22]       0.578 0.070 0.417  0.539  0.590  0.625  0.696 1.001  4600
sj[23]       0.559 0.067 0.409  0.516  0.570  0.609  0.665 1.005   740
sj[24]       0.621 0.054 0.506  0.589  0.620  0.654  0.730 1.001  3600
sj[25]       0.571 0.066 0.422  0.528  0.581  0.618  0.680 1.004  1300
sj[26]       0.585 0.059 0.453  0.550  0.593  0.625  0.689 1.009   280
sj[27]       0.647 0.056 0.541  0.611  0.643  0.682  0.765 1.002  2300
sj[28]       0.594 0.065 0.444  0.559  0.602  0.634  0.713 1.010   340
sj[29]       0.595 0.065 0.446  0.560  0.603  0.635  0.715 1.001  2600
sj[30]       0.567 0.067 0.410  0.529  0.578  0.614  0.672 1.016   220
sj[31]       0.664 0.065 0.551  0.620  0.656  0.704  0.805 1.001  3100
sj[32]       0.627 0.059 0.501  0.594  0.626  0.662  0.749 1.002  1500
sj[33]       0.651 0.061 0.543  0.610  0.644  0.688  0.783 1.003   850
sj[34]       0.642 0.066 0.514  0.603  0.637  0.681  0.782 1.003  1700
sj[35]       0.638 0.067 0.503  0.598  0.634  0.676  0.784 1.002  5000
sj[36]       0.616 0.072 0.458  0.578  0.619  0.657  0.759 1.002  4900
mean.sj      0.620 0.021 0.580  0.606  0.620  0.634  0.660 1.004  1400
sigma2.sj    0.095 0.084 0.000  0.031  0.076  0.138  0.302 1.106   130
sa[1]        0.818 0.050 0.710  0.787  0.822  0.854  0.905 1.010   270
sa[2]        0.806 0.043 0.715  0.778  0.807  0.835  0.883 1.003   920
sa[3]        0.814 0.039 0.727  0.790  0.817  0.841  0.884 1.004  5000
sa[4]        0.818 0.042 0.726  0.793  0.822  0.848  0.893 1.013   170
sa[5]        0.837 0.039 0.755  0.813  0.839  0.863  0.905 1.019   970
sa[6]        0.835 0.036 0.762  0.813  0.837  0.860  0.901 1.007   870
sa[7]        0.823 0.038 0.746  0.800  0.825  0.849  0.892 1.004   550
sa[8]        0.729 0.038 0.654  0.703  0.731  0.755  0.797 1.036    63
sa[9]        0.816 0.036 0.742  0.792  0.818  0.840  0.881 1.004   660
sa[10]       0.832 0.034 0.763  0.810  0.833  0.854  0.894 1.005   470
sa[11]       0.861 0.030 0.802  0.840  0.862  0.884  0.916 1.013   160
sa[12]       0.843 0.031 0.780  0.822  0.843  0.864  0.899 1.007   390
sa[13]       0.840 0.030 0.777  0.820  0.841  0.860  0.895 1.010   210
sa[14]       0.808 0.036 0.730  0.787  0.811  0.832  0.872 1.006  1100
sa[15]       0.813 0.032 0.746  0.791  0.815  0.835  0.873 1.022   140
sa[16]       0.787 0.036 0.711  0.765  0.790  0.811  0.848 1.051    49
sa[17]       0.817 0.033 0.749  0.795  0.818  0.840  0.880 1.008   270
sa[18]       0.810 0.033 0.740  0.789  0.812  0.832  0.870 1.010   360
sa[19]       0.797 0.033 0.729  0.775  0.798  0.820  0.855 1.032    72
sa[20]       0.827 0.033 0.757  0.806  0.829  0.850  0.888 1.003  1300
sa[21]       0.840 0.031 0.775  0.819  0.841  0.862  0.896 1.004   700
sa[22]       0.739 0.038 0.663  0.713  0.740  0.766  0.809 1.004  3400
sa[23]       0.719 0.041 0.633  0.693  0.721  0.748  0.792 1.050    46
sa[24]       0.754 0.042 0.668  0.727  0.756  0.783  0.829 1.005   560
sa[25]       0.773 0.039 0.692  0.749  0.775  0.800  0.844 1.005   560
sa[26]       0.829 0.036 0.755  0.806  0.831  0.854  0.895 1.034    76
```

sa[27]	0.815	0.038	0.729	0.791	0.817	0.841	0.882	1.002	1600
sa[28]	0.816	0.038	0.737	0.792	0.817	0.842	0.882	1.033	81
sa[29]	0.763	0.041	0.677	0.737	0.766	0.791	0.838	1.018	140
sa[30]	0.726	0.053	0.614	0.691	0.729	0.764	0.817	1.009	250
sa[31]	0.803	0.039	0.720	0.779	0.806	0.830	0.875	1.004	1200
sa[32]	0.830	0.037	0.751	0.807	0.832	0.855	0.897	1.004	710
sa[33]	0.810	0.038	0.730	0.785	0.812	0.836	0.876	1.005	740
sa[34]	0.788	0.041	0.703	0.762	0.790	0.817	0.862	1.024	98
sa[35]	0.818	0.040	0.734	0.792	0.821	0.845	0.894	1.009	600
sa[36]	0.805	0.054	0.684	0.774	0.809	0.841	0.898	1.004	1000
mean.sa	0.809	0.013	0.781	0.800	0.809	0.818	0.834	1.006	870
sigma2.sa	0.115	0.053	0.037	0.077	0.107	0.143	0.243	1.033	75
rj[1]	0.020	0.006	0.011	0.016	0.019	0.023	0.034	1.001	5000
rj[2]	0.023	0.006	0.014	0.019	0.023	0.027	0.037	1.004	610
rj[3]	0.019	0.005	0.011	0.016	0.019	0.022	0.031	1.002	1300
rj[4]	0.015	0.004	0.009	0.012	0.015	0.017	0.024	1.003	720
rj[5]	0.015	0.004	0.009	0.012	0.014	0.017	0.023	1.001	5000
rj[6]	0.011	0.003	0.007	0.009	0.011	0.013	0.017	1.002	1700
rj[7]	0.016	0.004	0.010	0.013	0.015	0.018	0.025	1.001	5000
rj[8]	0.012	0.003	0.008	0.010	0.012	0.014	0.018	1.011	190
rj[9]	0.009	0.002	0.005	0.007	0.009	0.010	0.014	1.002	1100
rj[10]	0.009	0.002	0.006	0.008	0.009	0.011	0.014	1.001	5000
rj[11]	0.010	0.002	0.006	0.009	0.010	0.011	0.015	1.003	720
rj[12]	0.009	0.002	0.006	0.008	0.009	0.010	0.014	1.001	2700
rj[13]	0.011	0.002	0.007	0.009	0.011	0.012	0.016	1.002	1200
rj[14]	0.012	0.002	0.008	0.010	0.012	0.013	0.017	1.003	930
rj[15]	0.011	0.002	0.007	0.009	0.011	0.012	0.016	1.004	710
rj[16]	0.014	0.003	0.010	0.012	0.014	0.016	0.020	1.010	210
rj[17]	0.008	0.002	0.005	0.007	0.008	0.009	0.012	1.006	370
rj[18]	0.010	0.002	0.007	0.009	0.010	0.011	0.015	1.003	920
rj[19]	0.014	0.003	0.010	0.012	0.014	0.016	0.020	1.013	180
rj[20]	0.008	0.002	0.005	0.007	0.008	0.009	0.012	1.002	1200
rj[21]	0.011	0.002	0.008	0.010	0.011	0.013	0.016	1.003	1100
rj[22]	0.010	0.002	0.007	0.009	0.010	0.011	0.014	1.001	4800
rj[23]	0.011	0.002	0.007	0.009	0.010	0.012	0.014	1.018	120
rj[24]	0.007	0.001	0.005	0.006	0.007	0.008	0.010	1.002	1500
rj[25]	0.006	0.001	0.004	0.006	0.006	0.007	0.009	1.001	5000
rj[26]	0.008	0.002	0.005	0.007	0.008	0.009	0.011	1.010	220
rj[27]	0.007	0.002	0.005	0.006	0.007	0.008	0.011	1.001	4900
rj[28]	0.008	0.002	0.005	0.007	0.008	0.009	0.012	1.010	220
rj[29]	0.006	0.001	0.004	0.005	0.006	0.007	0.009	1.005	510
rj[30]	0.008	0.002	0.005	0.007	0.008	0.009	0.012	1.002	1300
rj[31]	0.008	0.002	0.005	0.006	0.008	0.009	0.012	1.006	390
rj[32]	0.008	0.002	0.005	0.007	0.008	0.009	0.012	1.005	460
rj[33]	0.007	0.002	0.005	0.006	0.007	0.008	0.011	1.004	1900
rj[34]	0.007	0.002	0.004	0.006	0.007	0.008	0.010	1.005	520
rj[35]	0.004	0.001	0.003	0.004	0.004	0.005	0.007	1.003	760
rj[36]	0.011	0.005	0.004	0.007	0.010	0.013	0.024	1.001	4600
mean.rj	0.010	0.001	0.008	0.009	0.010	0.010	0.012	1.001	5000
sigma2.rj	0.199	0.072	0.091	0.150	0.188	0.238	0.368	1.006	410
fec[1]	1.011	0.448	0.324	0.677	0.960	1.282	2.028	1.028	83
fec[2]	0.883	0.314	0.404	0.654	0.842	1.060	1.651	1.016	270
fec[3]	1.200	0.335	0.658	0.957	1.172	1.398	1.968	1.023	94
fec[4]	1.495	0.360	0.933	1.247	1.443	1.692	2.331	1.016	1500
fec[5]	1.097	0.298	0.584	0.900	1.069	1.271	1.755	1.005	5000
fec[6]	1.281	0.288	0.780	1.083	1.250	1.458	1.913	1.003	730
fec[7]	0.364	0.163	0.109	0.244	0.342	0.458	0.735	1.044	52
fec[8]	0.759	0.184	0.443	0.629	0.747	0.872	1.165	1.007	320
fec[9]	0.759	0.220	0.369	0.610	0.744	0.888	1.256	1.030	220
fec[10]	0.703	0.188	0.375	0.568	0.694	0.821	1.105	1.015	160
fec[11]	0.715	0.232	0.342	0.547	0.688	0.856	1.233	1.010	270
fec[12]	0.596	0.173	0.291	0.479	0.580	0.703	0.971	1.025	120
fec[13]	1.223	0.280	0.774	1.036	1.190	1.374	1.872	1.009	1100
fec[14]	0.827	0.220	0.454	0.672	0.807	0.966	1.311	1.029	74
fec[15]	0.733	0.215	0.359	0.590	0.712	0.860	1.215	1.069	38
fec[16]	0.747	0.193	0.395	0.616	0.733	0.862	1.164	1.030	90
fec[17]	0.591	0.183	0.289	0.457	0.574	0.706	0.989	1.010	260
fec[18]	0.721	0.197	0.384	0.580	0.699	0.844	1.164	1.026	87
fec[19]	0.589	0.186	0.278	0.456	0.568	0.703	0.990	1.001	2800
fec[20]	1.293	0.255	0.855	1.117	1.272	1.446	1.851	1.004	650
fec[21]	0.799	0.264	0.325	0.614	0.782	0.967	1.348	1.014	790
fec[22]	0.510	0.217	0.172	0.358	0.483	0.629	1.031	1.064	43
fec[23]	0.471	0.197	0.165	0.329	0.438	0.591	0.920	1.014	200
fec[24]	0.455	0.175	0.150	0.332	0.443	0.565	0.836	1.018	280
fec[25]	0.655	0.224	0.308	0.500	0.622	0.780	1.176	1.036	61
fec[26]	0.610	0.222	0.246	0.450	0.587	0.745	1.119	1.004	600
fec[27]	0.870	0.256	0.430	0.694	0.844	1.024	1.420	1.033	67
fec[28]	0.473	0.207	0.148	0.323	0.443	0.596	0.960	1.048	50

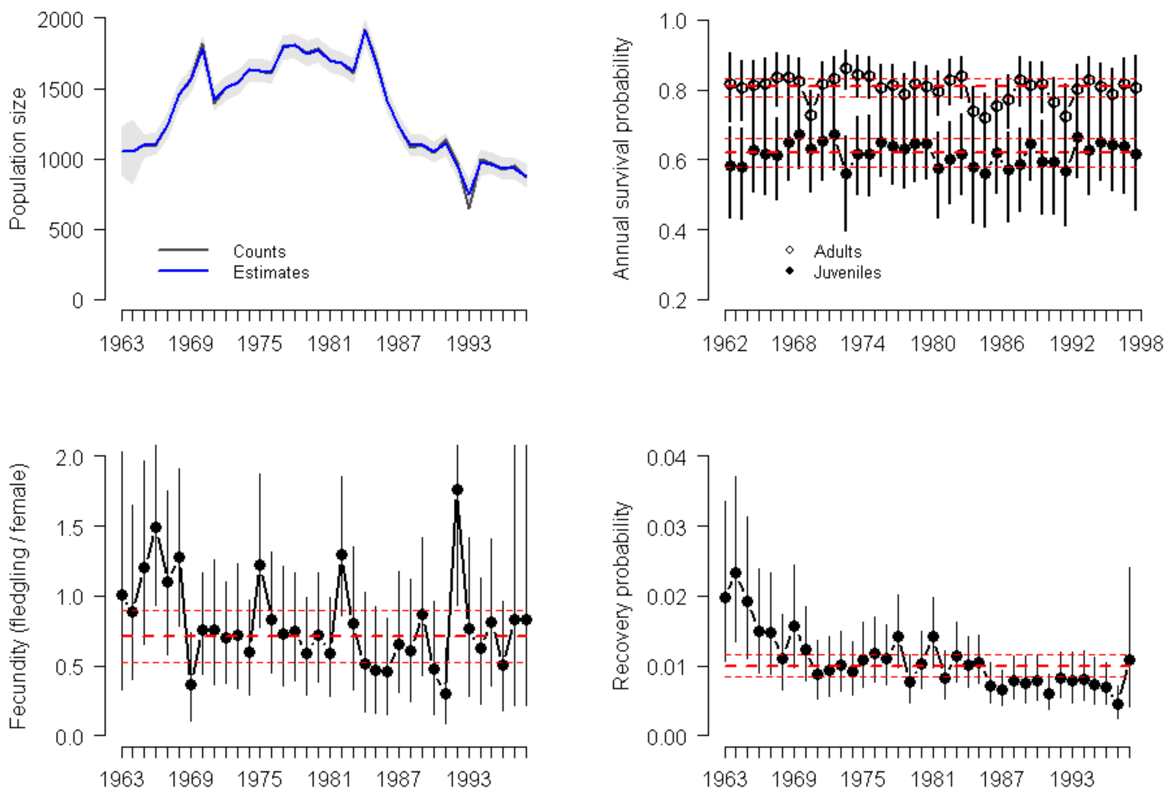
fec[29]	0.299	0.144	0.081	0.194	0.279	0.384	0.627	1.022	110
fec[30]	1.760	0.456	0.930	1.460	1.722	2.035	2.783	1.032	100
fec[31]	0.763	0.296	0.283	0.545	0.728	0.945	1.419	1.002	1200
fec[32]	0.628	0.233	0.227	0.466	0.610	0.772	1.128	1.020	220
fec[33]	0.809	0.267	0.357	0.617	0.785	0.981	1.408	1.029	74
fec[34]	0.501	0.206	0.178	0.350	0.480	0.626	0.963	1.009	660
fec[35]	0.829	0.537	0.220	0.489	0.709	1.015	2.216	1.004	850
fec[36]	0.835	0.546	0.213	0.490	0.708	1.027	2.203	1.006	760
mean.fec	0.709	0.096	0.523	0.646	0.707	0.772	0.898	1.037	76
sigma2.fec	0.318	0.159	0.108	0.210	0.286	0.394	0.701	1.042	54
sigma2.y	1797.485	596.478	268.300	1475.000	1963.000	2270.000	2479.000	1.028	2200
N1[1]	235.877	93.431	50.890	172.100	236.900	298.900	414.800	1.012	250
N1[2]	304.892	122.988	99.160	215.200	296.100	384.400	571.100	1.027	87
N1[3]	264.232	81.875	129.000	205.800	255.400	314.900	448.900	1.012	390
N1[4]	409.416	106.691	232.700	332.200	399.800	477.500	640.000	1.024	89
N1[5]	508.041	108.736	327.400	432.600	496.000	571.400	749.100	1.018	2200
N1[6]	412.882	103.502	221.000	345.100	406.600	474.300	628.400	1.005	5000
N1[7]	603.732	122.863	384.700	522.600	593.600	675.800	864.700	1.004	540
N1[8]	187.348	84.127	55.880	125.300	175.800	237.200	378.700	1.040	57
N1[9]	424.060	97.195	253.800	356.000	417.900	485.000	633.800	1.009	240
N1[10]	349.064	94.038	179.000	285.800	344.800	406.100	560.600	1.033	210
N1[11]	352.928	88.263	194.100	289.900	347.500	410.400	535.100	1.016	160
N1[12]	303.504	90.756	146.500	238.100	296.900	360.400	497.800	1.012	250
N1[13]	297.508	84.098	146.300	241.000	290.500	350.000	477.000	1.024	130
N1[14]	610.837	124.490	400.200	530.000	598.000	676.900	886.000	1.007	5000
N1[15]	430.948	107.578	241.500	355.300	426.200	498.900	660.200	1.030	71
N1[16]	418.551	120.051	213.000	337.700	406.900	488.000	688.300	1.068	39
N1[17]	422.903	105.590	229.300	352.700	414.700	488.200	645.700	1.031	86
N1[18]	331.088	99.313	168.600	258.900	321.500	392.700	551.300	1.013	180
N1[19]	410.165	107.405	219.900	335.700	401.200	480.400	643.000	1.031	72
N1[20]	286.459	89.445	133.700	221.000	279.200	342.600	480.800	1.003	1000
N1[21]	648.106	105.940	455.900	575.000	642.900	717.900	867.200	1.006	420
N1[22]	398.162	127.378	161.700	310.500	396.000	482.000	653.800	1.014	600
N1[23]	276.761	113.836	94.900	195.500	262.100	344.300	537.000	1.065	42
N1[24]	220.100	89.923	74.380	153.400	207.200	273.600	420.500	1.014	200
N1[25]	196.625	74.397	65.260	143.700	190.500	244.500	355.100	1.020	250
N1[26]	225.840	71.859	106.200	174.900	219.000	269.100	392.200	1.040	55
N1[27]	194.017	69.828	76.250	143.600	187.800	236.900	346.200	1.004	710
N1[28]	306.369	86.241	154.100	247.000	298.500	361.000	484.100	1.034	64
N1[29]	144.821	64.121	43.790	98.100	137.200	182.200	292.300	1.049	50
N1[30]	93.673	44.695	24.770	60.600	87.320	121.600	194.000	1.023	100
N1[31]	467.928	99.214	274.000	403.700	468.100	531.300	669.600	1.037	100
N1[32]	187.232	71.287	69.540	134.400	180.700	232.400	351.800	1.002	1300
N1[33]	189.846	68.168	70.360	140.900	185.400	232.900	337.800	1.016	300
N1[34]	251.978	80.678	114.800	195.400	244.200	303.700	427.000	1.035	62
N1[35]	147.572	61.333	49.970	102.100	141.500	184.300	288.600	1.009	530
N1[36]	246.273	164.700	63.000	142.000	209.000	304.000	651.000	1.004	760
Nad[1]	1054.386	88.360	882.400	992.700	1053.000	1116.000	1229.000	1.008	280
Nad[2]	1056.342	115.050	825.000	978.000	1057.000	1139.000	1273.000	1.042	69
Nad[3]	1094.580	40.539	1011.000	1069.000	1095.000	1121.000	1173.000	1.003	5000
Nad[4]	1104.652	38.830	1029.000	1079.000	1104.000	1130.000	1183.000	1.016	130
Nad[5]	1237.238	43.522	1148.000	1211.000	1238.000	1265.000	1324.000	1.004	680
Nad[6]	1458.815	41.545	1375.000	1431.000	1458.000	1486.000	1541.000	1.003	950
Nad[7]	1562.247	42.806	1476.000	1535.000	1564.000	1590.000	1646.000	1.009	260
Nad[8]	1781.417	43.048	1694.000	1752.000	1783.000	1811.000	1862.000	1.004	550
Nad[9]	1425.056	40.194	1349.000	1397.000	1423.000	1451.000	1508.000	1.003	980
Nad[10]	1506.081	42.382	1422.000	1479.000	1506.000	1535.000	1589.000	1.011	280
Nad[11]	1541.215	40.087	1463.000	1515.000	1540.000	1567.000	1624.000	1.004	690
Nad[12]	1630.256	41.060	1549.000	1603.000	1631.000	1657.000	1712.000	1.007	880
Nad[13]	1627.770	39.108	1548.000	1602.000	1627.000	1653.000	1707.000	1.008	330
Nad[14]	1614.815	39.705	1538.000	1589.000	1614.000	1641.000	1694.000	1.001	4300
Nad[15]	1795.950	41.923	1712.000	1769.000	1797.000	1823.000	1876.000	1.013	200
Nad[16]	1808.524	41.044	1721.000	1783.000	1809.000	1835.000	1888.000	1.008	280
Nad[17]	1749.702	42.177	1665.000	1723.000	1750.000	1777.000	1835.000	1.008	260
Nad[18]	1773.482	41.479	1690.000	1747.000	1774.000	1799.000	1858.000	1.006	380
Nad[19]	1701.429	40.623	1620.000	1676.000	1702.000	1726.000	1785.000	1.004	620
Nad[20]	1679.673	39.500	1602.000	1654.000	1680.000	1706.000	1758.000	1.003	880
Nad[21]	1624.174	40.459	1546.000	1598.000	1622.000	1651.000	1707.000	1.002	1400
Nad[22]	1907.601	40.890	1821.000	1882.000	1910.000	1935.000	1986.000	1.001	4200
Nad[23]	1700.829	40.631	1618.000	1675.000	1702.000	1727.000	1779.000	1.003	830
Nad[24]	1415.252	37.753	1339.000	1391.000	1415.000	1440.000	1490.000	1.001	5000
Nad[25]	1227.600	38.461	1150.000	1202.000	1227.000	1252.000	1306.000	1.005	430
Nad[26]	1097.505	37.665	1024.000	1074.000	1096.000	1121.000	1176.000	1.006	420
Nad[27]	1095.665	37.121	1021.000	1072.000	1095.000	1120.000	1168.000	1.002	1300
Nad[28]	1048.818	36.748	974.000	1026.000	1049.000	1074.000	1120.000	1.004	710
Nad[29]	1103.938	40.055	1022.000	1077.000	1106.000	1132.000	1179.000	1.010	230
Nad[30]	948.587	36.830	875.000	924.000	950.000	974.000	1018.000	1.021	100
Nad[31]	745.092	45.740	655.000	714.000	746.000	776.000	835.000	1.009	320

```
Nad[32]     974.518  43.128  886.000  947.000  977.000 1002.000 1058.000 1.028    80
Nad[33]     963.562  37.632  891.000  939.000  963.000  988.000 1039.000 1.016   200
Nad[34]     931.966  36.094  861.000  909.000  931.000  955.000 1007.000 1.005   520
Nad[35]     931.953  38.462  855.000  907.000  933.000  957.000 1007.000 1.009   240
Nad[36]     879.971  37.729  806.000  856.000  879.000  904.000  958.000 1.002  1100
Ntot[1]    1290.262 121.137 1050.000 1205.000 1293.000 1377.000 1519.000 1.026   100
Ntot[2]    1361.238  86.270 1202.000 1300.000 1357.000 1418.000 1545.000 1.004  2600
Ntot[3]    1358.812  77.758 1221.000 1304.000 1351.000 1406.000 1530.000 1.004   540
Ntot[4]    1514.078  99.440 1351.000 1443.000 1504.000 1575.000 1741.000 1.014   150
Ntot[5]    1745.278  99.753 1585.000 1674.000 1736.000 1802.000 1976.000 1.020  1500
Ntot[6]    1871.698  97.494 1695.000 1806.000 1867.000 1929.000 2076.000 1.005  1400
Ntot[7]    2165.986 117.312 1966.000 2085.000 2155.000 2237.000 2425.000 1.006   360
Ntot[8]    1968.761  91.396 1814.000 1903.000 1962.000 2025.000 2163.000 1.045    50
Ntot[9]    1849.113  95.493 1686.000 1784.000 1844.000 1907.000 2062.000 1.009   260
Ntot[10]   1855.159  87.385 1698.000 1795.000 1849.000 1910.000 2052.000 1.002  2300
Ntot[11]   1894.148  83.595 1746.000 1835.000 1889.000 1949.000 2069.000 1.016   150
Ntot[12]   1933.761  84.274 1788.000 1873.000 1927.000 1989.000 2113.000 1.003   720
Ntot[13]   1925.284  80.731 1785.000 1868.000 1918.000 1977.000 2098.000 1.011   220
Ntot[14]   2225.662 117.783 2025.000 2149.000 2213.000 2290.000 2483.000 1.008  5000
Ntot[15]   2226.898 100.698 2053.000 2156.000 2221.000 2290.000 2443.000 1.028    91
Ntot[16]   2227.076 114.962 2037.000 2146.000 2214.000 2296.000 2474.000 1.067    38
Ntot[17]   2172.610 100.304 1989.000 2104.000 2167.000 2235.000 2391.000 1.015   150
Ntot[18]   2104.566  94.287 1944.000 2036.000 2096.000 2163.000 2316.000 1.013   220
Ntot[19]   2111.601 102.450 1939.000 2037.000 2102.000 2175.000 2334.000 1.033    69
Ntot[20]   1966.133  85.892 1819.000 1904.000 1959.000 2020.000 2152.000 1.001  3200
Ntot[21]   2272.282  98.114 2097.000 2204.000 2267.000 2333.000 2484.000 1.004   610
Ntot[22]   2305.759 128.451 2072.000 2214.000 2302.000 2391.000 2568.000 1.007  1100
Ntot[23]   1977.594 115.435 1791.000 1897.000 1963.000 2045.000 2251.000 1.058    43
Ntot[24]   1635.350  92.333 1480.000 1570.000 1625.000 1690.000 1840.000 1.009   250
Ntot[25]   1424.216  77.015 1287.000 1369.000 1419.000 1473.000 1588.000 1.007   310
Ntot[26]   1323.338  68.802 1203.000 1276.000 1317.000 1365.000 1482.000 1.036    75
Ntot[27]   1289.679  68.995 1172.000 1240.000 1284.000 1332.000 1443.000 1.005   500
Ntot[28]   1355.188  80.782 1212.000 1298.000 1350.000 1405.000 1519.000 1.044    56
Ntot[29]   1248.756  70.810 1124.000 1200.000 1242.000 1292.000 1407.000 1.032    67
Ntot[30]   1042.258  59.729  936.400  999.900 1038.000 1081.000 1166.000 1.034    64
Ntot[31]   1213.021  83.452 1061.000 1158.000 1205.000 1262.000 1397.000 1.013   190
Ntot[32]   1161.749  66.601 1047.000 1114.000 1156.000 1203.000 1307.000 1.014   190
Ntot[33]   1153.412  63.995 1042.000 1109.000 1147.000 1194.000 1292.000 1.005  5000
Ntot[34]   1183.951  76.322 1051.000 1129.000 1179.000 1232.000 1350.000 1.032    68
Ntot[35]   1079.522  60.892  975.700 1036.000 1075.000 1117.000 1215.000 1.004  3500
Ntot[36]   1126.244 169.685  915.000 1020.000 1093.000 1191.000 1540.000 1.005  1000
deviance   1551.810  25.588 1491.000 1541.000 1555.000 1568.000 1590.000 1.007  5000
```

```
# Produce graph similar to the one in Fig. 11-7 to visualize results
par(mfrow = c(2, 2), cex.axis = 1.2, cex.lab = 1.2, mar = c(5, 6, 1.5, 2),
las = 1)
lower <- upper <- numeric()
year <- 1963:1998
nyears <- length(year)
for (i in 1:nyears){
   lower[i] <- quantile(ipm.lapwing1$sims.list$Nad[,i], 0.025)
   upper[i] <- quantile(ipm.lapwing1$sims.list$Nad[,i], 0.975)}
m1 <- min(c(ipm.lapwing1$mean$Nad, y, lower), na.rm = T)
m2 <- max(c(ipm.lapwing1$mean$Nad, y, upper), na.rm = T)
plot(0, 0, ylim = c(0, m2), xlim = c(1, nyears), ylab = "Population size",
xlab = " ", col = "black", type = "l",  axes = F, frame = F)
axis(2)
axis(1, at = 1:nyears, labels = year)
polygon(x = c(1:nyears, nyears:1), y = c(lower, upper[nyears:1]), col =
"grey90", border = "grey90")
points(y, type = "l", col = "grey30", lwd = 2)
points(ipm.lapwing1$mean$Nad, type = "l", col = "blue", lwd = 2)
legend(x = 2, y = 500, legend = c("Counts", "Estimates"), lty = c(1, 1),lwd
= c(2, 2), col = c("grey30", "blue"), bty = "n", cex = 1)

lower <- upper <- numeric()
T <- nyears
for (t in 1:T){
   lower[t] <- quantile(ipm.lapwing1$sims.list$sj[,t], 0.025)
   upper[t] <- quantile(ipm.lapwing1$sims.list$sj[,t], 0.975)}
```

```
par(mgp=c(3.8,1,0))
plot(y = ipm.lapwing1$mean$sj, x = (1:T)+0.5, xlim= c(1, T), type = "b",
pch = 16, ylim = c(0.2, 1), ylab = "Annual survival probability", xlab =
"", axes = F, cex = 1.5, frame = F, lwd = 2)
axis(2)
axis(1, at = 1:(T+1), labels = 1962:1998)
segments((1:T)+0.5, lower, (1:T)+0.5, upper, lwd = 2)
segments(1, ipm.lapwing1$mean$mean.sj, T+1, ipm.lapwing1$mean$mean.sj, lty
= 2, lwd = 2, col = "red")
segments(1, quantile(ipm.lapwing1$sims.list$mean.sj, 0.025), T+1,
quantile(ipm.lapwing1$sims.list$mean.sj, 0.025), lty = 2, col = "red")
segments(1, quantile(ipm.lapwing1$sims.list$mean.sj, 0.975), T+1,
quantile(ipm.lapwing1$sims.list$mean.sj, 0.975), lty = 2, col = "red")
for (t in 1:T){
   lower[t] <- quantile(ipm.lapwing1$sims.list$sa[,t], 0.025)
   upper[t] <- quantile(ipm.lapwing1$sims.list$sa[,t], 0.975)}
points(y=ipm.lapwing1$mean$sa, x = (1:T)+0.5, type = "b", pch = 1, cex =
1.5, lwd = 2)
segments((1:T)+0.5, lower, (1:T)+0.5, upper, lwd = 2)
segments(1, ipm.lapwing1$mean$mean.sa, T+1, ipm.lapwing1$mean$mean.sa, lty
= 2, lwd = 2, col = "red")
segments(1, quantile(ipm.lapwing1$sims.list$mean.sa, 0.025), T+1,
quantile(ipm.lapwing1$sims.list$mean.sa, 0.025), lty = 2, col = "red")
segments(1, quantile(ipm.lapwing1$sims.list$mean.sa, 0.975), T+1,
quantile(ipm.lapwing1$sims.list$mean.sa, 0.975), lty = 2, col = "red")
legend(x = 4.5, y = 0.4, legend = c("Adults", "Juveniles"), pch = c(1, 16),
bty = "n")

lower <- upper <- numeric()
T <- nyears
for (t in 1:T){
   lower[t] <- quantile(ipm.lapwing1$sims.list$fec[,t], 0.025)
   upper[t] <- quantile(ipm.lapwing1$sims.list$fec[,t], 0.975)}
plot(y=ipm.lapwing1$mean$fec, x = (1:T), type = "b", pch = 16, ylim = c(0,
2), ylab = "Fecundity (fledgling / female)", xlab = "", axes = F, cex =
1.5, frame = F, lwd = 2)
axis(2)
axis(1, at = 1:T, labels = 1963:1998)
segments((1:T), lower, (1:T), upper)
segments(1, ipm.lapwing1$mean$mean.fec, T, ipm.lapwing1$mean$mean.fec, lty
= 2, lwd = 2, col = "red")
segments(1, quantile(ipm.lapwing1$sims.list$mean.fec, 0.025), T,
quantile(ipm.lapwing1$sims.list$mean.fec, 0.025), lty = 2, col = "red")
segments(1, quantile(ipm.lapwing1$sims.list$mean.fec, 0.975), T,
quantile(ipm.lapwing1$sims.list$mean.fec, 0.975), lty = 2, col = "red")

lower <- upper <- numeric()
T <- nyears
for (t in 1:T){
   lower[t] <- quantile(ipm.lapwing1$sims.list$rj[,t], 0.025)
   upper[t] <- quantile(ipm.lapwing1$sims.list$rj[,t], 0.975)}
plot(y=ipm.lapwing1$mean$rj, x = (1:T), type = "b", pch = 16, ylim = c(0,
0.04), ylab = "Recovery probability", xlab = "", axes = F, cex = 1.5, frame
= F, lwd = 2)
axis(2)
axis(1, at = 1:T, labels = 1963:1998)
segments((1:T), lower, (1:T), upper)
segments(1, ipm.lapwing1$mean$mean.rj, T, ipm.lapwing1$mean$mean.rj, lty =
2, lwd = 2, col = "red")
segments(1, quantile(ipm.lapwing1$sims.list$mean.rj, 0.025), T,
quantile(ipm.lapwing1$sims.list$mean.rj, 0.025), lty = 2, col = "red")
```

```
segments(1, quantile(ipm.lapwing1$sims.list$mean.rj, 0.975), T,
quantile(ipm.lapwing1$sims.list$mean.rj, 0.975), lty = 2, col = "red")
```



The recovery probability declines over time, and thus a better (i.e., more parsimonious) model may be one that constrains recovery probability to decline linearly over time, with some annual variability around the logit-linear trend. You can build such a model easily by replacing

```
logit(rj[t]) <- mu.rj + ep.rj[t]
```

with

```
logit(rj[t]) <- mu.rj + beta*t + ep.rj[t].
```

You then need to give a prior for beta.

b) Age at first breeding is 3 years

```
# Specify model in BUGS language
sink("ipm-lapwing2.bug")
cat("
model {
#--------------------------------------------------
#  Integrated population model
#  - Age structured model with 2 age classes:
#          1-year old and adult (at least 2 years old)
#  - Age at first breeding = 3 years
```

```
#  - Prebreeding census, female-based
#  - All vital rates assumed to be constant
#-------------------------------------------------

#-------------------------------------------------
# 1. Define the priors for the parameters
#-------------------------------------------------
# Observation error
tauy <- pow(sigma.y, -2)
sigma.y ~ dunif(0, 50)
sigma2.y <- pow(sigma.y, 2)

# Initial population sizes
N1[1] ~ dnorm(200, 0.001)I(0,)      # 1-year
N2[1] ~ dnorm(200, 0.001)I(0,)      # 1-year
Nad[1] ~ dnorm(1000, 0.001)I(0,)    # Adults

# Survival and recapture probabilities, as well as productivity
for (t in 1:(n.occasions+1)){
   logit(sj[t]) <- mu.sj + ep.sj[t]
   logit(sa[t]) <- mu.sa + ep.sa[t]
   logit(rj[t]) <- mu.rj + ep.rj[t]
   ra[t] <- rj[t]
   log(fec[t]) <- mu.fec + ep.fec[t]

   ep.sj[t] ~ dnorm(0, tau.sj)I(-10,10)
   ep.sa[t] ~ dnorm(0, tau.sa)I(-10,10)
   ep.rj[t] ~ dnorm(0, tau.rj)I(-10,10)
   ep.fec[t] ~ dnorm(0, tau.fec)I(-10,10)
   }

mean.sj ~ dunif(0, 1)
mu.sj <- log(mean.sj / (1-mean.sj))
mean.sa ~ dunif(0, 1)
mu.sa <- log(mean.sa / (1-mean.sa))
mean.rj ~ dunif(0, 1)
mu.rj <- log(mean.rj / (1-mean.rj))
mean.fec ~ dunif(0, 5)
mu.fec <- log(mean.fec)

sigma.sj ~ dunif(0, 10)
tau.sj <- pow(sigma.sj, -2)
sigma2.sj <- pow(sigma.sj, 2)
sigma.sa ~ dunif(0, 10)
tau.sa <- pow(sigma.sa, -2)
sigma2.sa <- pow(sigma.sa, 2)
sigma.rj ~ dunif(0, 10)
tau.rj <- pow(sigma.rj, -2)
sigma2.rj <- pow(sigma.rj, 2)
sigma.fec ~ dunif(0, 10)
tau.fec <- pow(sigma.fec, -2)
sigma2.fec <- pow(sigma.fec, 2)

#-------------------------------------------------
# 2. The likelihoods of the single data sets
#-------------------------------------------------
# 2.1. Likelihood for population count data (state-space model)
   # 3.1.1 System process
   for (t in 2:(n.occasions+1)){
      mean1[t] <- fec[t-1] / 2 * sj[t-1] * Nad[t-1]
      N1[t] ~ dpois(mean1[t])
      N2[t] ~ dbin(sa[t-1], N1[t-1])
```

```
      Nad[t] ~ dbin(sa[t-1], Np[t-1])
      }
   for (t in 1:(n.occasions+1)){
      Np[t] <- Nad[t] + N2[t]
      Ntot[t] <- N1[t] + N2[t] + Nad[t]
      }

   # 3.1.2 Observation process
   for (t in 3:(n.occasions+1)){
      y[t] ~ dnorm(Nad[t], tauy)
      }

# Define the multinomial likelihoods
for (t in 1:n.occasions){
   marr.j[t,1:(n.occasions+1)] ~ dmulti(pr.j[t,], rel.j[t])
   }
# Calculate the number of birds released each year
for (t in 1:n.occasions){
   rel.j[t] <- sum(marr.j[t,])
   }
# Define the cell probabilities of the juvenile m-array
# Main diagonal
for (t in 1:n.occasions){
   pr.j[t,t] <- (1-sj[t])*rj[t]
   # Further above main diagonal
   for (j in (t+2):n.occasions){
      pr.j[t,j] <- sj[t]*prod(sa[(t+1):(j-1)])*(1-sa[j])*ra[j]
      } #j
   # Below main diagonal
   for (j in 1:(t-1)){
      pr.j[t,j] <- 0
      } #j
   } #t
for (t in 1:(n.occasions-1)){
   # One above main diagonal
   pr.j[t,t+1] <- sj[t]*(1-sa[t+1])*ra[t+1]
   } #t
# Last column: probability of non-recovery
for (t in 1:n.occasions){
   pr.j[t,n.occasions+1] <- 1-sum(pr.j[t,1:n.occasions])
   } #t
}
",fill = TRUE)
sink()


# Bundle data
bugs.data <- list(marr.j = dead.recov, y = y, n.occasions =
dim(dead.recov)[2]-1)

# Initial values
inits <- function(){list(mean.sj = runif(1, 0.4, 0.6), mean.sa = runif(1,
0.7, 0.9), mean.rj = runif(1, 0, 0.2), mean.fec = runif(1, 0, 2), sigma.sj
= runif(1, 0, 1), sigma.sa = runif(1, 0, 1), sigma.rj = runif(1, 0, 1),
sigma.fec = runif(1, 0, 1), N1 = rpois(36, 200), N2 = rpois(36, 100), Nad =
rpois(36, 1000), sigma.y = runif(1, 0, 10))}

# Parameters monitored
parameters <- c("sj", "mean.sj", "sigma2.sj", "sa", "mean.sa", "sigma2.sa",
"rj", "mean.rj", "sigma2.rj", "fec", "mean.fec", "sigma2.fec", "sigma2.y",
"N1", "N2", "Nad", "Ntot")
```

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

# Call WinBUGS from R (BRT 420 min)
ipm.lapwing2 <- bugs(bugs.data, inits, parameters, "ipm-lapwing2.bug",
n.chains = nc, n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE,
bugs.directory = bugs.dir, working.directory = getwd())

print(ipm.lapwing2, 3)
Inference for Bugs model at "ipm-lapwing2.bug", fit using WinBUGS,
 3 chains, each with 10000 iterations (first 5000 discarded), n.thin = 3
 n.sims = 5001 iterations saved
              mean    sd   2.5%    25%    50%    75%  97.5%  Rhat n.eff
sj[1]        0.586 0.062  0.448  0.550  0.596  0.628  0.694 1.016   200
sj[2]        0.595 0.061  0.458  0.562  0.603  0.633  0.704 1.019   200
sj[3]        0.629 0.056  0.512  0.597  0.627  0.663  0.746 1.030   110
sj[4]        0.611 0.058  0.488  0.578  0.614  0.645  0.726 1.014   510
sj[5]        0.619 0.056  0.504  0.588  0.620  0.652  0.732 1.007   950
sj[6]        0.631 0.054  0.522  0.599  0.629  0.664  0.744 1.008   580
sj[7]        0.679 0.058  0.585  0.634  0.674  0.719  0.797 1.045    50
sj[8]        0.628 0.056  0.512  0.596  0.627  0.663  0.739 1.005  1800
sj[9]        0.654 0.061  0.548  0.614  0.646  0.692  0.790 1.013   240
sj[10]       0.670 0.057  0.574  0.628  0.664  0.709  0.790 1.019   110
sj[11]       0.560 0.069  0.407  0.514  0.570  0.612  0.666 1.042    60
sj[12]       0.627 0.053  0.516  0.596  0.627  0.660  0.733 1.005  5000
sj[13]       0.614 0.055  0.499  0.582  0.614  0.647  0.722 1.013   850
sj[14]       0.649 0.057  0.543  0.612  0.644  0.684  0.772 1.015   150
sj[15]       0.634 0.054  0.522  0.601  0.631  0.669  0.742 1.006  1100
sj[16]       0.626 0.054  0.516  0.596  0.625  0.659  0.737 1.012   430
sj[17]       0.646 0.056  0.542  0.610  0.641  0.682  0.767 1.011   250
sj[18]       0.643 0.053  0.544  0.609  0.638  0.676  0.758 1.006   480
sj[19]       0.591 0.059  0.458  0.556  0.599  0.630  0.694 1.001  3200
sj[20]       0.587 0.063  0.440  0.551  0.597  0.628  0.693 1.007  1300
sj[21]       0.617 0.055  0.502  0.586  0.618  0.650  0.726 1.005  4800
sj[22]       0.576 0.072  0.407  0.534  0.590  0.625  0.691 1.058    47
sj[23]       0.560 0.066  0.406  0.521  0.571  0.609  0.660 1.036    69
sj[24]       0.625 0.052  0.519  0.594  0.624  0.657  0.731 1.006  3800
sj[25]       0.573 0.065  0.426  0.533  0.582  0.619  0.681 1.012   180
sj[26]       0.594 0.058  0.462  0.561  0.602  0.630  0.695 1.008  1000
sj[27]       0.632 0.055  0.522  0.600  0.629  0.665  0.747 1.016   180
sj[28]       0.591 0.067  0.434  0.556  0.600  0.633  0.707 1.012   880
sj[29]       0.626 0.056  0.510  0.594  0.626  0.660  0.736 1.008  1300
sj[30]       0.553 0.074  0.373  0.509  0.567  0.608  0.662 1.040    75
sj[31]       0.659 0.061  0.552  0.617  0.651  0.698  0.792 1.011   220
sj[32]       0.632 0.057  0.512  0.599  0.631  0.665  0.747 1.017   210
sj[33]       0.641 0.062  0.514  0.606  0.638  0.679  0.768 1.010   370
sj[34]       0.647 0.066  0.518  0.606  0.641  0.688  0.791 1.016   210
sj[35]       0.641 0.064  0.512  0.603  0.637  0.679  0.777 1.012   240
sj[36]       0.618 0.071  0.464  0.582  0.620  0.656  0.758 1.006  5000
mean.sj      0.621 0.020  0.582  0.607  0.621  0.635  0.661 1.016   150
sigma2.sj    0.091 0.079  0.000  0.030  0.072  0.134  0.288 1.147    25
sa[1]        0.862 0.031  0.797  0.842  0.864  0.884  0.917 1.009   250
sa[2]        0.845 0.030  0.784  0.825  0.845  0.866  0.904 1.008   280
sa[3]        0.813 0.041  0.722  0.789  0.816  0.841  0.884 1.015   140
sa[4]        0.825 0.038  0.747  0.801  0.826  0.852  0.896 1.040    60
sa[5]        0.839 0.038  0.757  0.815  0.840  0.866  0.907 1.071    36
sa[6]        0.841 0.038  0.759  0.817  0.845  0.869  0.906 1.021   140
sa[7]        0.829 0.036  0.755  0.806  0.830  0.855  0.893 1.027    99
sa[8]        0.728 0.035  0.659  0.704  0.729  0.752  0.794 1.006  3200
sa[9]        0.821 0.033  0.752  0.799  0.822  0.844  0.881 1.018   240
sa[10]       0.831 0.031  0.763  0.811  0.834  0.853  0.886 1.016   150
sa[11]       0.857 0.029  0.799  0.838  0.857  0.877  0.910 1.023    97
sa[12]       0.835 0.030  0.771  0.815  0.836  0.856  0.889 1.016   180
sa[13]       0.841 0.030  0.779  0.822  0.842  0.861  0.900 1.009   450
sa[14]       0.804 0.037  0.724  0.780  0.807  0.832  0.867 1.034    72
sa[15]       0.803 0.032  0.734  0.782  0.805  0.826  0.861 1.006   400
sa[16]       0.781 0.037  0.694  0.759  0.785  0.808  0.843 1.101    46
sa[17]       0.816 0.033  0.748  0.794  0.816  0.839  0.880 1.023   110
sa[18]       0.818 0.032  0.751  0.798  0.820  0.841  0.875 1.022    95
sa[19]       0.797 0.036  0.724  0.772  0.800  0.822  0.860 1.039    64
sa[20]       0.827 0.031  0.756  0.808  0.830  0.849  0.881 1.060    48
sa[21]       0.840 0.033  0.775  0.818  0.841  0.864  0.903 1.029    84
```

sa[22]	0.735	0.042	0.648	0.709	0.738	0.765	0.810 1.089	27
sa[23]	0.720	0.042	0.629	0.692	0.723	0.750	0.793 1.095	31
sa[24]	0.751	0.042	0.665	0.722	0.753	0.782	0.827 1.005	990
sa[25]	0.775	0.041	0.683	0.751	0.779	0.804	0.846 1.045	68
sa[26]	0.829	0.039	0.738	0.807	0.833	0.856	0.893 1.018	250
sa[27]	0.804	0.035	0.729	0.783	0.807	0.828	0.869 1.043	73
sa[28]	0.814	0.037	0.736	0.791	0.816	0.839	0.882 1.013	240
sa[29]	0.757	0.042	0.665	0.731	0.760	0.785	0.832 1.031	77
sa[30]	0.731	0.049	0.628	0.699	0.735	0.767	0.818 1.048	52
sa[31]	0.797	0.042	0.704	0.771	0.801	0.826	0.871 1.006	390
sa[32]	0.829	0.036	0.755	0.805	0.831	0.856	0.893 1.054	43
sa[33]	0.810	0.041	0.712	0.787	0.814	0.837	0.880 1.090	37
sa[34]	0.783	0.041	0.694	0.757	0.786	0.812	0.856 1.025	99
sa[35]	0.818	0.040	0.733	0.793	0.821	0.847	0.888 1.027	81
sa[36]	0.804	0.056	0.675	0.771	0.810	0.843	0.896 1.006	1900
mean.sa	0.809	0.012	0.786	0.802	0.809	0.817	0.832 1.018	130
sigma2.sa	0.123	0.055	0.041	0.083	0.114	0.151	0.254 1.056	44
rj[1]	0.020	0.006	0.011	0.016	0.019	0.023	0.034 1.003	820
rj[2]	0.025	0.006	0.015	0.020	0.024	0.028	0.039 1.002	2400
rj[3]	0.019	0.005	0.011	0.016	0.019	0.022	0.031 1.006	380
rj[4]	0.015	0.004	0.009	0.012	0.015	0.017	0.024 1.002	1100
rj[5]	0.015	0.004	0.009	0.012	0.014	0.017	0.024 1.007	330
rj[6]	0.011	0.003	0.007	0.009	0.011	0.012	0.017 1.006	390
rj[7]	0.016	0.004	0.010	0.013	0.016	0.018	0.025 1.010	230
rj[8]	0.012	0.003	0.008	0.010	0.012	0.014	0.018 1.001	5000
rj[9]	0.009	0.002	0.005	0.007	0.009	0.010	0.014 1.001	5000
rj[10]	0.009	0.002	0.006	0.008	0.009	0.011	0.014 1.009	240
rj[11]	0.010	0.002	0.006	0.008	0.010	0.011	0.015 1.002	2100
rj[12]	0.009	0.002	0.006	0.008	0.009	0.010	0.014 1.001	2700
rj[13]	0.011	0.002	0.007	0.009	0.011	0.012	0.016 1.001	5000
rj[14]	0.012	0.002	0.008	0.010	0.011	0.013	0.017 1.009	250
rj[15]	0.011	0.002	0.007	0.009	0.010	0.012	0.015 1.001	4500
rj[16]	0.014	0.003	0.010	0.012	0.014	0.016	0.020 1.024	130
rj[17]	0.008	0.002	0.005	0.007	0.008	0.009	0.012 1.006	370
rj[18]	0.011	0.002	0.007	0.009	0.010	0.012	0.015 1.010	210
rj[19]	0.015	0.003	0.010	0.013	0.014	0.016	0.020 1.004	610
rj[20]	0.008	0.002	0.005	0.007	0.008	0.009	0.012 1.004	600
rj[21]	0.011	0.002	0.008	0.010	0.011	0.013	0.017 1.004	570
rj[22]	0.010	0.002	0.007	0.009	0.010	0.011	0.014 1.057	40
rj[23]	0.011	0.002	0.007	0.009	0.011	0.012	0.015 1.025	85
rj[24]	0.007	0.002	0.005	0.006	0.007	0.008	0.011 1.006	460
rj[25]	0.007	0.001	0.004	0.006	0.006	0.007	0.009 1.004	710
rj[26]	0.008	0.002	0.005	0.007	0.008	0.009	0.012 1.004	720
rj[27]	0.007	0.002	0.004	0.006	0.007	0.008	0.011 1.016	140
rj[28]	0.008	0.002	0.005	0.007	0.008	0.009	0.012 1.003	1200
rj[29]	0.006	0.001	0.004	0.005	0.006	0.007	0.009 1.006	360
rj[30]	0.008	0.002	0.006	0.007	0.008	0.009	0.012 1.030	73
rj[31]	0.008	0.002	0.005	0.006	0.007	0.009	0.012 1.004	550
rj[32]	0.008	0.002	0.005	0.007	0.008	0.009	0.012 1.014	150
rj[33]	0.007	0.002	0.005	0.006	0.007	0.008	0.011 1.008	290
rj[34]	0.007	0.002	0.004	0.006	0.007	0.008	0.010 1.003	940
rj[35]	0.005	0.001	0.003	0.004	0.004	0.005	0.007 1.010	210
rj[36]	0.011	0.005	0.004	0.007	0.010	0.013	0.024 1.003	1000
mean.rj	0.010	0.001	0.008	0.009	0.010	0.010	0.012 1.003	920
sigma2.rj	0.204	0.072	0.099	0.152	0.193	0.242	0.375 1.003	980
fec[1]	1.214	0.421	0.547	0.911	1.151	1.451	2.206 1.078	31
fec[2]	1.553	0.416	0.882	1.253	1.506	1.799	2.470 1.050	47
fec[3]	1.782	0.432	1.073	1.476	1.735	2.038	2.782 1.069	34
fec[4]	1.480	0.416	0.813	1.168	1.427	1.749	2.378 1.021	110
fec[5]	1.802	0.423	1.082	1.492	1.770	2.068	2.744 1.028	79
fec[6]	0.523	0.216	0.185	0.359	0.501	0.659	1.008 1.016	210
fec[7]	1.048	0.252	0.586	0.871	1.045	1.219	1.549 1.042	110
fec[8]	0.759	0.191	0.433	0.626	0.744	0.875	1.188 1.025	240
fec[9]	0.962	0.238	0.516	0.802	0.948	1.117	1.454 1.086	39
fec[10]	0.784	0.214	0.433	0.635	0.761	0.906	1.251 1.054	55
fec[11]	0.803	0.242	0.337	0.644	0.788	0.940	1.347 1.053	74
fec[12]	1.454	0.333	0.937	1.197	1.409	1.680	2.182 1.038	62
fec[13]	1.159	0.283	0.654	0.959	1.140	1.330	1.772 1.010	850
fec[14]	1.050	0.306	0.578	0.833	1.005	1.221	1.708 1.053	69
fec[15]	0.976	0.240	0.566	0.802	0.964	1.134	1.487 1.038	62
fec[16]	0.657	0.207	0.327	0.503	0.634	0.795	1.102 1.049	48
fec[17]	0.886	0.258	0.459	0.691	0.860	1.064	1.437 1.054	42
fec[18]	0.640	0.188	0.358	0.505	0.606	0.743	1.079 1.100	29
fec[19]	1.589	0.351	0.956	1.360	1.559	1.803	2.371 1.022	120
fec[20]	0.990	0.360	0.366	0.745	0.946	1.193	1.767 1.090	28
fec[21]	0.750	0.321	0.277	0.516	0.701	0.933	1.525 1.096	26
fec[22]	0.593	0.227	0.247	0.418	0.565	0.740	1.086 1.013	730
fec[23]	0.538	0.218	0.206	0.380	0.510	0.656	1.091 1.061	45

fec[24]	0.639	0.235	0.277	0.472	0.611	0.767	1.189	1.033	130
fec[25]	0.782	0.236	0.392	0.615	0.762	0.917	1.334	1.016	480
fec[26]	1.234	0.335	0.612	0.998	1.213	1.455	1.929	1.032	120
fec[27]	0.497	0.233	0.150	0.323	0.464	0.628	1.085	1.044	65
fec[28]	0.396	0.200	0.109	0.257	0.360	0.502	0.868	1.044	58
fec[29]	1.920	0.473	1.118	1.587	1.878	2.196	3.035	1.012	270
fec[30]	0.907	0.353	0.374	0.653	0.858	1.096	1.728	1.047	47
fec[31]	0.901	0.379	0.356	0.626	0.840	1.104	1.874	1.190	15
fec[32]	1.086	0.338	0.568	0.841	1.042	1.276	1.873	1.014	200
fec[33]	0.614	0.253	0.211	0.417	0.594	0.776	1.155	1.091	27
fec[34]	1.019	0.633	0.265	0.609	0.882	1.253	2.577	1.003	790
fec[35]	1.035	0.701	0.262	0.604	0.887	1.268	2.627	1.003	970
fec[36]	1.015	0.628	0.273	0.604	0.879	1.261	2.621	1.005	510
mean.fec	0.877	0.110	0.658	0.807	0.876	0.947	1.103	1.071	35
sigma2.fec	0.322	0.154	0.106	0.214	0.292	0.398	0.697	1.065	51
sigma2.y	1819.500	545.665	473.200	1509.000	1948.000	2255.000	2477.000	1.204	77
N1[1]	216.623	29.412	158.200	197.000	216.900	236.700	273.100	1.006	400
N1[2]	358.682	117.942	168.000	274.000	342.000	425.000	629.000	1.101	25
N1[3]	489.394	118.907	290.000	403.000	478.000	559.000	761.000	1.059	39
N1[4]	592.645	130.051	377.000	504.000	578.000	671.000	887.000	1.150	18
N1[5]	499.468	135.035	288.000	393.000	478.000	594.000	784.000	1.028	78
N1[6]	689.016	144.895	440.000	577.000	684.000	790.000	985.000	1.037	60
N1[7]	236.516	97.519	83.000	161.000	229.000	299.000	454.000	1.013	360
N1[8]	555.557	126.831	323.000	468.000	552.000	640.000	809.000	1.074	43
N1[9]	421.572	104.096	238.000	349.000	413.000	485.000	648.000	1.022	200
N1[10]	446.477	104.903	242.000	378.000	441.000	517.000	651.000	1.084	41
N1[11]	390.617	97.426	222.000	322.000	384.000	451.000	592.000	1.049	61
N1[12]	340.916	99.365	131.000	280.000	335.000	398.000	557.000	1.072	66
N1[13]	739.208	155.834	498.000	620.000	713.000	847.000	1075.000	1.047	55
N1[14]	578.089	134.072	330.000	479.000	568.000	668.000	849.000	1.017	370
N1[15]	545.481	151.974	305.000	440.000	520.000	621.000	924.000	1.081	45
N1[16]	551.855	131.530	329.000	456.000	544.000	639.000	828.000	1.042	55
N1[17]	368.907	112.236	187.000	285.000	356.000	447.000	605.000	1.066	38
N1[18]	499.936	144.153	262.000	386.000	484.000	606.000	789.000	1.038	59
N1[19]	362.609	102.924	208.000	288.000	343.000	423.000	600.000	1.098	29
N1[20]	791.201	147.375	487.000	696.000	786.000	889.000	1090.000	1.041	92
N1[21]	483.060	172.523	180.000	362.000	464.000	590.000	864.000	1.091	28
N1[22]	373.921	164.330	135.000	253.000	348.000	472.000	755.000	1.093	27
N1[23]	321.430	119.587	127.000	228.000	310.000	405.000	568.000	1.014	900
N1[24]	252.673	106.028	93.000	179.000	238.000	308.000	537.000	1.055	55
N1[25]	279.933	103.293	118.000	207.000	267.000	334.000	528.000	1.035	130
N1[26]	272.621	79.607	138.000	216.000	266.000	319.000	467.000	1.029	230
N1[27]	398.930	103.895	193.000	327.000	396.000	473.000	602.000	1.040	96
N1[28]	167.510	80.187	48.000	108.000	156.000	211.000	366.000	1.041	82
N1[29]	119.044	61.092	31.000	77.000	107.000	151.000	265.000	1.039	65
N1[30]	668.577	153.356	403.000	557.000	655.000	771.000	986.000	1.013	290
N1[31]	234.237	90.676	94.000	170.000	221.000	282.000	447.000	1.062	37
N1[32]	218.066	87.807	88.000	152.000	210.000	266.000	448.000	1.214	14
N1[33]	333.699	97.409	178.000	264.000	320.000	390.000	546.000	1.011	340
N1[34]	187.383	76.642	65.000	127.000	182.000	239.000	351.000	1.080	30
N1[35]	305.121	191.177	75.000	179.000	262.000	378.000	783.000	1.004	730
N1[36]	309.439	213.426	75.000	176.000	263.000	381.000	805.000	1.002	1500
N2[1]	216.534	30.076	158.000	196.400	216.900	236.200	275.900	1.002	2000
N2[2]	187.063	25.620	136.000	170.000	187.000	204.000	237.000	1.008	310
N2[3]	302.069	96.321	145.000	233.000	289.000	358.000	524.000	1.101	25
N2[4]	397.292	93.857	235.000	329.000	390.000	453.000	605.000	1.077	31
N2[5]	488.112	101.619	314.000	418.000	478.000	547.000	719.000	1.131	21
N2[6]	418.314	109.979	239.000	332.000	403.000	493.000	648.000	1.039	62
N2[7]	578.514	114.899	375.000	493.000	574.000	658.000	820.000	1.042	53
N2[8]	195.655	81.706	67.000	133.000	189.000	247.000	378.000	1.016	420
N2[9]	403.201	89.210	242.000	345.000	400.000	460.000	591.000	1.083	42
N2[10]	345.021	82.472	198.000	287.000	340.000	396.000	526.000	1.021	220
N2[11]	370.544	86.211	202.000	316.000	365.000	428.000	546.000	1.081	44
N2[12]	334.122	82.912	192.000	275.000	328.000	387.000	501.000	1.044	63
N2[13]	283.715	81.379	113.000	232.000	281.000	332.000	462.000	1.073	62
N2[14]	621.309	129.958	422.000	521.000	600.000	709.000	898.000	1.053	52
N2[15]	464.692	108.791	271.000	384.000	457.000	538.000	688.000	1.016	390
N2[16]	437.916	123.407	245.000	354.000	416.000	498.000	750.000	1.088	42
N2[17]	430.349	100.300	252.000	358.000	427.000	498.000	632.000	1.050	49
N2[18]	300.838	92.185	152.000	232.000	289.000	364.000	497.000	1.064	38
N2[19]	407.691	114.008	217.000	317.000	396.000	492.000	631.000	1.033	67
N2[20]	287.198	77.204	169.000	231.000	274.000	334.000	460.000	1.087	33
N2[21]	652.653	114.122	411.000	580.000	652.000	733.000	870.000	1.026	160
N2[22]	406.167	145.422	147.000	306.000	389.000	494.000	712.000	1.102	26
N2[23]	272.014	115.027	103.000	188.000	253.000	338.000	530.000	1.075	32
N2[24]	230.116	83.906	89.000	166.000	223.000	290.000	402.000	1.020	410
N2[25]	188.064	75.237	71.000	136.000	178.000	229.000	389.000	1.050	56
N2[26]	215.921	77.830	92.000	162.000	208.000	256.000	399.000	1.035	130

N2[27]	225.253	64.182	113.000	181.000	221.000	263.000	376.000	1.023	310
N2[28]	320.495	80.912	161.000	264.000	321.000	378.000	475.000	1.034	130
N2[29]	136.032	65.322	38.000	88.000	126.000	171.000	294.000	1.040	88
N2[30]	88.462	43.956	23.000	58.000	81.000	111.000	194.000	1.041	60
N2[31]	486.025	98.607	302.000	416.000	481.000	555.000	684.000	1.010	2700
N2[32]	185.971	70.026	74.000	136.000	177.000	224.000	346.000	1.059	39
N2[33]	180.503	72.232	74.000	126.000	174.000	222.000	362.000	1.200	15
N2[34]	268.786	73.489	146.000	217.000	260.000	313.000	422.000	1.004	1500
N2[35]	145.634	58.088	52.000	100.000	142.000	185.000	270.000	1.077	31
N2[36]	249.604	157.765	63.000	146.000	215.000	312.000	659.000	1.004	830
Nad[1]	1017.948	30.811	959.700	996.900	1018.000	1038.000	1079.000	1.001	4700
Nad[2]	1066.756	45.206	974.000	1037.000	1069.000	1099.000	1152.000	1.004	1400
Nad[3]	1062.400	38.726	982.000	1038.000	1065.000	1089.000	1133.000	1.031	88
Nad[4]	1107.620	40.667	1028.000	1081.000	1106.000	1134.000	1191.000	1.074	33
Nad[5]	1240.784	41.636	1160.000	1214.000	1239.000	1267.000	1326.000	1.012	170
Nad[6]	1449.662	42.039	1372.000	1422.000	1449.000	1476.000	1538.000	1.064	38
Nad[7]	1569.721	40.350	1487.000	1544.000	1571.000	1596.000	1648.000	1.034	69
Nad[8]	1779.030	43.066	1691.000	1751.000	1781.000	1809.000	1860.000	1.003	790
Nad[9]	1427.840	42.782	1347.000	1399.000	1426.000	1455.000	1517.000	1.029	75
Nad[10]	1501.714	38.758	1426.000	1476.000	1502.000	1527.000	1579.000	1.020	110
Nad[11]	1533.608	40.183	1449.000	1508.000	1535.000	1560.000	1613.000	1.026	86
Nad[12]	1629.262	40.239	1549.000	1603.000	1630.000	1655.000	1710.000	1.005	520
Nad[13]	1636.129	38.799	1563.000	1610.000	1635.000	1662.000	1714.000	1.024	90
Nad[14]	1613.252	41.378	1530.000	1586.000	1614.000	1641.000	1692.000	1.011	410
Nad[15]	1794.267	42.174	1713.000	1766.000	1794.000	1822.000	1880.000	1.012	180
Nad[16]	1811.126	42.093	1729.000	1783.000	1811.000	1839.000	1895.000	1.008	290
Nad[17]	1753.508	39.374	1673.000	1728.000	1754.000	1780.000	1830.000	1.017	130
Nad[18]	1779.915	41.585	1695.000	1753.000	1781.000	1807.000	1863.000	1.019	130
Nad[19]	1700.624	38.706	1624.000	1676.000	1700.000	1725.000	1779.000	1.012	190
Nad[20]	1676.571	39.939	1595.000	1651.000	1677.000	1702.000	1756.000	1.023	93
Nad[21]	1621.733	40.155	1545.000	1595.000	1620.000	1647.000	1705.000	1.005	950
Nad[22]	1910.109	42.637	1825.000	1882.000	1911.000	1938.000	1995.000	1.008	360
Nad[23]	1700.037	39.567	1618.000	1674.000	1702.000	1726.000	1775.000	1.005	650
Nad[24]	1413.212	38.064	1339.000	1388.000	1413.000	1438.000	1489.000	1.001	3400
Nad[25]	1230.148	37.955	1158.000	1205.000	1230.000	1255.000	1306.000	1.008	290
Nad[26]	1095.000	37.383	1018.000	1072.000	1096.000	1120.000	1167.000	1.002	5000
Nad[27]	1084.866	36.885	1012.000	1061.000	1085.000	1109.000	1159.000	1.002	2700
Nad[28]	1051.450	37.753	977.000	1026.000	1051.000	1076.000	1127.000	1.012	220
Nad[29]	1116.001	39.080	1037.000	1091.000	1117.000	1142.000	1190.000	1.001	5000
Nad[30]	943.176	38.624	868.000	918.000	944.000	970.000	1018.000	1.003	4200
Nad[31]	743.500	43.646	657.000	715.000	743.000	772.000	833.000	1.019	220
Nad[32]	978.631	39.036	901.000	953.000	979.000	1005.000	1054.000	1.008	270
Nad[33]	964.639	37.549	894.000	940.000	964.000	987.000	1043.000	1.039	64
Nad[34]	925.673	36.057	855.000	903.000	926.000	949.000	999.000	1.023	92
Nad[35]	933.893	35.190	864.000	910.000	934.000	957.000	1003.000	1.004	790
Nad[36]	880.447	37.856	808.000	856.000	879.000	905.000	958.000	1.019	110
Ntot[1]	1451.097	46.364	1363.000	1419.000	1452.000	1483.000	1544.000	1.003	1000
Ntot[2]	1612.501	118.674	1414.000	1532.000	1596.000	1680.000	1898.000	1.072	33
Ntot[3]	1853.862	142.169	1627.000	1748.000	1837.000	1947.000	2171.000	1.004	540
Ntot[4]	2097.558	158.049	1828.000	1987.000	2083.000	2196.000	2425.000	1.165	17
Ntot[5]	2228.365	161.687	1935.000	2107.000	2225.000	2344.000	2553.000	1.046	54
Ntot[6]	2556.993	190.759	2238.000	2405.000	2547.000	2679.000	2972.000	1.033	140
Ntot[7]	2384.751	140.186	2138.000	2285.000	2371.000	2478.000	2683.000	1.032	70
Ntot[8]	2530.242	150.797	2237.000	2425.000	2532.000	2635.000	2813.000	1.032	110
Ntot[9]	2252.613	139.630	2003.000	2153.000	2241.000	2340.000	2560.000	1.034	99
Ntot[10]	2293.211	125.173	2064.000	2206.000	2293.000	2375.000	2542.000	1.028	82
Ntot[11]	2294.770	117.011	2087.000	2213.000	2292.000	2372.000	2527.000	1.054	53
Ntot[12]	2304.300	131.170	2039.000	2221.000	2296.000	2388.000	2581.000	1.014	210
Ntot[13]	2659.051	166.068	2366.000	2539.000	2648.000	2774.000	3010.000	1.032	75
Ntot[14]	2812.649	174.199	2513.000	2685.000	2808.000	2918.000	3190.000	1.037	69
Ntot[15]	2804.441	170.530	2502.000	2683.000	2791.000	2911.000	3172.000	1.044	72
Ntot[16]	2800.897	180.569	2502.000	2671.000	2779.000	2911.000	3213.000	1.048	64
Ntot[17]	2552.765	138.149	2292.000	2457.000	2550.000	2646.000	2834.000	1.045	64
Ntot[18]	2580.689	176.969	2278.000	2452.000	2569.000	2701.000	2973.000	1.070	39
Ntot[19]	2470.924	168.481	2196.000	2349.000	2453.000	2573.000	2853.000	1.096	29
Ntot[20]	2754.971	175.034	2447.000	2633.000	2731.000	2863.000	3151.000	1.096	33
Ntot[21]	2757.447	189.983	2406.000	2634.000	2744.000	2865.000	3173.000	1.036	75
Ntot[22]	2690.197	250.822	2293.000	2506.000	2661.000	2844.000	3243.000	1.162	18
Ntot[23]	2293.481	177.342	1988.000	2165.000	2278.000	2415.000	2651.000	1.049	52
Ntot[24]	1896.001	155.371	1658.000	1781.000	1878.000	1985.000	2282.000	1.072	82
Ntot[25]	1698.145	131.569	1500.000	1598.000	1681.000	1772.000	2018.000	1.035	76
Ntot[26]	1583.542	110.510	1398.000	1505.000	1568.000	1649.000	1841.000	1.027	100
Ntot[27]	1709.049	117.626	1499.000	1627.000	1703.000	1781.000	1975.000	1.045	65
Ntot[28]	1539.455	103.613	1355.000	1472.000	1530.000	1596.000	1777.000	1.029	78
Ntot[29]	1371.076	108.108	1201.000	1296.000	1355.000	1431.000	1618.000	1.043	85
Ntot[30]	1700.215	165.407	1443.000	1580.000	1675.000	1800.000	2065.000	1.016	130
Ntot[31]	1463.763	124.227	1260.000	1377.000	1449.000	1538.000	1757.000	1.055	58
Ntot[32]	1382.668	101.291	1213.000	1314.000	1375.000	1437.000	1628.000	1.182	16

```
Ntot[33]   1478.840 130.841 1279.000 1388.000 1459.000 1543.000 1827.000 1.074    41
Ntot[34]   1381.842 110.281 1200.000 1298.000 1371.000 1456.000 1615.000 1.063    38
Ntot[35]   1384.649 202.103 1109.000 1250.000 1349.000 1477.000 1875.000 1.010   230
Ntot[36]   1439.490 272.747 1072.000 1261.000 1393.000 1561.000 2069.000 1.004   650
deviance   1551.729  27.607 1503.000 1541.000 1554.000 1567.000 1590.000 1.158    36
```

Convergence is perhaps not yet completely satisfactory (if we adhere to the rule that values of Rhat of 1.1 or less indicate convergence), so longer MCMC runs would be necessary to get results for a publication. The parameter estimates for this model, where we assumed that all lapwings start to reproduce at an age of 3 years, are close those under the previous model, where we assumed that all lapwings start to reproduce already at an age of 2 years. The most striking difference is the estimate of the mean fecundity, which was 0.877 for model with a start of three years, but 0.709 for the model with a start of two years. That there is a difference in the estimate of fecundity is not that surprising: there are no data on fecundity, so this parameetr is estimated “by difference”, i.e., in such a way that the match between the population dynamics based on the demographic parameters and the population count data is as close as possible. If the underlying demographic model changes some demographic parameters need a change as well. Those parameters for which no other information is available are often the first to change. This illustrates a difficulty of the integrated population models when parameters are estimated for which no explicit data are available – their estimates typically depend on other assumptions of the model (in our case whether we assume the start of breeding at 2 or at 3 years).

c) Survival is a function of the number of frost days

We consider time here also as a random effect, such that only part of the temporal variation is due to the variation in the number of frost days. Also, we assume an additive model, thus juvenile and adult survival are similarly impacted by frost days. The number of frost days in year 1998 is unknown, hence we only analyze the data up to the year 1997.

```
# Specify model in BUGS language
sink("ipm-lapwing3.bug")
cat("
model {
#-------------------------------------------------
#  Integrated population model
#  - Age structured model with 2 age classes:
#           1-year old and adult (at least 2 years old)
#  - Age at first breeding = 2 years
#  - Prebreeding census, female-based
#  - All vital rates assumed to be constant
#-------------------------------------------------

#-------------------------------------------------
# 1. Define the priors for the parameters
#-------------------------------------------------
# Observation error
tauy <- pow(sigma.y, -2)
sigma.y ~ dunif(0, 50)
sigma2.y <- pow(sigma.y, 2)

# Initial population sizes
N1[1] ~ dnorm(200, 0.0001)I(0,)      # 1-year
Nad[1] ~ dnorm(1000, 0.0001)I(0,)    # Adults
```

```
# Survival and recapture probabilities, as well as productivity
for (t in 1:n.occasions){
   logit(sj[t]) <- mu.sj + beta*frost[t] + ep.sj[t]
   logit(sa[t]) <- mu.sa + beta*frost[t] + ep.sa[t]
   logit(rj[t]) <- mu.rj + ep.rj[t]
   ra[t] <- rj[t]
   log(fec[t]) <- mu.fec + ep.fec[t]

   ep.sj[t] ~ dnorm(0, tau.sj)I(-10,10)
   ep.sa[t] ~ dnorm(0, tau.sa)I(-10,10)
   ep.rj[t] ~ dnorm(0, tau.rj)I(-10,10)
   ep.fec[t] ~ dnorm(0, tau.fec)I(-10,10)
   }

mean.sj ~ dunif(0, 1)
mu.sj <- log(mean.sj / (1-mean.sj))
mean.sa ~ dunif(0, 1)
mu.sa <- log(mean.sa / (1-mean.sa))
mean.rj ~ dunif(0, 1)
mu.rj <- log(mean.rj / (1-mean.rj))
mean.fec ~ dunif(0, 5)
mu.fec <- log(mean.fec)
beta ~ dnorm(0, 0.001)

sigma.sj ~ dunif(0, 10)
tau.sj <- pow(sigma.sj, -2)
sigma2.sj <- pow(sigma.sj, 2)
sigma.sa ~ dunif(0, 10)
tau.sa <- pow(sigma.sa, -2)
sigma2.sa <- pow(sigma.sa, 2)
sigma.rj ~ dunif(0, 10)
tau.rj <- pow(sigma.rj, -2)
sigma2.rj <- pow(sigma.rj, 2)
sigma.fec ~ dunif(0, 10)
tau.fec <- pow(sigma.fec, -2)
sigma2.fec <- pow(sigma.fec, 2)


#-------------------------------------------------
# 2. The likelihoods of the single data sets
#-------------------------------------------------
# 2.1. Likelihood for population count data (state-space model)
   # 3.1.1 System process
   for (t in 2:n.occasions){
      mean1[t] <- fec[t-1] / 2 * sj[t-1] * Nad[t-1]
      N1[t] ~ dpois(mean1[t])
      Nad[t] ~ dbin(sa[t-1], Ntot[t-1])
      }
   for (t in 1:n.occasions){
      Ntot[t] <- Nad[t] + N1[t]   # only breeding birds are counted
      }

   # 3.1.2 Observation process
   for (t in 3:n.occasions){
      y[t] ~ dnorm(Nad[t], tauy)
      }

# Define the multinomial likelihoods
for (t in 1:n.occasions){
   marr.j[t,1:(n.occasions+1)] ~ dmulti(pr.j[t,], rel.j[t])
   }
# Calculate the number of birds released each year
```

```
for (t in 1:n.occasions){
   rel.j[t] <- sum(marr.j[t,])
   }
# Define the cell probabilities of the juvenile m-array
# Main diagonal
for (t in 1:n.occasions){
   pr.j[t,t] <- (1-sj[t])*rj[t]
   # Further above main diagonal
   for (j in (t+2):n.occasions){
      pr.j[t,j] <- sj[t]*prod(sa[(t+1):(j-1)])*(1-sa[j])*ra[j]
      } #j
   # Below main diagonal
   for (j in 1:(t-1)){
      pr.j[t,j] <- 0
      } #j
   } #t
for (t in 1:(n.occasions-1)){
   # One above main diagonal
   pr.j[t,t+1] <- sj[t]*(1-sa[t+1])*ra[t+1]
   } #t
# Last column: probability of non-recovery
for (t in 1:n.occasions){
   pr.j[t,n.occasions+1] <- 1-sum(pr.j[t,1:n.occasions])
   } #t
}
",fill = TRUE)
sink()


# Bundle data
bugs.data <- list(marr.j = dead.recov, y = y[-36], n.occasions =
dim(dead.recov)[2]-1, frost = f)  # last year of census not used

# Initial values
inits <- function(){list(mean.sj = runif(1, 0.4, 0.6), mean.sa = runif(1,
0.7, 0.9), mean.rj = runif(1, 0, 0.2), mean.fec = runif(1, 0, 2), sigma.sj
= runif(1, 0, 1), sigma.sa = runif(1, 0, 1), sigma.rj = runif(1, 0, 1),
sigma.fec = runif(1, 0, 1), N1 = rpois(35, 400), Nad = rpois(35, 1000),
sigma.y = runif(1, 0, 10), beta = rnorm(1))}


# Parameters monitored
parameters <- c("sj", "mean.sj", "sigma2.sj", "sa", "mean.sa", "sigma2.sa",
"rj", "mean.rj", "sigma2.rj", "fec", "mean.fec", "sigma2.fec", "sigma2.y",
"N1", "Nad", "Ntot", "beta")

# MCMC settings
ni <- 20
nt <- 1
nb <- 5
nc <- 3

# Call WinBUGS from R (BRT 808 min)
ipm.lapwing3 <- bugs(bugs.data, inits, parameters, "ipm-lapwing3.bug",
n.chains = nc, n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE,
bugs.directory = bugs.dir, working.directory = getwd())

print(ipm.lapwing3, 3)
Inference for Bugs model at "ipm-lapwing3.bug", fit using WinBUGS,
 3 chains, each with 20000 iterations (first 10000 discarded), n.thin = 3
 n.sims = 10002 iterations saved
              mean      sd     2.5%      25%      50%      75%    97.5%  Rhat n.eff
sj[1]        0.576   0.064    0.432    0.538    0.585    0.619    0.684 1.001  9200
```

sj[2]	0.572	0.059	0.437	0.536	0.579	0.613	0.672	1.002	2300
sj[3]	0.610	0.055	0.493	0.578	0.610	0.642	0.721	1.001	7700
sj[4]	0.658	0.057	0.532	0.624	0.664	0.695	0.762	1.002	1300
sj[5]	0.587	0.057	0.462	0.554	0.590	0.622	0.698	1.001	10000
sj[6]	0.638	0.055	0.537	0.602	0.632	0.672	0.756	1.001	8000
sj[7]	0.619	0.062	0.516	0.572	0.613	0.660	0.751	1.001	10000
sj[8]	0.630	0.057	0.516	0.595	0.626	0.663	0.753	1.003	920
sj[9]	0.690	0.056	0.580	0.654	0.687	0.725	0.807	1.004	580
sj[10]	0.713	0.052	0.619	0.677	0.710	0.747	0.820	1.003	920
sj[11]	0.590	0.070	0.432	0.547	0.601	0.643	0.700	1.002	2400
sj[12]	0.671	0.053	0.553	0.641	0.674	0.705	0.767	1.005	480
sj[13]	0.634	0.056	0.509	0.602	0.639	0.670	0.738	1.002	2200
sj[14]	0.658	0.053	0.557	0.624	0.653	0.691	0.771	1.002	1300
sj[15]	0.644	0.052	0.543	0.610	0.640	0.676	0.753	1.001	4000
sj[16]	0.512	0.065	0.397	0.468	0.508	0.552	0.654	1.006	410
sj[17]	0.664	0.054	0.558	0.631	0.662	0.697	0.776	1.002	2100
sj[18]	0.690	0.049	0.592	0.659	0.689	0.722	0.789	1.006	410
sj[19]	0.489	0.061	0.359	0.452	0.493	0.530	0.605	1.001	3200
sj[20]	0.635	0.057	0.509	0.601	0.640	0.672	0.736	1.002	1300
sj[21]	0.639	0.051	0.529	0.610	0.640	0.670	0.736	1.002	1300
sj[22]	0.503	0.066	0.359	0.463	0.510	0.546	0.624	1.005	470
sj[23]	0.473	0.061	0.347	0.434	0.476	0.514	0.590	1.003	790
sj[24]	0.599	0.053	0.491	0.568	0.598	0.632	0.711	1.002	2700
sj[25]	0.621	0.066	0.467	0.582	0.630	0.668	0.728	1.003	940
sj[26]	0.635	0.061	0.495	0.600	0.644	0.678	0.737	1.004	670
sj[27]	0.689	0.051	0.583	0.658	0.689	0.722	0.792	1.005	470
sj[28]	0.588	0.061	0.449	0.553	0.596	0.627	0.696	1.001	4700
sj[29]	0.594	0.063	0.451	0.559	0.601	0.634	0.711	1.002	1400
sj[30]	0.583	0.063	0.438	0.544	0.592	0.628	0.683	1.001	8600
sj[31]	0.684	0.059	0.571	0.646	0.679	0.722	0.808	1.003	1100
sj[32]	0.666	0.054	0.551	0.634	0.668	0.701	0.770	1.004	740
sj[33]	0.609	0.063	0.494	0.567	0.602	0.647	0.746	1.001	4600
sj[34]	0.657	0.062	0.541	0.618	0.650	0.694	0.790	1.001	3900
sj[35]	0.704	0.059	0.582	0.668	0.702	0.742	0.821	1.002	1400
mean.sj	0.624	0.020	0.584	0.610	0.624	0.637	0.662	1.003	1000
sigma2.sj	0.086	0.074	0.000	0.029	0.071	0.123	0.269	1.070	290
sa[1]	0.814	0.036	0.741	0.792	0.814	0.837	0.885	1.004	1400
sa[2]	0.802	0.035	0.727	0.782	0.803	0.824	0.868	1.003	1200
sa[3]	0.803	0.034	0.731	0.784	0.804	0.825	0.868	1.003	1200
sa[4]	0.843	0.030	0.778	0.826	0.845	0.862	0.899	1.015	150
sa[5]	0.806	0.034	0.739	0.785	0.805	0.828	0.875	1.006	750
sa[6]	0.820	0.033	0.757	0.798	0.819	0.841	0.886	1.004	2600
sa[7]	0.786	0.036	0.718	0.761	0.784	0.809	0.859	1.002	1300
sa[8]	0.756	0.035	0.680	0.733	0.759	0.782	0.815	1.020	120
sa[9]	0.837	0.028	0.774	0.820	0.839	0.856	0.887	1.008	350
sa[10]	0.852	0.026	0.796	0.836	0.853	0.869	0.900	1.004	730
sa[11]	0.858	0.026	0.808	0.839	0.858	0.876	0.909	1.003	1200
sa[12]	0.859	0.025	0.807	0.843	0.859	0.876	0.907	1.007	330
sa[13]	0.841	0.026	0.789	0.825	0.840	0.858	0.891	1.003	760
sa[14]	0.817	0.028	0.755	0.800	0.818	0.836	0.869	1.005	450
sa[15]	0.814	0.028	0.755	0.797	0.815	0.832	0.867	1.012	180
sa[16]	0.717	0.042	0.635	0.690	0.716	0.745	0.800	1.022	100
sa[17]	0.830	0.028	0.770	0.813	0.830	0.848	0.881	1.005	480
sa[18]	0.839	0.027	0.781	0.823	0.841	0.856	0.886	1.012	220
sa[19]	0.751	0.034	0.685	0.728	0.751	0.774	0.819	1.002	2500
sa[20]	0.843	0.026	0.789	0.827	0.843	0.861	0.894	1.012	230
sa[21]	0.838	0.027	0.782	0.820	0.837	0.856	0.892	1.006	410
sa[22]	0.716	0.039	0.630	0.692	0.719	0.743	0.785	1.034	74
sa[23]	0.693	0.038	0.612	0.669	0.694	0.718	0.762	1.007	330
sa[24]	0.762	0.035	0.686	0.741	0.766	0.786	0.821	1.008	300
sa[25]	0.815	0.034	0.738	0.794	0.820	0.840	0.870	1.009	260
sa[26]	0.852	0.027	0.793	0.836	0.853	0.870	0.904	1.004	860
sa[27]	0.842	0.029	0.777	0.825	0.844	0.860	0.894	1.008	300
sa[28]	0.810	0.031	0.748	0.791	0.810	0.830	0.870	1.011	260
sa[29]	0.777	0.035	0.700	0.756	0.782	0.801	0.835	1.005	710
sa[30]	0.768	0.048	0.656	0.740	0.776	0.804	0.839	1.003	1300
sa[31]	0.822	0.030	0.753	0.805	0.825	0.842	0.875	1.006	410
sa[32]	0.849	0.027	0.795	0.832	0.849	0.867	0.901	1.007	380
sa[33]	0.789	0.033	0.721	0.768	0.790	0.810	0.854	1.006	410
sa[34]	0.808	0.032	0.737	0.790	0.810	0.828	0.865	1.006	430
sa[35]	0.863	0.029	0.803	0.846	0.864	0.882	0.918	1.007	320
mean.sa	0.814	0.013	0.789	0.805	0.814	0.822	0.840	1.022	130
sigma2.sa	0.057	0.043	0.001	0.025	0.049	0.079	0.161	1.006	1600
rj[1]	0.018	0.005	0.010	0.015	0.018	0.021	0.030	1.002	1600
rj[2]	0.021	0.005	0.013	0.018	0.021	0.025	0.034	1.002	1400
rj[3]	0.018	0.005	0.010	0.014	0.017	0.020	0.028	1.001	6600
rj[4]	0.016	0.004	0.009	0.013	0.015	0.018	0.025	1.002	2000
rj[5]	0.013	0.003	0.008	0.011	0.013	0.015	0.020	1.001	7800

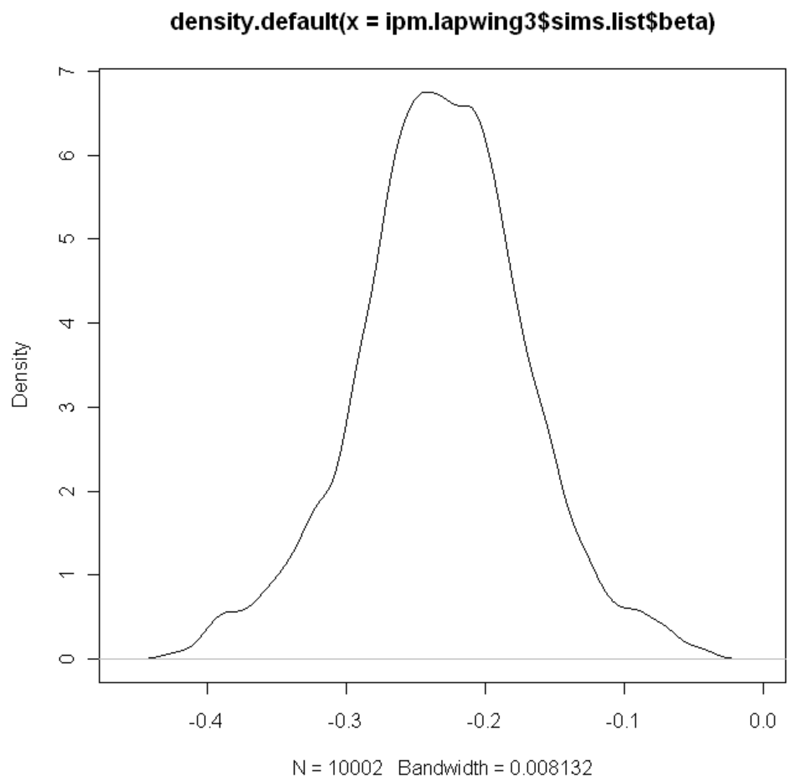
rj[6]	0.011	0.003	0.006	0.009	0.010	0.012	0.016	1.001	7700
rj[7]	0.014	0.003	0.009	0.011	0.013	0.015	0.020	1.001	9400
rj[8]	0.013	0.003	0.009	0.011	0.013	0.015	0.019	1.004	700
rj[9]	0.010	0.002	0.006	0.008	0.010	0.011	0.015	1.002	1300
rj[10]	0.010	0.002	0.006	0.009	0.010	0.012	0.015	1.001	3800
rj[11]	0.010	0.002	0.007	0.009	0.010	0.012	0.015	1.001	8000
rj[12]	0.010	0.002	0.007	0.009	0.010	0.011	0.015	1.002	1800
rj[13]	0.011	0.002	0.007	0.009	0.010	0.012	0.015	1.001	10000
rj[14]	0.012	0.002	0.008	0.010	0.011	0.013	0.016	1.001	3500
rj[15]	0.011	0.002	0.007	0.009	0.010	0.012	0.015	1.002	2600
rj[16]	0.011	0.002	0.008	0.009	0.011	0.012	0.015	1.016	140
rj[17]	0.009	0.002	0.005	0.007	0.008	0.010	0.013	1.004	580
rj[18]	0.012	0.002	0.008	0.010	0.012	0.013	0.017	1.005	510
rj[19]	0.012	0.002	0.008	0.010	0.012	0.013	0.016	1.001	10000
rj[20]	0.009	0.002	0.006	0.008	0.009	0.010	0.014	1.003	980
rj[21]	0.012	0.002	0.008	0.010	0.011	0.013	0.016	1.002	2000
rj[22]	0.009	0.002	0.007	0.008	0.009	0.010	0.013	1.012	180
rj[23]	0.010	0.002	0.007	0.009	0.010	0.011	0.013	1.002	1700
rj[24]	0.008	0.001	0.005	0.007	0.008	0.009	0.011	1.001	3900
rj[25]	0.008	0.002	0.005	0.007	0.008	0.009	0.011	1.004	630
rj[26]	0.009	0.002	0.006	0.008	0.009	0.010	0.013	1.002	1600
rj[27]	0.008	0.002	0.005	0.007	0.008	0.009	0.012	1.004	610
rj[28]	0.008	0.001	0.005	0.007	0.007	0.008	0.011	1.001	5300
rj[29]	0.006	0.001	0.004	0.005	0.006	0.007	0.009	1.001	6200
rj[30]	0.009	0.002	0.006	0.008	0.009	0.010	0.013	1.002	1900
rj[31]	0.008	0.002	0.005	0.007	0.008	0.009	0.012	1.001	4200
rj[32]	0.009	0.002	0.006	0.007	0.008	0.010	0.013	1.003	970
rj[33]	0.006	0.001	0.004	0.006	0.006	0.007	0.010	1.001	4300
rj[34]	0.007	0.001	0.005	0.006	0.007	0.008	0.011	1.002	2900
rj[35]	0.006	0.001	0.003	0.005	0.005	0.006	0.009	1.001	10000
mean.rj	0.010	0.001	0.009	0.010	0.010	0.010	0.012	1.005	460
sigma2.rj	0.148	0.058	0.061	0.107	0.140	0.179	0.285	1.001	3600
fec[1]	0.991	0.472	0.291	0.651	0.904	1.258	2.108	1.007	1500
fec[2]	0.937	0.305	0.412	0.717	0.915	1.129	1.604	1.006	400
fec[3]	1.078	0.286	0.576	0.876	1.063	1.256	1.693	1.004	560
fec[4]	1.612	0.342	1.013	1.369	1.592	1.825	2.349	1.002	3300
fec[5]	1.238	0.323	0.687	1.011	1.209	1.435	1.942	1.004	670
fec[6]	1.497	0.312	0.926	1.278	1.474	1.700	2.147	1.005	510
fec[7]	0.303	0.142	0.095	0.197	0.280	0.386	0.635	1.018	130
fec[8]	0.664	0.179	0.363	0.537	0.648	0.773	1.053	1.011	190
fec[9]	0.607	0.181	0.300	0.479	0.590	0.717	1.010	1.003	1100
fec[10]	0.669	0.179	0.358	0.545	0.655	0.778	1.064	1.004	5500
fec[11]	0.615	0.193	0.283	0.477	0.601	0.737	1.028	1.010	230
fec[12]	0.548	0.157	0.268	0.439	0.538	0.643	0.882	1.020	160
fec[13]	1.143	0.240	0.724	0.973	1.130	1.293	1.666	1.015	160
fec[14]	0.793	0.209	0.432	0.645	0.777	0.921	1.237	1.018	120
fec[15]	1.108	0.301	0.555	0.903	1.093	1.295	1.748	1.032	110
fec[16]	0.837	0.227	0.436	0.681	0.820	0.975	1.326	1.002	4100
fec[17]	0.470	0.158	0.208	0.357	0.452	0.567	0.812	1.018	190
fec[18]	0.857	0.210	0.488	0.706	0.845	0.993	1.291	1.007	650
fec[19]	0.589	0.200	0.261	0.445	0.569	0.710	1.036	1.003	2100
fec[20]	1.273	0.238	0.846	1.111	1.261	1.418	1.777	1.010	220
fec[21]	0.919	0.308	0.405	0.703	0.889	1.096	1.613	1.040	64
fec[22]	0.672	0.272	0.245	0.474	0.640	0.834	1.304	1.012	250
fec[23]	0.496	0.204	0.165	0.350	0.475	0.619	0.965	1.013	170
fec[24]	0.356	0.145	0.116	0.250	0.340	0.445	0.676	1.009	460
fec[25]	0.496	0.180	0.191	0.370	0.479	0.606	0.903	1.008	520
fec[26]	0.480	0.182	0.180	0.347	0.460	0.590	0.888	1.009	340
fec[27]	0.788	0.225	0.405	0.634	0.764	0.920	1.288	1.005	440
fec[28]	0.390	0.188	0.110	0.258	0.359	0.490	0.837	1.022	150
fec[29]	0.246	0.125	0.065	0.157	0.224	0.312	0.552	1.021	140
fec[30]	1.549	0.400	0.881	1.278	1.513	1.768	2.436	1.011	210
fec[31]	0.638	0.244	0.233	0.459	0.615	0.796	1.167	1.002	1300
fec[32]	0.659	0.233	0.252	0.498	0.643	0.801	1.170	1.007	350
fec[33]	0.808	0.268	0.347	0.616	0.786	0.975	1.395	1.004	2200
fec[34]	0.855	0.644	0.188	0.462	0.702	1.050	2.466	1.004	580
fec[35]	0.848	0.634	0.190	0.464	0.694	1.044	2.371	1.003	1000
mean.fec	0.700	0.104	0.497	0.632	0.696	0.772	0.908	1.063	47
sigma2.fec	0.391	0.183	0.142	0.264	0.355	0.477	0.850	1.021	110
sigma2.y	2007.595	423.989	904.527	1794.250	2123.000	2337.000	2486.000	1.001	5200
N1[1]	245.655	94.462	65.623	179.325	245.050	309.000	436.292	1.008	1100
N1[2]	295.376	124.710	89.892	204.125	279.950	373.875	566.695	1.008	1900
N1[3]	282.071	84.819	127.500	222.500	278.900	337.900	451.495	1.011	250
N1[4]	357.233	86.334	196.300	298.025	356.200	413.400	536.597	1.004	560
N1[5]	582.253	105.077	381.202	511.100	581.700	650.200	793.787	1.005	7200
N1[6]	444.383	102.017	259.200	370.900	439.300	512.475	652.995	1.005	510
N1[7]	694.033	127.489	455.307	606.825	690.750	776.900	957.590	1.003	790
N1[8]	143.347	69.053	42.290	92.265	132.300	182.800	305.582	1.017	130

N1[9]	367.005	90.293	204.800	302.925	361.700	425.475	557.295	1.011	200
N1[10]	298.333	83.109	149.900	240.300	292.200	352.400	476.297	1.002	1800
N1[11]	357.573	88.980	198.805	296.900	351.950	413.800	544.992	1.004	4200
N1[12]	274.733	79.367	133.900	217.700	270.950	325.600	445.200	1.010	230
N1[13]	295.736	79.686	145.607	240.600	293.050	345.800	462.797	1.018	200
N1[14]	587.577	108.222	390.200	511.600	585.800	656.400	817.997	1.020	120
N1[15]	419.046	104.108	235.502	347.100	411.550	485.400	636.887	1.019	110
N1[16]	639.648	165.254	331.012	528.500	631.600	742.675	995.082	1.039	87
N1[17]	383.116	96.270	208.602	316.200	378.100	443.875	586.597	1.005	570
N1[18]	270.081	86.707	123.000	206.900	261.850	325.600	458.097	1.016	220
N1[19]	522.445	123.752	303.302	433.350	515.750	603.375	780.500	1.005	1300
N1[20]	243.595	81.737	104.102	184.700	236.600	295.000	424.297	1.003	1200
N1[21]	671.868	103.051	471.710	601.800	671.000	741.675	878.900	1.013	200
N1[22]	473.648	157.697	206.707	363.025	459.450	563.100	830.585	1.045	56
N1[23]	317.854	121.247	119.507	231.200	304.100	390.500	595.997	1.011	230
N1[24]	196.854	80.281	63.881	138.725	188.900	246.000	375.487	1.012	180
N1[25]	146.318	58.972	47.190	103.400	141.200	183.100	276.900	1.009	510
N1[26]	183.318	62.056	75.341	140.100	179.600	223.075	314.000	1.006	630
N1[27]	165.183	59.823	63.304	121.800	159.500	202.675	296.492	1.007	510
N1[28]	295.096	78.508	157.702	240.400	289.200	343.775	461.597	1.004	590
N1[29]	118.520	57.388	33.241	78.292	109.200	148.500	254.192	1.021	160
N1[30]	76.042	38.731	19.500	48.120	69.710	96.670	165.997	1.020	150
N1[31]	418.666	85.562	257.600	359.500	415.200	473.975	601.400	1.017	140
N1[32]	166.514	61.864	60.190	120.725	162.400	208.200	297.997	1.004	730
N1[33]	212.328	72.636	82.581	161.125	209.400	257.975	364.097	1.007	330
N1[34]	236.326	75.027	102.702	183.600	232.850	283.875	397.490	1.003	2600
N1[35]	261.570	201.084	55.025	139.000	214.000	323.000	762.975	1.005	510
Nad[1]	1064.916	90.477	892.305	1002.000	1065.000	1125.000	1242.000	1.005	440
Nad[2]	1067.848	106.576	849.000	999.000	1075.000	1145.000	1256.000	1.009	320
Nad[3]	1092.064	43.084	1005.000	1064.000	1093.000	1121.000	1177.000	1.003	880
Nad[4]	1103.881	42.905	1022.000	1075.000	1103.000	1132.000	1189.000	1.004	710
Nad[5]	1231.960	44.583	1145.000	1202.000	1233.000	1262.000	1320.000	1.002	3100
Nad[6]	1462.226	43.220	1376.000	1434.000	1462.000	1491.000	1547.000	1.007	310
Nad[7]	1562.834	44.116	1476.000	1534.000	1563.000	1592.000	1650.000	1.001	3200
Nad[8]	1771.587	44.625	1682.000	1742.000	1772.000	1802.000	1858.000	1.004	570
Nad[9]	1438.254	41.049	1360.025	1410.000	1438.000	1466.000	1521.000	1.002	1900
Nad[10]	1509.023	42.864	1425.025	1481.000	1509.000	1537.000	1594.000	1.005	510
Nad[11]	1538.233	42.921	1450.000	1511.000	1539.000	1567.000	1619.975	1.002	2500
Nad[12]	1624.943	43.372	1540.000	1596.000	1625.000	1654.000	1710.000	1.006	370
Nad[13]	1630.548	40.814	1551.000	1603.000	1630.000	1657.000	1712.000	1.001	5600
Nad[14]	1617.882	43.429	1531.000	1589.000	1618.000	1647.000	1703.000	1.015	140
Nad[15]	1801.159	44.443	1714.000	1771.000	1800.000	1831.000	1889.000	1.005	540
Nad[16]	1806.922	44.219	1718.000	1778.000	1807.000	1836.000	1893.000	1.003	900
Nad[17]	1750.597	43.831	1662.025	1722.000	1751.000	1780.000	1835.000	1.004	580
Nad[18]	1770.139	42.828	1684.000	1743.000	1771.000	1798.000	1854.000	1.001	10000
Nad[19]	1709.020	42.088	1628.000	1680.000	1709.000	1737.000	1793.000	1.002	1700
Nad[20]	1674.742	43.343	1588.000	1646.000	1675.000	1703.000	1758.975	1.003	940
Nad[21]	1616.536	42.630	1532.000	1588.000	1617.000	1645.000	1701.000	1.002	1400
Nad[22]	1916.323	45.164	1826.000	1887.000	1917.000	1946.000	2006.000	1.005	550
Nad[23]	1708.045	42.908	1621.000	1680.000	1708.000	1737.000	1792.000	1.008	270
Nad[24]	1400.258	40.123	1321.000	1374.000	1401.000	1427.000	1481.000	1.002	2900
Nad[25]	1213.800	38.000	1139.025	1188.000	1214.000	1239.000	1288.000	1.002	1400
Nad[26]	1103.722	36.004	1034.000	1080.000	1103.000	1127.000	1176.000	1.002	2000
Nad[27]	1095.004	37.513	1023.000	1070.000	1095.000	1120.000	1170.000	1.003	1100
Nad[28]	1058.658	36.809	986.000	1034.000	1058.000	1083.000	1132.000	1.003	1200
Nad[29]	1096.176	40.266	1015.000	1069.000	1097.000	1124.000	1175.000	1.002	1700
Nad[30]	939.057	34.023	872.000	916.000	939.000	962.000	1007.000	1.003	1300
Nad[31]	769.102	42.414	684.000	742.000	771.000	798.000	848.000	1.003	770
Nad[32]	976.420	42.255	891.025	949.000	977.000	1005.000	1057.000	1.012	180
Nad[33]	970.030	37.972	895.000	945.000	970.000	995.000	1046.000	1.002	1300
Nad[34]	931.228	39.762	853.000	905.000	931.000	958.000	1011.000	1.007	450
Nad[35]	942.681	42.623	860.000	914.000	942.000	970.000	1028.000	1.004	1000
Ntot[1]	1310.570	122.989	1060.000	1229.000	1316.000	1395.000	1539.000	1.006	410
Ntot[2]	1363.222	80.886	1214.000	1308.000	1360.000	1413.000	1531.000	1.001	4100
Ntot[3]	1374.128	77.140	1236.000	1320.000	1370.000	1424.000	1535.000	1.006	410
Ntot[4]	1461.109	75.251	1324.000	1409.000	1457.000	1508.000	1621.000	1.008	270
Ntot[5]	1814.212	95.638	1635.000	1751.000	1813.000	1873.750	2017.000	1.004	2300
Ntot[6]	1906.610	93.471	1732.000	1842.000	1902.000	1969.000	2098.000	1.003	1700
Ntot[7]	2256.865	117.922	2038.000	2177.000	2251.000	2332.000	2502.000	1.005	530
Ntot[8]	1914.932	78.880	1788.000	1858.000	1906.000	1961.000	2092.000	1.029	89
Ntot[9]	1805.258	81.422	1659.025	1750.000	1801.000	1856.000	1980.000	1.015	190
Ntot[10]	1807.358	75.431	1670.000	1755.000	1802.000	1854.000	1972.000	1.004	660
Ntot[11]	1895.811	78.476	1751.000	1841.000	1891.000	1946.000	2060.000	1.001	4600
Ntot[12]	1899.679	72.729	1765.000	1849.000	1896.000	1947.000	2051.975	1.005	470
Ntot[13]	1926.286	75.882	1784.000	1877.000	1923.000	1974.000	2083.000	1.010	220
Ntot[14]	2205.457	98.137	2031.000	2136.000	2200.000	2266.000	2419.975	1.008	300
Ntot[15]	2220.204	95.208	2047.000	2156.000	2215.000	2280.000	2419.975	1.014	150
Ntot[16]	2446.563	158.495	2150.025	2340.000	2438.000	2545.000	2780.000	1.021	110

```
Ntot[17]   2133.711  88.492 1973.025 2073.000 2128.000 2189.000 2321.975 1.003  1000
Ntot[18]   2040.220  81.273 1899.000 1984.000 2032.000 2091.000 2219.000 1.015   200
Ntot[19]   2231.466 119.439 2012.025 2146.000 2226.000 2310.000 2482.975 1.004  1200
Ntot[20]   1918.340  75.437 1781.000 1866.000 1915.000 1966.000 2076.000 1.005   560
Ntot[21]   2288.402  94.071 2106.000 2225.000 2288.000 2349.000 2484.000 1.009   250
Ntot[22]   2389.969 152.608 2140.000 2282.000 2374.000 2478.000 2743.000 1.039    62
Ntot[23]   2025.899 120.132 1827.000 1939.000 2014.000 2100.000 2300.950 1.005   490
Ntot[24]   1597.112  81.812 1462.000 1539.000 1588.000 1645.000 1783.000 1.010   210
Ntot[25]   1360.116  61.857 1252.000 1316.000 1355.000 1399.000 1493.000 1.006   440
Ntot[26]   1287.037  59.312 1181.000 1245.000 1284.000 1325.000 1410.975 1.004   690
Ntot[27]   1260.180  57.094 1161.000 1221.000 1256.000 1295.000 1385.000 1.003  1000
Ntot[28]   1353.757  71.578 1227.000 1303.000 1350.000 1399.000 1503.975 1.003   870
Ntot[29]   1214.698  63.367 1105.000 1172.000 1209.000 1252.000 1350.000 1.009   410
Ntot[30]   1015.099  52.080  924.305  979.300 1012.000 1045.750 1128.000 1.012   210
Ntot[31]   1187.768  69.112 1063.000 1140.000 1184.000 1231.750 1335.000 1.014   160
Ntot[32]   1142.935  56.188 1040.000 1104.000 1140.000 1179.000 1260.000 1.001  4000
Ntot[33]   1182.356  68.430 1060.000 1135.000 1179.000 1225.000 1325.975 1.008   300
Ntot[34]   1167.559  68.939 1046.000 1120.000 1163.000 1210.000 1313.000 1.003  5600
Ntot[35]   1204.251 206.078  968.000 1078.000 1161.000 1272.000 1709.925 1.003  1200
beta         -0.231   0.062   -0.363   -0.269   -0.231   -0.192   -0.103 1.021   100
deviance   1548.107  18.203 1510.000 1537.000 1549.000 1560.000 1582.000 1.001 10000
```

The most interesting parameter in this exercise is the slope parameter `beta`, which expresses the strength of the relationship between survival and the number of frost days. We plot the posterior distribution of `beta`:

```
plot(density(ipm.lapwing3$sims.list$beta))
```



We clearly see that `beta` is negative, thus the survival of both age classes declines as the number of frost days increases. We can also easily compute the probability `beta < 0`:

```
sum(ipm.lapwing3$sims.list$beta<0)/length(ipm.lapwing3$sims.list$beta)
[1] 1
```

The result is 1, so, within the limits imposed by a correlative study, we can say that there is no doubt that `beta` is lower than 0, and thus that the number of frost days negatively impacted lapwing survival.

Chapter 12

Exercise 1

Task: With hierarchical models such as the binomial-mixture model, we have several kinds of covariates: here, we have covariates that vary among sites ('site covariates') and those that vary among individual surveys ('sampling covariates'). It is important in practice to know how to fit both kinds. Invent a sampling covariate in the example of Section 12.2.2 and fit it also to see how this works.

Solution: We assume that you have a data set ready that was generated with the function in section 12.2.2. Now we generate a covariate matrix with just some random numbers and fit that into the detection model as well to see how that goes. We will also fit that model using maximum likelihood in the R package **unmarked**.

```
# Bundle data, including the new sampling covariate X2
y <- data$y
X2 <- matrix(rnorm(length(y)), ncol = ncol(y))
win.data <- list(y = y, R = nrow(y), T = ncol(y), X = data$X, X2 = X2)

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

# Priors
alpha0 ~ dunif(-10, 10)
alpha1 ~ dunif(-10, 10)
beta0 ~ dunif(-10, 10)
beta1 ~ dunif(-10, 10)
beta2 ~ dunif(-10, 10)

# Likelihood
# Ecological model for true abundance
for (i in 1:R){
   N[i] ~ dpois(lambda[i])
   log(lambda[i]) <- alpha0 + alpha1 * X[i]

   # Observation model for replicated counts
   for (j in 1:T){
      y[i,j] ~ dbin(p[i,j], N[i])
      p[i,j] <- exp(lp[i,j])/(1+exp(lp[i,j]))
      lp[i,j] <- beta0 + beta1 * X[i] + beta2 * X2[i,j]
      } #j
   } #i

# Derived quantities
totalN <- sum(N[])
}
",fill = TRUE)
sink()

# Initial values
Nst <- apply(y, 1, max) + 1   # Important to give good inits for latent N
inits <- function() list(N = Nst, alpha0 = runif(1, -1, 1), alpha1 =
runif(1, -1, 1), beta0 = runif(1, -1, 1), beta1 = runif(1, -1, 1), beta2 =
runif(1, -1, 1))
```

```
# Parameters monitored
params <- c("totalN", "alpha0", "alpha1", "beta0", "beta1", "beta2")

# MCMC settings
ni <- 22000
nt <- 20
nb <- 2000
nc <- 3

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

# Summarize posteriors
print(out, dig = 2)
Inference for Bugs model at "model.txt", fit using WinBUGS,
 3 chains, each with 22000 iterations (first 2000 discarded), n.thin = 20
 n.sims = 3000 iterations saved
            mean     sd   2.5%     25%     50%     75%   97.5% Rhat n.eff
totalN   1589.10 630.22 833.95 1170.00 1418.00 1791.25 3343.10 1.01  3000
alpha0      1.12   0.15   0.86    1.01    1.10    1.21    1.44 1.01  2900
alpha1      2.61   0.37   1.96    2.35    2.58    2.83    3.43 1.01  1400
beta0      -0.09   0.22  -0.55   -0.23   -0.07    0.06    0.29 1.01  2100
beta1      -4.68   0.39  -5.53   -4.93   -4.66   -4.41   -3.97 1.00  3000
beta2      -0.10   0.06  -0.22   -0.14   -0.10   -0.06    0.02 1.00  3000
deviance  932.26  20.84 893.70  917.40  930.90  946.40  974.70 1.00  3000
```

As an aside, we note that with several replicates of the simulated data, we got the impression that the posterior of `beta2` was often moved away from zero (though rarely did the 95% credible interval *not* include 0). So perhaps our choice of priors was not as innocuous as we had wanted? To check whether we were inadvertently introducing information via the priors, we may compare the Bayesian solutions with maximum likelihood estimates (MLEs). For N-mixture models, we can obtain MLEs using the function `pcount()` in the new R package **unmarked**, which we now load (must be installed first, of course).

```
library(unmarked)
```

We create the **unmarked** data frame for the `pcount` function and fit the model.

```
umf <- unmarkedFramePCount(y = y, siteCovs = data.frame(X = data$X),
obsCovs = list(X2 = X2))
#summary(umf)
summary(fm <- pcount(~X+X2 ~X, umf, engine = "C")))

Call:
pcount(formula = ~X + X2 ~ X, data = umf, engine = "C")

Abundance (log-scale):
            Estimate    SE    z  P(>|z|)
(Intercept)     1.10 0.137 8.02 1.10e-15
X               2.55 0.340 7.48 7.21e-14

Detection (logit-scale):
            Estimate     SE       z  P(>|z|)
(Intercept)  -0.0629 0.2013  -0.312 7.55e-01
X            -4.6203 0.3729 -12.391 2.93e-35
X2           -0.1013 0.0621  -1.631 1.03e-01

AIC: 1190.049
Number of sites: 200
```

```
optim convergence code: 0
optim iterations: 60
Bootstrap iterations: 0

Warnmeldung:
In pcount(~X + X2 ~ X, umf, engine = "C") :
  K was not specified and was set to 105.
```

Fitting the model using WinBUGS and unmarked for a couple of replicate data sets however dispelled our worries: generally, the posterior means were very similar to the MLEs.

Exercise 2

Task: In the fritillary data, fit a simpler binomial-mixture model than the one in section 12.3.3. with detection random effects specific to day and site (i.e., drop the index *j* in the $\delta_{i,j,k}$). See whether that model also fits.

Solution: Random effects are effects of the levels of a grouping factor that are constrained by the assumption of a common prior distribution. These effects are assumed to be exchangeable, i.e., there is no further structure recognizable among them. It is up to the ecologist to decide on the grouping levels of one or several factors, to which this assumption of exchangeability (and therefore of a common prior distribution) applies (see also exercise 4 in chapter 4).

In the fritillary example in section 12.3.3, we assumed that detection probability, on the logit scale, has an exchangeable contribution from each combination of site (*i*), replicate (*j*) and day (*k*), and that all those contriutions are drawn from a common normal distribution. This could be so if, say, the meteorological conditions affecting detection probability vary swiftly from one hour to the next and therefore, from one survey to the next. However, if we can assume that the effects of such unmeasured factors act at the scale of days rather than of hours, a model with random site-by-day effects may be more adequate. We fit such a model here. See also the next exercise, which fits an even more complex random effects model.

We assume that you have read in the data into your R workspace and therefore, you are ready to run the analysis. We will denote the model in this exercise as model 2A, and the one in the next exercise as 2B. In this exercise, we have to define the random site-by-day effects and `p[i,k]` outside of the loop over replicates and drop the index *j* in each `p[]` and `lp[]`.

```
# Specify model in BUGS language
sink("Nmix2A.txt")
cat("
model {

# Priors
for (k in 1:7){
   alpha.lam[k] ~ dnorm(0, 0.1)
   beta[k] ~ dnorm(0, 0.1)
   }

# Abundance site and detection site-by-day random effects
for (i in 1:R){
```

```
   eps[i] ~ dnorm(0, tau.lam)
   }
tau.lam <- 1 / (sd.lam * sd.lam)
sd.lam ~ dunif(0, 3)
tau.p <- 1 / (sd.p * sd.p)
sd.p ~ dunif(0, 3)

# Likelihood
# Ecological model for true abundance
for (i in 1:R){                              # Loop for R sites (95)
   for (k in 1:7){                           # Loop for days (7)
      N[i,k] ~ dpois(lambda[i,k])            # Abundance
      log(lambda[i,k]) <- alpha.lam[k] + eps[i]

      # Observation model for replicated counts
      lp[i,k] ~ dnorm(beta[k], tau.p)       # random delta def. implicitly
      p[i,k] <- 1 / (1 + exp(-lp[i,k]))
      for (j in 1:T){                        # Loop for temporal reps (2)
         y[i,j,k] ~ dbin(p[i,k], N[i,k])    # Detection

         # Assess model fit using Chi-squared discrepancy
         # Compute fit statistic for observed data
         eval[i,j,k] <- p[i,k] * N[i,k]
         E[i,j,k] <- pow((y[i,j,k] - eval[i,j,k]),2) / (eval[i,j,k]+0.5)
         # Generate replicate data and compute fit stats for them
         y.new[i,j,k] ~ dbin(p[i,k], N[i,k])
         E.new[i,j,k] <- pow((y.new[i,j,k] - eval[i,j,k]),2) /
(eval[i,j,k]+0.5)
         } #j
         ik.p[i,k] <- mean(p[i,k])
      } #k
   } #i

# Derived and other quantities
for (k in 1:7){
   totalN[k] <- sum(N[,k])   # Estimate total pop. size across all sites
   mean.abundance[k] <- mean(lambda[,k])
   mean.N[k] <- mean(N[,k])
   mean.detection[k] <- mean(ik.p[,k])
   }
fit <- sum(E[,,])
fit.new <- sum(E.new[,,])
}
",fill = TRUE)
sink()

# Bundle data
R = nrow(y)
T = ncol(y)
win.data <- list(y = y, R = R, T = T)

# Initial values
Nst <- apply(y, c(1, 3), max) + 1
Nst[is.na(Nst)] <- 1
inits <- function(){list(N = Nst, alpha.lam = runif(7, -1, 1), beta =
runif(7, -1, 1), sd.lam = runif(1, 0, 1), sd.p = runif(1, 0, 1))}

# Parameters monitored
params <- c("totalN", "alpha.lam", "beta", "sd.lam", "sd.p",
"mean.abundance", "mean.N", "mean.detection", "fit", "fit.new")

# MCMC settings
```

```
ni <- 350000
nt <- 300
nb <- 50000
nc <- 3

# Call WinBUGS from R
out2A <- bugs(win.data, inits, params, "Nmix2A.txt", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = FALSE, bugs.directory =
bugs.dir, working.directory = getwd())

# Evaluation of fit
plot(out2A$sims.list$fit, out2A$sims.list$fit.new, main = "", xlab =
"Discrepancy actual data", ylab = "Discrepancy replicate data", frame.plot
= FALSE, xlim = c(100, 300), ylim = c(100, 300))
abline(0, 1, lwd = 2, col = "black")
mean(out2A$sims.list$fit.new > out2A$sims.list$fit)
```

Indeed, with a Bayesian p-value of about 0.4, this model also fits and we can summarize the posteriors.

```
# Summarize posteriors
print(out2A, dig = 2)
Inference for Bugs model at "Nmix2A.txt", fit using WinBUGS,
 3 chains, each with 350000 iterations (first 50000 discarded), n.thin = 300
 n.sims = 3000 iterations saved
                     mean      sd   2.5%     25%     50%     75%    97.5% Rhat n.eff
totalN[1]         1611.89 3718.15   6.00   22.00   81.00  616.25 14382.99 1.28    13
totalN[2]           13.46   38.23   0.00    0.00    1.00    8.00   128.03 1.15    99
totalN[3]          429.05  540.51  32.00   97.00  219.00  517.25  1973.22 1.04    64
totalN[4]          359.60  574.62  44.00   74.00  127.00  323.00  2312.32 1.02   170
totalN[5]         3483.05 2421.41 672.90 1608.75 2771.50 4822.50  9426.77 1.11    24
totalN[6]          833.11 1609.27 141.00  220.00  310.00  509.00  6794.90 1.32    15
totalN[7]          883.31 1717.79 127.97  221.75  358.00  708.25  6827.32 1.05    88
alpha.lam[1]        -1.15    2.31  -4.58   -2.93   -1.66    0.38     3.57 1.27    13
alpha.lam[2]        -5.36    2.10  -9.36   -6.83   -5.46   -3.88    -1.20 1.01   520
alpha.lam[3]        -0.61    1.17  -2.75   -1.48   -0.62    0.23     1.59 1.04    62
alpha.lam[4]        -0.90    1.12  -2.46   -1.74   -1.19   -0.29     1.69 1.02   150
alpha.lam[5]         1.86    0.76   0.41    1.31    1.88    2.45     3.20 1.10    25
alpha.lam[6]        -0.05    0.98  -1.25   -0.68   -0.29    0.24     2.84 1.30    15
alpha.lam[7]         0.06    0.99  -1.33   -0.65   -0.15    0.53     2.62 1.04    87
beta[1]             -4.36    2.48  -9.25   -6.08   -4.03   -2.50    -0.31 1.23    14
beta[2]             -2.38    3.54  -8.31   -4.92   -2.88   -0.14     5.20 1.00  1000
beta[3]             -3.73    1.33  -6.15   -4.70   -3.73   -2.80    -1.17 1.04    68
beta[4]             -2.00    1.41  -5.04   -2.92   -1.76   -0.91     0.20 1.01   180
beta[5]             -3.93    0.79  -5.28   -4.54   -3.94   -3.34    -2.37 1.10    27
beta[6]             -1.88    1.20  -5.17   -2.33   -1.64   -1.09    -0.21 1.26    17
beta[7]             -2.27    1.14  -5.22   -2.85   -2.08   -1.46    -0.59 1.04    86
sd.lam               1.91    0.25   1.48    1.74    1.90    2.07     2.44 1.00  3000
sd.p                 1.02    0.14   0.76    0.91    1.01    1.11     1.32 1.00  1300
mean.abundance[1]   16.98   39.17   0.05    0.24    0.85    6.26   151.21 1.28    13
mean.abundance[2]    0.15    0.40   0.00    0.00    0.02    0.09     1.35 1.01   560
mean.abundance[3]    4.52    5.70   0.33    1.04    2.30    5.47    20.60 1.04    65
mean.abundance[4]    3.79    6.05   0.43    0.78    1.34    3.38    23.97 1.02   170
mean.abundance[5]   36.67   25.50   6.99   16.94   29.21   50.62    99.34 1.11    24
mean.abundance[6]    8.77   16.95   1.47    2.31    3.27    5.38    72.15 1.32    15
mean.abundance[7]    9.29   18.08   1.34    2.35    3.73    7.47    71.50 1.05    89
mean.N[1]           16.97   39.14   0.06    0.23    0.85    6.49   151.33 1.28    13
mean.N[2]            0.14    0.40   0.00    0.00    0.01    0.08     1.35 1.15    99
mean.N[3]            4.52    5.69   0.34    1.02    2.31    5.44    20.77 1.04    64
mean.N[4]            3.79    6.05   0.46    0.78    1.34    3.40    24.34 1.02   170
mean.N[5]           36.66   25.49   7.08   16.94   29.17   50.76    99.23 1.11    24
mean.N[6]            8.77   16.94   1.48    2.32    3.26    5.36    71.53 1.32    15
mean.N[7]            9.30   18.08   1.35    2.33    3.77    7.46    71.86 1.05    88
mean.detection[1]    0.08    0.12   0.00    0.00    0.03    0.10     0.44 1.26    13
mean.detection[2]    0.27    0.34   0.00    0.01    0.08    0.47     0.99 1.00   730
mean.detection[3]    0.06    0.07   0.00    0.02    0.04    0.08     0.27 1.04    65
mean.detection[4]    0.21    0.15   0.01    0.08    0.19    0.31     0.54 1.02   160
mean.detection[5]    0.04    0.03   0.01    0.02    0.03    0.05     0.12 1.10    26
mean.detection[6]    0.21    0.12   0.01    0.12    0.20    0.29     0.45 1.32    15
mean.detection[7]    0.16    0.10   0.01    0.08    0.15    0.23     0.38 1.05    82
fit                173.00   14.59 146.90  162.80  171.90  182.40   204.50 1.02   110
```

```
fit.new          169.44   21.10 131.50  154.70  168.10  183.50   213.50 1.02   110
deviance         780.57   30.38 721.99  759.90  779.80  800.70   841.81 1.04    65
```

<u>Exercise 3</u>

Task: In the fritillary data, fit a more complex binomial-mixture model by introducing (in addition to the random site-day-rep effect) a random site effect in the linear predictor for detection in the model in section 12.3.3. Compare the estimates under the model in section 12.3.3. and those in exercises 2 and 3. Explain.

Solution: One of the practical challenges in this exercise is to put the differently-indexed quantities (e.g., `mu.lp[i,k]`) at the right place within the hierarchy of loops. It is easy to produce an error message such as "`multiple definition of node mu.lp[i,k]`".

```
# Specify model in BUGS language
sink("Nmix2B.txt")
cat("
model {

# Priors
for (k in 1:7){
   alpha.lam[k] ~ dnorm(0, 0.1)
   beta[k] ~ dnorm(0, 0.1)
   }

# Abundance and detection site effects
# and detection site-by-day-by-rep random effects
for (i in 1:R){
   eps.lam[i] ~ dnorm(0, tau.lam)       # Random site effects on abundance
   eps.p[i] ~ dnorm(0, tau.site.p)      # Random site effects on detection
   }
tau.lam <- pow(sd.lam, -2)
sd.lam ~ dunif(0, 3)
tau.site.p <- pow(sd.site.p, -2)
sd.site.p ~ dunif(0, 3)
tau.p <- pow(sd.p, -2)
sd.p ~ dunif(0, 3)

# Likelihood
# Ecological model for true abundance
for (i in 1:R){                                    # Loop over sites (95)
   for (k in 1:7){                                 # Loop over days (7)
      N[i,k] ~ dpois(lambda[i,k])                  # Abundance
      log(lambda[i,k]) <- alpha.lam[k] + eps.lam[i]

      # Observation model for replicated counts
      mu.lp[i,k] <- beta[k] + eps.p[i]
      for (j in 1:T){                              # Loop over reps (2)
         y[i,j,k] ~ dbin(p[i,j,k], N[i,k])         # Detection
         p[i,j,k] <- 1 / (1 + exp(-lp[i,j,k]))
         lp[i,j,k] ~ dnorm(mu.lp[i,k], tau.p) # random delta defined
implicitly
         # Assess model fit using Chi-squared discrepancy
         # Compute fit statistic for observed data
         eval[i,j,k] <- p[i,j,k] * N[i,k]
         E[i,j,k] <- pow((y[i,j,k] - eval[i,j,k]),2) / (eval[i,j,k]+0.5)
         # Generate replicate data and compute fit stats for them
         y.new[i,j,k] ~ dbin(p[i,j,k], N[i,k])
```

```
         E.new[i,j,k] <- pow((y.new[i,j,k] - eval[i,j,k]),2) /
(eval[i,j,k]+0.5)
         } #j
         ik.p[i,k] <- mean(p[i,,k])
      } #k
   } #i

# Derived and other quantities
for (k in 1:7){
   totalN[k] <- sum(N[,k])   # Estimate total pop. size across all sites
   mean.abundance[k] <- mean(lambda[,k])
   mean.N[k] <- mean(N[,k])
   mean.detection[k] <- mean(ik.p[,k])
   }
fit <- sum(E[,,])
fit.new <- sum(E.new[,,])
}
",fill = TRUE)
sink()

# Bundle data
R = nrow(y)
T = ncol(y)
win.data <- list(y = y, R = R, T = T)

# Initial values
Nst <- apply(y, c(1, 3), max) + 1
Nst[is.na(Nst)] <- 1
inits <- function(){list(N = Nst, alpha.lam = runif(7, -3, 3), beta =
runif(7, -3, 3), sd.lam = runif(1, 0, 1), sd.p = runif(1, 0, 1))}

# Parameters monitored
params <- c("totalN", "alpha.lam", "beta", "sd.lam", "sd.site.p", "sd.p",
"mean.abundance", "mean.N", "mean.detection", "fit", "fit.new")

# MCMC settings
ni <- 350000
nt <- 300
nb <- 50000
nc <- 3

# Call WinBUGS from R
out2B <- bugs(win.data, inits, params, "Nmix2B.txt", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = TRUE, bugs.directory =
bugs.dir, working.directory = getwd())
```

We evaluate the fit of the model.

```
# Evaluation of fit
plot(out2B$sims.list$fit, out2B$sims.list$fit.new, main = "", xlab =
"Discrepancy actual data", ylab = "Discrepancy replicate data", frame.plot
= FALSE, xlim = c(50, 200), ylim = c(50, 200))
abline(0, 1, lwd = 2, col = "black")
mean(out2B$sims.list$fit.new > out2B$sims.list$fit)
mean(out2B$mean$fit) / mean(out2B$mean$fit.new)
```

This model fits, which should come as no surprise, since the model in section 12.3.3. also fitted and model 2B is more complex still.

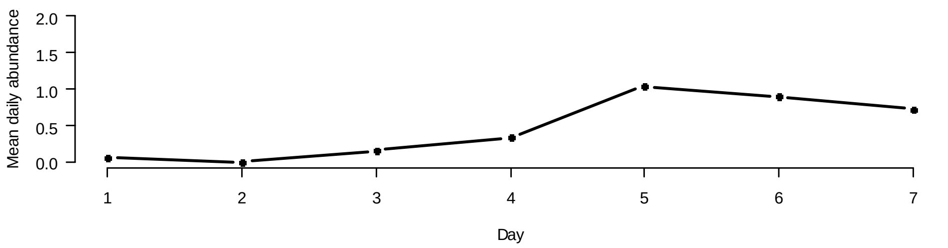
```
# Summarize posteriors
print(out2B, dig = 2)
```

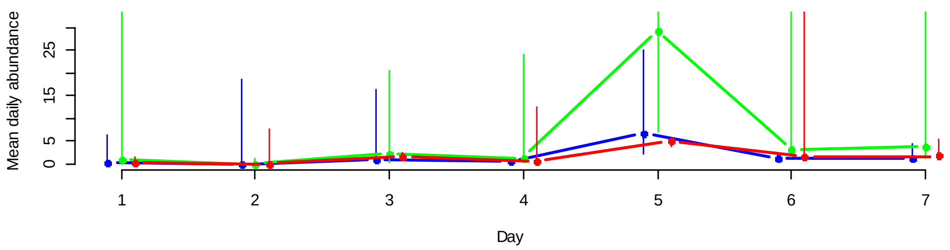
```
Inference for Bugs model at "Nmix2B.txt", fit using WinBUGS,
 3 chains, each with 350000 iterations (first 50000 discarded), n.thin = 300
 n.sims = 3000 iterations saved
                    mean     sd   2.5%    25%    50%     75%   97.5% Rhat n.eff
totalN[1]         117.99 239.55   5.00  13.00  32.00  110.00  734.02 1.05    54
totalN[2]          21.31  66.08   0.00   0.00   1.00    9.00  239.03 1.25    47
totalN[3]         275.88 358.53  20.00  61.00 151.00  350.25 1196.30 1.01   170
totalN[4]         100.03 123.47  36.00  48.00  65.00  105.00  357.00 1.02   150
totalN[5]        1147.20 1488.87 161.00 270.00 493.00 1326.00 5932.27 1.02   250
totalN[6]         186.44 125.35 100.00 122.00 148.00  201.00  529.07 1.03   230
totalN[7]         204.90 146.39  89.00 125.00 166.00  232.00  564.05 1.01   500
alpha.lam[1]       -2.17   1.44  -4.55  -3.26  -2.36   -1.17    0.72 1.05    59
alpha.lam[2]       -5.05   2.23  -9.39  -6.52  -5.16   -3.61   -0.53 1.02   130
alpha.lam[3]       -0.85   1.21  -3.05  -1.78  -0.82    0.06    1.37 1.01   280
alpha.lam[4]       -1.51   0.76  -2.66  -2.06  -1.64   -1.09    0.27 1.01   400
alpha.lam[5]        0.60   1.01  -0.86  -0.20   0.42    1.30    2.80 1.02   260
alpha.lam[6]       -0.73   0.59  -1.64  -1.13  -0.83   -0.43    0.69 1.01   220
alpha.lam[7]       -0.67   0.58  -1.65  -1.08  -0.71   -0.31    0.58 1.00   870
beta[1]            -3.36   1.87  -6.91  -4.70  -3.33   -2.04    0.11 1.04    61
beta[2]            -2.89   3.75  -9.21  -5.57  -3.26   -0.46    5.21 1.02   100
beta[3]            -3.41   1.73  -6.35  -4.68  -3.61   -2.26    0.29 1.02   210
beta[4]            -0.99   1.23  -3.81  -1.75  -0.83   -0.07    0.93 1.01   260
beta[5]            -2.41   1.27  -4.91  -3.35  -2.30   -1.40   -0.30 1.01   350
beta[6]            -0.58   0.94  -2.84  -1.11  -0.42    0.10    0.83 1.01   290
beta[7]            -1.29   0.82  -3.12  -1.82  -1.21   -0.70    0.08 1.01   840
sd.lam              1.75   0.28   1.19   1.58   1.75    1.92    2.30 1.00  3000
sd.site.p           0.94   0.59   0.05   0.45   0.89    1.36    2.20 1.02   410
sd.p                1.08   0.26   0.65   0.90   1.06    1.23    1.69 1.00  1000
mean.abundance[1]   1.24   2.52   0.04   0.14   0.34    1.16    7.67 1.05    56
mean.abundance[2]   0.23   0.69   0.00   0.01   0.02    0.10    2.45 1.03   130
mean.abundance[3]   2.91   3.78   0.20   0.65   1.59    3.67   12.37 1.01   180
mean.abundance[4]   1.06   1.30   0.34   0.51   0.70    1.12    3.74 1.02   160
mean.abundance[5]  12.07  15.67   1.68   2.84   5.13   14.00   62.83 1.02   250
mean.abundance[6]   1.96   1.32   1.00   1.29   1.57    2.09    5.58 1.03   260
mean.abundance[7]   2.15   1.55   0.90   1.31   1.74    2.44    5.94 1.01   550
mean.N[1]           1.24   2.52   0.05   0.14   0.34    1.16    7.73 1.05    54
mean.N[2]           0.22   0.70   0.00   0.00   0.01    0.09    2.52 1.25    47
mean.N[3]           2.90   3.77   0.21   0.64   1.59    3.69   12.59 1.01   170
mean.N[4]           1.05   1.30   0.38   0.51   0.68    1.10    3.76 1.02   150
mean.N[5]          12.08  15.67   1.70   2.84   5.19   13.96   62.44 1.02   250
mean.N[6]           1.96   1.32   1.05   1.28   1.56    2.12    5.57 1.03   230
mean.N[7]           2.16   1.54   0.94   1.32   1.75    2.44    5.94 1.01   500
mean.detection[1]   0.13   0.14   0.00   0.02   0.07    0.19    0.52 1.05    58
mean.detection[2]   0.25   0.32   0.00   0.01   0.08    0.41    0.99 1.02   100
mean.detection[3]   0.12   0.14   0.01   0.03   0.06    0.15    0.54 1.01   220
mean.detection[4]   0.36   0.16   0.08   0.23   0.35    0.48    0.67 1.01   230
mean.detection[5]   0.18   0.13   0.01   0.06   0.16    0.27    0.44 1.03   220
mean.detection[6]   0.41   0.14   0.13   0.32   0.42    0.52    0.65 1.03   170
mean.detection[7]   0.29   0.11   0.10   0.22   0.29    0.37    0.51 1.01   730
fit               121.70  20.33  84.43 107.77 121.10  134.60  163.51 1.00  1200
fit.new           122.49  20.78  83.98 107.87 122.10  136.22  163.31 1.00  1600
deviance          641.15  57.32 520.39 605.57 644.75  678.92  749.81 1.00   680
```

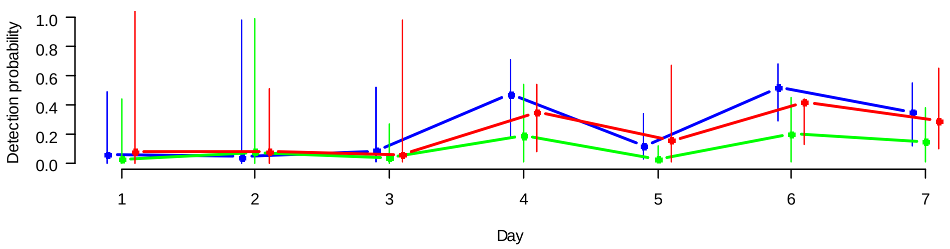
We can compare some of the estimates under the three models (the one in section 12.3.3. in the book (in blue) and those in this (green) and the previous exercise (red)) in a graph, analogous to Fig. 12-6 in the book. We plot posterior medians of mean abundance and detection probability and compare them with the observed mean counts.

```
max.day.count <- apply(y, c(1, 3), max, na.rm = TRUE)
max.day.count[max.day.count == "-Inf"] <- NA
mean.max.count <- apply(max.day.count, 2, mean, na.rm = TRUE)
mean.max.count

jit <- 0.1         # degree of translation of x-axis
par(mfrow = c(3, 1))
plot(1:7, mean.max.count, xlab = "Day", ylab = "Mean daily abundance", las
= 1, ylim = c(0, 2), type = "b", main = "", frame.plot = FALSE, pch = 16,
lwd = 2)
```

```
plot(1:7-jit, out2$summary[24:30,5], type = "b", pch = 16, col = "blue",
lwd = 2, xlab = "Day", ylab = "Mean daily abundance", ylim = c(0, 32), main
= "", frame.plot = FALSE)
segments(1:7-jit, out2$summary[24:30,3], 1:7-jit, out2$summary[24:30,7],
col = "blue")
lines(1:7, out2A$summary[24:30,5], type = "b", pch = 16, col = "green", lwd
= 2)
segments(1:7, out2A$summary[24:30,3], 1:7, out2A$summary[24:30,7], col =
"green")
lines(1:7+jit, out2B$summary[25:31,5], type = "b", pch = 16, col = "red",
lwd = 2)
segments(1:7+jit, out2B$summary[25:31,5], 1:7+jit, out2B$summary[24:30,7],
col = "red")

plot(1:7-jit, out2$summary[38:44,5], xlab = "Day", ylab = "Detection
probability ", las = 1, ylim = c(0, 1), type = "b", col = "blue", pch = 16,
frame.plot = FALSE, lwd = 2)
segments(1:7-jit, out2$summary[38:44,3], 1:7-jit, out2$summary[38:44,7],
col = "blue")
lines(1:7, out2A$summary[38:44,5], type = "b", col = "green", pch = 16, lwd
= 2)
segments(1:7, out2A$summary[38:44,3], 1:7, out2A$summary[38:44,7], col =
"green")
lines(1:7+jit, out2B$summary[39:45,5], type = "b", col = "red", pch = 16,
lwd = 2)
segments(1:7+jit, out2B$summary[39:45,3], 1:7+jit, out2B$summary[38:44,7],
col = "red")
```







We see that while the overall trends are quite comparable among the three models, the point estimates differ quite a bit in their magnitude and so do their uncertainty intervals. So care is needed when choosing a model for the variance terms. Unfortunately, this is not so straightforward when using a Bayesian analysis and may have to be done in an *ad hoc* way. Apart from subject matter considerations, the deviance of the model and the posterior distributions of the variance terms may assist in this part of model selection.

Here are the posterior summaries of the three models for the variance terms as well as for the deviance.

Model 2 (from section 12.3.3 in the book):

```
                   mean     sd   2.5%    25%    50%     75%   97.5% Rhat n.eff
sd.lam             1.87   0.23   1.46   1.70   1.84    2.01    2.37 1.00  1000
sd.p               1.05   0.21   0.70   0.91   1.03    1.17    1.50 1.00   980
deviance         640.44  49.86 540.00 607.37 641.30  674.42  737.21 1.01   250
```

Model 2A (exercise 2):

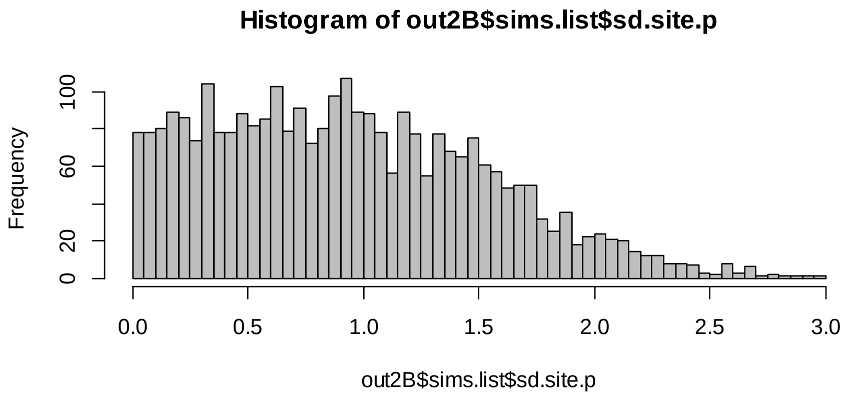
```
                   mean      sd   2.5%     25%     50%     75%    97.5% Rhat n.eff
sd.lam             1.91    0.25   1.48    1.74    1.90    2.07     2.44 1.00  3000
sd.p               1.02    0.14   0.76    0.91    1.01    1.11     1.32 1.00  1300
deviance         780.57   30.38 721.99  759.90  779.80  800.70   841.81 1.04    65
```

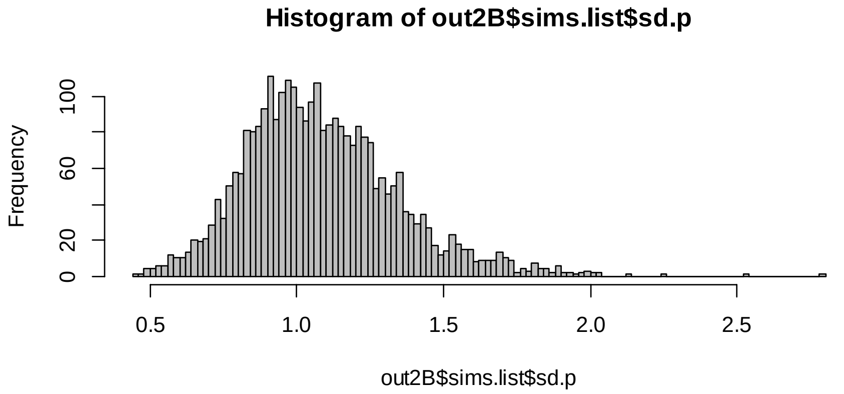
Model 2B (exercise 3):

```
                    mean      sd   2.5%    25%    50%     75%   97.5% Rhat n.eff
sd.lam              1.75    0.28   1.19   1.58   1.75    1.92    2.30 1.00  3000
sd.site.p           0.94    0.59   0.05   0.45   0.89    1.36    2.20 1.02   410
sd.p                1.08    0.26   0.65   0.90   1.06    1.23    1.69 1.00  1000
deviance          641.15   57.32 520.39 605.57 644.75  678.92  749.81 1.00   680
```

We see that the deviance for models 2 and 2B are much lower than that for model 2A. On the other hand, we see that the posterior distribution of the additional variance in model 2B has much more mass close to zero than the other variance contained in detection probability. This is confirmed by the following plot. (Actually, the posterior of sd.site.p looks so diffuse that we may doubt that it is really informed by the data, i.e. is estimable. Our intuition tells us that it should, but we would have to conduct a simulation exercise to confirm that hunch.)

```
par(mfrow = c(2, 1))
hist(out2B$sims.list$sd.site.p, breaks = 100, col = "gray")
hist(out2B$sims.list$sd.p, breaks = 100, col = "gray")
```





This, together with the fact that the additional variance component in detection in model 2B did not increase the amount of explained variation (i.e., did not decrease the deviance) might point out to model 2 as the one to base our inference on. However, this is all rather tentative.

Chapter 13

Exercise 1

Task: In the blue bug example, fit a 'behavioral response' effect, i.e., fit a separate detection probability dependent on whether the species has been detected ever before at a site or not. Hint, you can use the following R code to generate the 'seen-before' covariate matrix. How do you interpret the results? Would you use the behavioral response model for inference about the system behind the blue bug data set? Discuss.

```
# Generate a 'seen-before' covariate
sb <- array(NA, dim = dim(y))
for (i in 1:27){
   for (j in 1:6){
      sb[i,j] <- max(y[i, 1:(j-1)])
      }
   }
sb[is.na(y)] <- 0                  # Impute 'irrelevant' zeroes
```

Solution:

First of all, we have to **report an error** in the analysis published in section 13.4, although we don't quite understand it and why it happens. For some reason, the posterior distributions of the two occupancy parameters, `alpha.psi` and `beta.psi`, are pushed away from zero in opposite directions. This is not obvious from looking at the posterior summaries, but really is quite striking when looking well at the time-series plots. It happens regardless of whether uniform or flat normal priors are chosen for those parameters. We found that error when conducting a maximum likelihood analysis of the model for the data set for comparative reasons, therefore, below we give R code to use the function `occu()` in the package **unmarked** to fit the model using maximum likelihood.

We load **unmarked**, create the **unmarked** data frame and fit the model (note that in the model formula in `occu()`, detection comes first).

```
library(unmarked)
bugdata <- unmarkedFrameOccu(y = y, siteCovs = data.frame(edge = edge),
obsCovs = list(DATES = DATES, HOURS = HOURS))
fm <- occu(formula = ~ DATES + I(DATES^2) + HOURS + I(HOURS^2) ~
as.factor(edge)-1, data = bugdata)

Call:
occu(formula = ~DATES + I(DATES^2) + HOURS + I(HOURS^2) ~ as.factor(edge) -
    1, data = bugdata)

Occupancy:
                 Estimate    SE     z P(>|z|)
as.factor(edge)0    1.820 1.777  1.02   0.306
as.factor(edge)1   -0.813 0.734 -1.11   0.268

Detection:
            Estimate    SE      z P(>|z|)
(Intercept)    0.379 0.694  0.546  0.5853
DATES          0.325 0.375  0.864  0.3874
I(DATES^2)     0.150 0.460  0.325  0.7451
HOURS         -0.456 0.399 -1.143  0.2532
```

```
I(HOURS^2)    -0.547 0.305 -1.791  0.0732

AIC: 88.60092
```

We compare the ML and the Bayesian estimates.

```
ML.results <- rbind(summary(fm)$state[,1:2], summary(fm)$det[,1:2])
Bayesian.results <- out$summary[c(1:2, 5:9), c(1:3, 7)]
print(cbind(ML.results, Bayesian.results))
                   Estimate        SE       mean        sd         2.5%      97.5%
as.factor(edge)0  1.8197534 1.7771125  4.6133720 3.9096602   0.01018625 14.6705000
as.factor(edge)1 -0.8130025 0.7342859 -5.3860120 3.9168595 -15.24150000 -0.4387400
(Intercept)       0.3788880 0.6943580  0.3378263 0.6852934  -0.99546750  1.7241500
DATES             0.3245009 0.3754216  0.3455949 0.3989312  -0.42085000  1.1680000
I(DATES^2)        0.1496100 0.4601880  0.1854664 0.4710434  -0.72255000  1.1011000
HOURS            -0.4556358 0.3987614 -0.4970859 0.4142280  -1.37102500  0.2540675
I(HOURS^2)       -0.5468332 0.3052399 -0.6012856 0.3275583  -1.23932500  0.0036495
```

We can also compare these estimates to a tabulation of the observed data.

```
# Check observed occurrence at forest interior (edge=0) and at edge sites
(edge=1)
tapply(apply(y, 1, max, na.rm = TRUE), edge, mean)
        0         1
0.5000000 0.2307692
```

We see that the estimates for detection match pretty well, but that the occupancy parameter estimates don't. We don't know why, but it is clear to us that the Bayesian estimates that we obtain in this analysis are faulty in some way. In contrast, in view of the observed data, the MLEs look more sensible, since `expit(1.8197534) = 0.86` and `expit(-0.8130025) = 0.31`.

Penultimatively, to see whether another MCMC engine, JAGS, can do better, we also fitted the model in JAGS.

```
# Call JAGS from R and get run time
library("R2jags")       # requires rjags
outJAGS <- jags(win.data, inits, params, "model.txt", n.chains = nc, n.thin
= nt, n.iter = ni, n.burnin = nb)

Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 1300

Initializing model

  |
  |++++++++++++++++++++++++++++++++++++++++++++++++++| 100%
  |**************************************************| 100%


print(outJAGS, dig = 3)
Inference for Bugs model at "model.txt", fit using jags,
 3 chains, each with 30000 iterations (first 20000 discarded), n.thin = 10
 n.sims = 3000 iterations saved
          mu.vect sd.vect    2.5%    25%    50%    75%  97.5%  Rhat n.eff
alpha.p     0.336   0.687  -0.975 -0.128  0.329  0.787  1.707 1.002  1800
alpha.psi   5.167   4.274   0.023  1.771  3.938  7.842 15.236 1.014   220
beta.psi   -5.966   4.291 -16.056 -8.466 -4.810 -2.653 -0.551 1.014   210
```

```
beta1.p     0.344   0.384  -0.380  0.086  0.348  0.594  1.111 1.001  3000
beta2.p     0.184   0.474  -0.738 -0.135  0.186  0.499  1.122 1.002  1900
beta3.p    -0.490   0.414  -1.354 -0.755 -0.468 -0.209  0.271 1.001  3000
beta4.p    -0.593   0.324  -1.232 -0.803 -0.583 -0.372  0.014 1.002  1600
mean.p      0.575   0.152   0.274  0.468  0.582  0.687  0.846 1.001  2000
occ.fs     17.016   2.356  11.975 16.000 17.000 18.000 21.000 1.004   800
deviance   67.798   5.901  55.843 64.283 67.933 71.466 79.640 1.001  2200
```

These estimates are pretty similar to the ones from WinBUGS.

And finally, we tried to reparameterize the forest interior/edge factor from an effects parameterisation (in the book) to a means parameterisation.

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

# Priors
alpha.in ~ dnorm(0, 0.01)
alpha.edge ~ dnorm(0, 0.01)
alpha.p ~ dnorm(0, 0.01)
beta1.p ~ dnorm(0, 0.01)
beta2.p ~ dnorm(0, 0.01)
beta3.p ~ dnorm(0, 0.01)
beta4.p ~ dnorm(0, 0.01)

# Likelihood
# Ecological model for the partially observed true state
for (i in 1:R){
   z[i] ~ dbern(psi[i])                # True occurrence z at site i
   psi[i] <- 1 / (1 + exp(-lpsi.lim[i]))
   lpsi.lim[i] <- min(999, max(-999, lpsi[i]))
   lpsi[i] <- alpha.in * (1-edge[i]) + alpha.edge * edge[i]

   # Observation model for the observations
   for (j in 1:T){
      y[i,j] ~ dbern(mu.p[i,j])    # Detection-nondetection at i and j
      mu.p[i,j] <- z[i] * p[i,j]
      p[i,j] <- 1 / (1 + exp(-lp.lim[i,j]))
      lp.lim[i,j] <- min(999, max(-999, lp[i,j]))
      lp[i,j] <- alpha.p + beta1.p * DATES[i,j] + beta2.p * pow(DATES[i,j],
2) + beta3.p * HOURS[i,j] + beta4.p * pow(HOURS[i,j], 2)
      } #j
   } #i

# Derived quantities
occ.fs <- sum(z[])                             # Number of occupied sites
mean.p <- exp(alpha.p) / (1 + exp(alpha.p))    # Sort of average detection
}
",fill = TRUE)
sink()

# Bundle data
win.data <- list(y = y, R = nrow(y), T = ncol(y), edge = edge, DATES =
DATES, HOURS = HOURS)

# Initial values
zst <- apply(y, 1, max, na.rm = TRUE)     # Good starting values crucial
inits <- function(){list(z = zst, alpha.p = runif(1, -3, 3))}

# Parameters monitored
```

```
params <- c("alpha.in", "alpha.edge", "mean.p", "occ.fs", "alpha.p",
"beta1.p", "beta2.p", "beta3.p", "beta4.p")

# MCMC settings
ni <- 30000
nt <- 10
nb <- 20000
nc <- 3

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

print(out, 2)
Inference for Bugs model at "model.txt", fit using WinBUGS,
 3 chains, each with 30000 iterations (first 20000 discarded), n.thin = 10
 n.sims = 3000 iterations saved
            mean    sd  2.5%   25%   50%   75% 97.5% Rhat n.eff
alpha.in   24.61 18.94  0.94  8.98 21.21 36.24 68.25 1.00  1300
alpha.edge -0.82  0.81 -2.50 -1.34 -0.81 -0.29  0.75 1.00  1600
mean.p      0.55  0.14  0.26  0.45  0.55  0.65  0.81 1.01   430
occ.fs     17.98  1.71 14.00 17.00 18.00 19.00 22.00 1.02   330
alpha.p     0.21  0.63 -1.03 -0.21  0.20  0.63  1.46 1.01   450
beta1.p     0.34  0.37 -0.35  0.09  0.33  0.58  1.06 1.00  1600
beta2.p     0.21  0.46 -0.69 -0.10  0.20  0.53  1.16 1.00   930
beta3.p    -0.43  0.40 -1.25 -0.69 -0.42 -0.16  0.31 1.00  1300
beta4.p    -0.57  0.31 -1.22 -0.78 -0.56 -0.36 -0.02 1.00   460
deviance   69.88  4.82 60.58 66.80 69.38 72.54 80.68 1.01   570
```

Interestingly, this helps for one of the parameters (`alpha.edge`), whose posterior mean now corresponds to its MLE, but not for `alpha.in`.

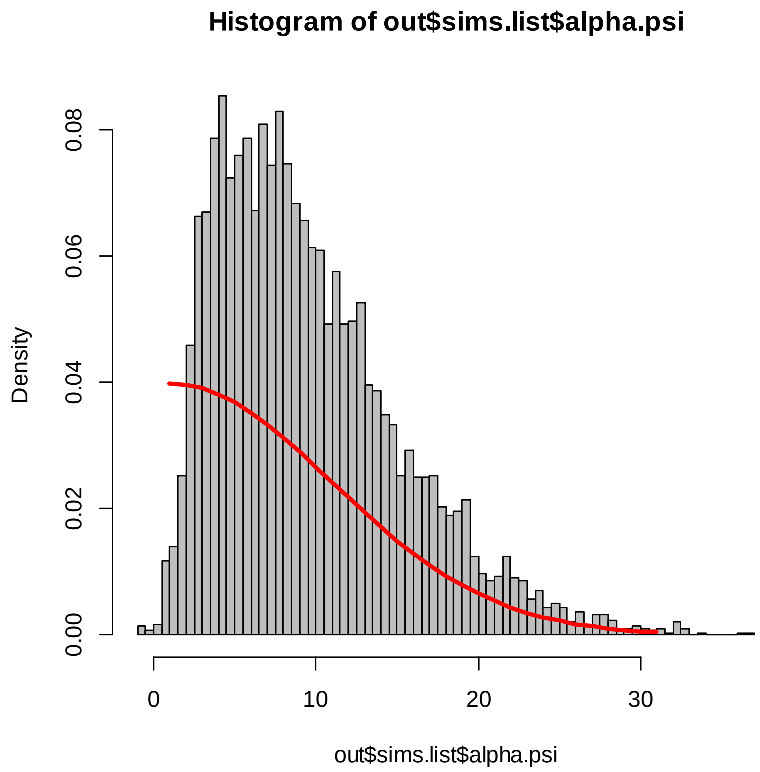
For now, we will solve exercise 1 with a smaller model, where we don't distinguish between occupancy at sites within the forest and at the forest edge. In the following, we assume that you have loaded the data and prepared the response and covariate data. Then, to fit a permanent trap response, we simply fit the `sb` covariate (which is a 'survey' or 'sampling covariate').

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

# Priors
alpha.psi ~ dnorm(0, 0.01)
# beta.psi ~ dnorm(0, 0.01) # Drop that term
alpha.p ~ dnorm(0, 0.01)
beta1.p ~ dnorm(0, 0.01)
beta2.p ~ dnorm(0, 0.01)
beta3.p ~ dnorm(0, 0.01)
beta4.p ~ dnorm(0, 0.01)
beta5.p ~ dnorm(0, 0.01)

# Likelihood
# Ecological model for the partially observed true state
for (i in 1:R){
   z[i] ~ dbern(psi[i])                # True occurrence z at site i
   psi[i] <- 1 / (1 + exp(-lpsi.lim[i]))
```

```
    lpsi.lim[i] <- min(999, max(-999, lpsi[i]))
    lpsi[i] <- alpha.psi ### + beta.psi * edge[i] # Drop beta.psi

    # Observation model for the observations
    for (j in 1:T){
       y[i,j] ~ dbern(mu.p[i,j])     # Detection-nondetection at i and j
       mu.p[i,j] <- z[i] * p[i,j]
       p[i,j] <- 1 / (1 + exp(-lp.lim[i,j]))
       lp.lim[i,j] <- min(999, max(-999, lp[i,j]))
       lp[i,j] <- alpha.p + beta1.p*DATES[i,j] + beta2.p*pow(DATES[i,j], 2)
+ beta3.p*HOURS[i,j] + beta4.p*pow(HOURS[i,j], 2) + beta5.p*sb[i,j]
       } #j
    } #i

# Derived quantities
occ.fs <- sum(z[])                               # Number of occupied sites
mean.p <- exp(alpha.p) / (1 + exp(alpha.p))     # Average first detection
}
",fill = TRUE)
sink()

# Bundle data
win.data <- list(y = y, R = nrow(y), T = ncol(y), DATES = DATES, HOURS =
HOURS, sb = sb)

# Initial values
zst <- apply(y, 1, max, na.rm = TRUE)     # Good starting values crucial
inits <- function(){list(z = zst, alpha.psi=runif(1, -3, 3), alpha.p =
runif(1, -3, 3))}

# Parameters monitored
params <- c("alpha.psi", "mean.p", "occ.fs", "alpha.p", "beta1.p",
"beta2.p", "beta3.p", "beta4.p", "beta5.p")

# MCMC settings
ni <- 25000
nt <- 10
nb <- 5000
nc <- 3

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

# Summarize posteriors
print(out, dig = 2)
Inference for Bugs model at "model.txt", fit using WinBUGS,
 3 chains, each with 25000 iterations (first 5000 discarded), n.thin = 10
 n.sims = 6000 iterations saved
           mean   sd  2.5%   25%   50%   75% 97.5% Rhat n.eff
alpha.psi  9.73 5.61  2.15  5.40  8.72 13.00 22.68 1.00  5100
mean.p     0.05 0.05  0.00  0.01  0.03  0.06  0.19 1.00  1100
occ.fs    26.78 0.81 25.00 27.00 27.00 27.00 27.00 1.01  6000
alpha.p   -3.48 1.16 -5.88 -4.25 -3.39 -2.68 -1.47 1.00  1100
beta1.p   -0.16 0.46 -1.09 -0.48 -0.16  0.16  0.74 1.00  1500
beta2.p    0.11 0.55 -0.94 -0.26  0.11  0.47  1.20 1.00   990
beta3.p   -0.74 0.44 -1.66 -1.03 -0.73 -0.44  0.08 1.00  1400
beta4.p   -0.33 0.36 -1.05 -0.56 -0.32 -0.09  0.34 1.00  1600
beta5.p    4.55 1.02  2.76  3.83  4.48  5.18  6.75 1.00  2800
deviance  61.61 3.54 56.69 59.05 60.92 63.55 70.12 1.00  6000
```

Now we estimate that all 27 sites are occupied. From our biological intuition and knowing the bug and the study area, this does not look like a sensible result. We see that the occupancy probability on the logit scale (alpha.psi) is estimated at an extremely high value. But then, we also note that its credible interval is huge, going from essentially 2 to 23. This looks like the posterior could reflect essentially its prior, i.e., that it is not estimable from the data. Therefore, we plot the posterior of alpha.psi and compare it with its prior (a normal distribution with mean zero and variance 100).

```
# Plot posterior and prior for alpha.psi in same graph
hist(out$sims.list$alpha.psi, breaks = 100, col = "grey", freq = FALSE)
lines(dnorm(0:30, mean = 0, sd = sqrt(1 / (0.01))), col = "red", lwd =3)
```



This plot confirms our suspicion. It seems that the posterior is only little informed by the data. For comparison, here is the same analysis using the R package **unmarked**.

```
library(unmarked)
bugdata <- unmarkedFrameOccu(y = y, siteCovs = data.frame(edge = edge),
obsCovs = list(DATES = DATES, HOURS = HOURS, sb = sb))
summary(bugdata)
summary(fm <- occu(formula = ~ DATES + I(DATES^2) + HOURS + I(HOURS^2) + sb
~ 1, data = bugdata))


Call:
occu(formula = ~DATES + I(DATES^2) + HOURS + I(HOURS^2) + sb ~
    1, data = bugdata)

Occupancy (logit-scale):
 Estimate  SE      z P(>|z|)
     12.4 170 0.0727   0.942

Detection (logit-scale):
            Estimate    SE      z  P(>|z|)
```

```
(Intercept)   -3.120 1.054 -2.959 3.09e-03
DATES         -0.162 0.434 -0.374 7.08e-01
I(DATES^2)     0.115 0.516  0.224 8.23e-01
HOURS         -0.643 0.412 -1.562 1.18e-01
I(HOURS^2)    -0.255 0.320 -0.797 4.26e-01
sb             4.010 0.926  4.331 1.48e-05

AIC: 69.40862
Number of sites: 27
optim convergence code: 0
optim iterations: 52
Bootstrap iterations: 0
```

The standard error of the occupancy intercept is huge. In ML analyses, this often is an indication that a parameter is not estimable or that there are some numerical difficulties with its estimation. We repeat the Bayesian analysis with a different prior for alpha.psi, a normal with variance 1000 and a uniform on the range (-10, 10), i.e., do a prior sensitivity analysis. Here are the summaries for its posterior.

Normal(0, 1000) prior for alpha.psi:

```
           mean    sd  2.5%   25%   50%   75% 97.5% Rhat n.eff
alpha.psi 27.33 18.49  3.45 12.52 23.58 38.60 71.93 1.00  1400
```

Uniform(-100, 100) prior for alpha.psi:

```
           mean    sd  2.5%   25%   50%   75% 97.5% Rhat n.eff
alpha.psi 50.53 28.07  4.96 26.51 50.58 74.26 97.38 1.00  6000
```

Clearly, the estimate of alpha.psi is very strongly affected by our choice of prior, i.e., there is extreme prior sensitivity of the posterior of alpha.psi. Together with the observation of a huge SE in the ML analysis of this model, this suggests to us that there are estimability problems in this model. Of course, this happens regardless of the method chosen for the analysis of the model; both the Bayesian and the ML estimates are similarly affected by this intrinsic deficiency of this model for this data set.

For curiosity, we check a model with only the sb effect using ML analysis:

```
summary(fm <- occu(formula = ~ sb ~ 1, data = bugdata))

Call:
occu(formula = ~sb ~ 1, data = bugdata)

Occupancy (logit-scale):
 Estimate SE     z P(>|z|)
     9.76 46 0.212   0.832

Detection (logit-scale):
            Estimate    SE     z  P(>|z|)
(Intercept)    -3.02 0.724 -4.17 3.03e-05
sb              3.58 0.810  4.42 9.77e-06

AIC: 65.44045
Number of sites: 27
optim convergence code: 0
optim iterations: 50
Bootstrap iterations: 0
```

Look at the huge SE of the occupancy parameter estimate; this model is not fine either. And what about a model with a single occupancy intercept and without the sb covariate in detection?

```
summary(fm <- occu(formula = ~ ~ DATES + I(DATES^2) + HOURS + I(HOURS^2) ~
1, data = bugdata))

Call:
occu(formula = ~~DATES + I(DATES^2) + HOURS + I(HOURS^2) ~ 1,
    data = bugdata)

Occupancy (logit-scale):
 Estimate    SE     z P(>|z|)
    0.105 0.564 0.187   0.852

Detection (logit-scale):
            Estimate    SE      z P(>|z|)
(Intercept)    0.439 0.679  0.647   0.518
DATES          0.279 0.405  0.690   0.490
I(DATES^2)     0.146 0.468  0.313   0.754
HOURS         -0.535 0.404 -1.324   0.186
I(HOURS^2)    -0.516 0.319 -1.618   0.106

AIC: 90.87979
Number of sites: 27
optim convergence code: 0
optim iterations: 31
Bootstrap iterations: 0
```

This looks fine. We compare this with the Bayesian analysis of the same model (code not shown, but by now this modification of the model should be trivial for you)

```
Inference for Bugs model at "model.txt", fit using WinBUGS,
 3 chains, each with 25000 iterations (first 5000 discarded), n.thin = 10
 n.sims = 6000 iterations saved
           mean   sd  2.5%   25%   50%   75% 97.5% Rhat n.eff
alpha.psi  0.23 1.26 -0.94 -0.28  0.10  0.51  1.67 1.08   490
mean.p     0.60 0.15  0.29  0.50  0.61  0.72  0.87 1.00  6000
occ.fs    14.37 3.07 10.00 12.00 14.00 16.00 21.00 1.00  1200
alpha.p    0.47 0.71 -0.92 -0.01  0.46  0.95  1.88 1.00  5400
beta1.p    0.27 0.45 -0.64 -0.03  0.27  0.57  1.16 1.00  5200
beta2.p    0.19 0.50 -0.78 -0.15  0.18  0.52  1.22 1.00  2300
beta3.p   -0.65 0.45 -1.58 -0.94 -0.63 -0.34  0.17 1.00  4100
beta4.p   -0.55 0.37 -1.29 -0.80 -0.54 -0.30  0.17 1.00  1400
deviance  63.48 7.61 52.27 58.29 62.41 67.18 81.98 1.00  1200
```

Given the small sample size, the match between these results is about what we would expect.

So what have we learned from this? On our side, perhaps that we should actually try out all the exercises before publishing a book ! And both you and us have been reminded of the fact that the analysis of an actual ecological data set, which is often small and may have limited information about the parameters we want to estimate, can be a challenge. What the analyses conducted seem to indicate to us is this:

- Inexplicably, WinBUGS (and JAGS as well) has problems with a model with different occupancy estimates for forest interior and forest edge sites, while a maximum likelihood analysis in **unmarked** doesn't. Hence, our Bayesian occupancy estimates should not be trusted. Interestingly, the detection estimates don't seem to be affected.

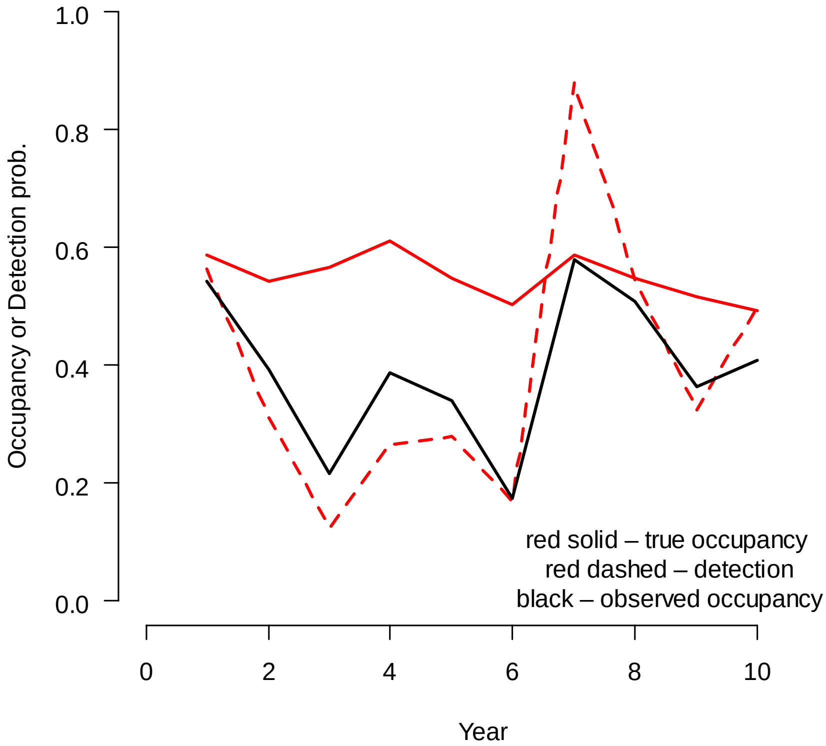
- By fitting a reparameterized model (choosing a means instead of an effects parameterization), we get a posterior mean for forest edge sites that matchens its MLE. However, the posterior distribution for occupancy at the forest interior is still bouncing away from 0 and reaching to very large values. By the way, this is an illustration of the fact that success in an MCMC analysis can depend vitally on the parameterization of a model chosen.
- A model with a single occupancy parameter and with a permanent trap response is not estimable. This means that the data set do not contain the information to estimate all of its parameters and has nothing to do with the choice of a Bayesian or a maximum likelihood analysis.
- Looking at strange features in the trace plots of the Markov chains, comparing posteriors and priors, doing prior sensitivity analyses, switching between different parameterizations of a model and comparing Bayesian and ML estimates can all be vital for a proper analysis of an ecological data set.
- To investigate further the question of estimability, we could try to adapt formal methods such as the ones described by Catchpole and Morgan (1997) and Catchpole et al. (2001) to hierarchical models such as the occupancy model. A less elegant and safe approach, which is nevertheless much more accessible to an ecologist, would consist of simulating data sets with known parameters values, and for the actual sample size of our data set (i.e., number of sites and number of surveys per site) and seeing whether we are able to get paramater estimates that resemble the input values over a large number of replicates or else for some single, very large copy of our data set (which perhaps should have data from many sites, but preserve the patterns of occasion frequency).

<u>Exercise 2</u>

Task: In the dynamic occupancy model of Section 13.5.1, ignore the detection process and aggregate the temporal within-day replicates. Adapt the WinBUGS code to fit a conventional metapopulation model and see how the estimated quantities are biased; see also Ruiz-Gutiérrez and Zipkin (2011).

Solution: We first use the function on p. 370 of the book to generate one data set and then attach it.

```
data <- data.fn(R = 250, J = 3, K = 10, psi1 = 0.6, range.p = c(0.1, 0.9),
range.phi = c(0.7, 0.9), range.gamma = c(0.1, 0.5))
attach(data)
```



We chose a nice replicate of the stochastic process which illustrates neatly how the dynamics of the observed occupancy may sometimes reflect mostly the dynamics of detection probability.

Next, we aggregate the 3D data to two dimensions (representing site and season) and then bundle the data. We call `zobs` the observed occurrence state variable.

```
zobs <- apply(y, c(1, 3), max)     # Observed occurrence as inits for z
win.data <- list(y = zobs, nsite = dim(zobs)[1], nyear = dim(zobs)[2])
```

Then, in the BUGS model code we get rid of the observation model. Since we now model (the observed) *z* directly, we replace by *y* each of the original *z*'s. We need no longer estimate n.occ, since the finite sample number of occupied sites is simply the observed number of occupied sites.

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

# Specify priors
psi1 ~ dunif(0, 1)
for (k in 1:(nyear-1)){
   phi[k] ~ dunif(0, 1)
   gamma[k] ~ dunif(0, 1)
   }

# Combined model: No separation of state and observation processes
for (i in 1:nsite){
   y[i,1] ~ dbern(psi1)
```

```
   for (k in 2:nyear){
      muZ[i,k]<- y[i,k-1]*phi[k-1] + (1-y[i,k-1])*gamma[k-1]
      y[i,k] ~ dbern(muZ[i,k])
      } #k
   } #i

# Derived parameters: Population occupancy, growth rate and turnover
psi[1] <- psi1
for (k in 2:nyear){
   psi[k] <- psi[k-1]*phi[k-1] + (1-psi[k-1])*gamma[k-1]
   growthr[k] <- psi[k]/psi[k-1]
   turnover[k-1] <- (1 - psi[k-1]) * gamma[k-1]/psi[k]
   }
}
",fill = TRUE)
sink()
```

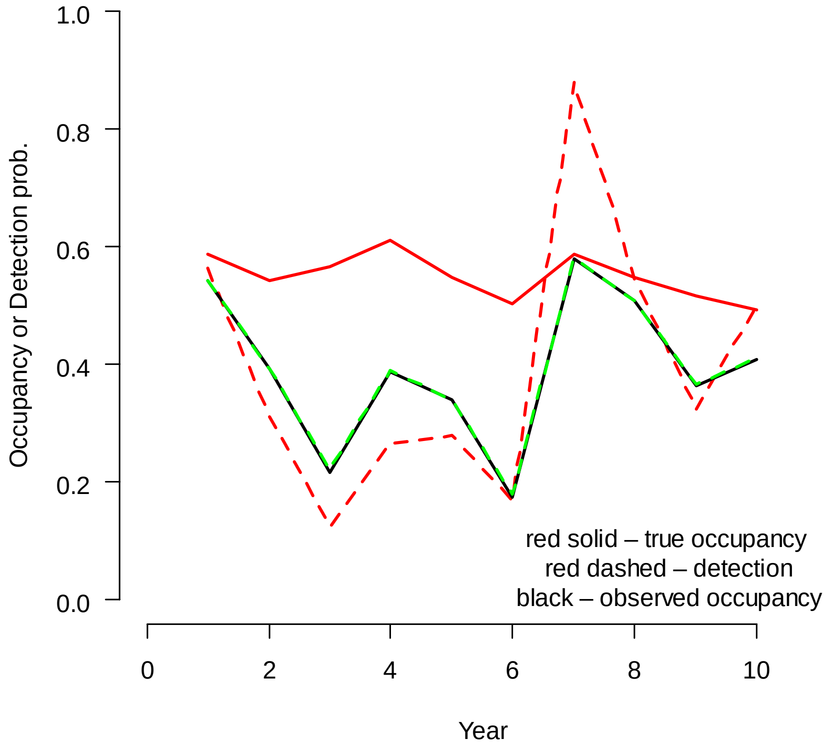
We must give at least one initial value when running WinBUGS from R via the function `bugs()`. We can no longer give *y* (which was *z* in the original code), so we give an initial value for psi1 instead.

```
# Initial values
inits <- function(){ list(psi1 = runif(1, 0, 1))}

# Parameters monitored
params <- c("psi", "phi", "gamma", "growthr", "turnover")

# MCMC settings
ni <- 1100   ;   nt <- 1   ;   nb <- 100   ;   nc <- 3

# Call WinBUGS from R
out <- bugs(win.data, inits, params, "Naive.Dynocc.txt", n.chains = nc,
n.thin = nt, n.iter = ni, n.burnin = nb, debug = FALSE, bugs.directory =
bugs.dir, working.directory = getwd())

# Summarize posteriors
print(out, dig = 2)
Inference for Bugs model at "Naive.Dynocc.txt", fit using WinBUGS,
 3 chains, each with 1100 iterations (first 100 discarded)
 n.sims = 3000 iterations saved
               mean   sd   2.5%    25%    50%    75%  97.5% Rhat n.eff
psi[1]         0.54 0.03   0.48   0.52   0.54   0.57   0.60    1  3000
psi[2]         0.39 0.03   0.33   0.37   0.39   0.41   0.46    1  3000
psi[3]         0.22 0.03   0.17   0.20   0.22   0.24   0.28    1  3000
psi[4]         0.39 0.03   0.33   0.37   0.39   0.41   0.45    1  3000
psi[5]         0.34 0.03   0.29   0.32   0.34   0.36   0.40    1  3000
psi[6]         0.18 0.02   0.14   0.16   0.18   0.20   0.23    1  3000
psi[7]         0.58 0.03   0.52   0.56   0.58   0.60   0.64    1  3000
psi[8]         0.51 0.03   0.44   0.49   0.51   0.53   0.57    1  3000
psi[9]         0.37 0.03   0.31   0.35   0.37   0.39   0.42    1  1800
psi[10]        0.41 0.03   0.35   0.39   0.41   0.43   0.47    1  1400
phi[1]         0.56 0.04   0.47   0.53   0.56   0.59   0.65    1  3000
phi[2]         0.27 0.04   0.19   0.24   0.27   0.30   0.36    1  3000
phi[3]         0.39 0.07   0.27   0.35   0.39   0.44   0.52    1  2200
phi[4]         0.46 0.05   0.37   0.43   0.46   0.50   0.56    1  3000
phi[5]         0.31 0.05   0.22   0.28   0.31   0.34   0.41    1  3000
phi[6]         0.72 0.06   0.58   0.68   0.72   0.76   0.84    1  3000
phi[7]         0.77 0.03   0.70   0.74   0.77   0.79   0.83    1  2600
phi[8]         0.55 0.04   0.46   0.52   0.55   0.58   0.64    1  2400
phi[9]         0.69 0.05   0.59   0.66   0.69   0.72   0.78    1  3000
gamma[1]       0.20 0.04   0.13   0.17   0.20   0.22   0.28    1  2200
```

```
gamma[2]       0.19 0.03    0.13    0.17    0.19    0.21    0.26    1  3000
gamma[3]       0.39 0.03    0.32    0.37    0.39    0.41    0.46    1  3000
gamma[4]       0.26 0.03    0.20    0.24    0.26    0.29    0.33    1  3000
gamma[5]       0.11 0.02    0.07    0.10    0.11    0.13    0.16    1  3000
gamma[6]       0.55 0.04    0.48    0.52    0.55    0.57    0.62    1  3000
gamma[7]       0.15 0.03    0.09    0.13    0.15    0.17    0.22    1  3000
gamma[8]       0.18 0.03    0.12    0.15    0.17    0.20    0.25    1  3000
gamma[9]       0.25 0.03    0.19    0.23    0.25    0.27    0.32    1  1400
growthr[2]     0.73 0.06    0.61    0.69    0.73    0.76    0.85    1  3000
growthr[3]     0.56 0.08    0.43    0.51    0.56    0.61    0.73    1  3000
growthr[4]     1.79 0.26    1.34    1.61    1.77    1.95    2.35    1  3000
growthr[5]     0.88 0.09    0.71    0.82    0.88    0.94    1.07    1  3000
growthr[6]     0.53 0.08    0.39    0.48    0.53    0.58    0.69    1  3000
growthr[7]     3.26 0.45    2.50    2.93    3.22    3.53    4.24    1  3000
growthr[8]     0.88 0.05    0.79    0.85    0.88    0.91    0.97    1  1300
growthr[9]     0.72 0.06    0.61    0.68    0.72    0.76    0.84    1  3000
growthr[10]    1.13 0.10    0.95    1.06    1.12    1.19    1.33    1  1900
turnover[1]    0.23 0.04    0.15    0.20    0.23    0.26    0.32    1  2200
turnover[2]    0.52 0.07    0.39    0.47    0.52    0.56    0.64    1  3000
turnover[3]    0.78 0.04    0.69    0.75    0.78    0.81    0.85    1  3000
turnover[4]    0.47 0.05    0.37    0.43    0.47    0.51    0.57    1  3000
turnover[5]    0.41 0.07    0.27    0.37    0.41    0.46    0.56    1  3000
turnover[6]    0.78 0.03    0.70    0.75    0.78    0.80    0.84    1  3000
turnover[7]    0.12 0.03    0.07    0.10    0.12    0.14    0.19    1  3000
turnover[8]    0.24 0.04    0.16    0.21    0.23    0.26    0.33    1  3000
turnover[9]    0.39 0.05    0.30    0.35    0.39    0.42    0.48    1  2300
deviance    2936.16 6.03 2926.00 2932.00 2935.00 2940.00 2950.00    1   740
```

We now add the occupancy estimates from the naïve occupancy model in the graph produced by the data-generation procedure. For this, the graph must still be active.

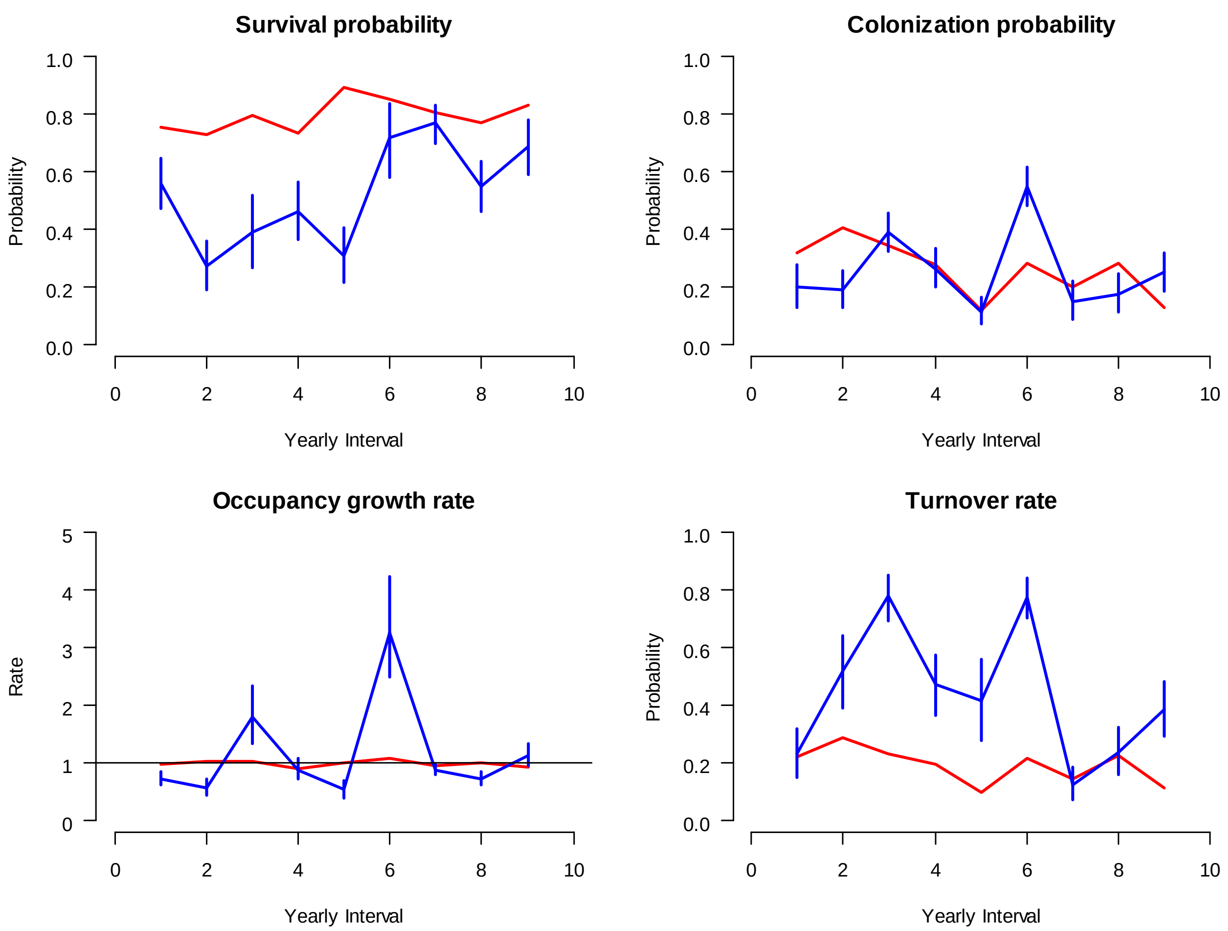
```
lines(1:data$K, out$mean$psi, type = "l", col = "green", lwd = 2, lty =
"dashed")
```



Of course, the estimated occupancy is exactly the observed proportion of sites with at least one detection in a year. We now plot the truth and the estimates under the naïve model to see how neglecting imperfect detection probability can bias all estimators from this traditional metapopulation model. We must first compute the true growth and turnover rates from the values of phi, gamma and phi in the simulated data object.

```
TO <- GR <- array(dim = 9)
for (k in 2:data$K){
   GR[k-1] <- data$psi[k] / data$psi[k-1]
   TO[k-1] <- (1 - data$psi[k-1]) * data$gamma[k-1] / data$psi[k]
   }

# Compare estimates of dynamic parameters with their true values
par(mfrow = c(2, 2), mar = c(5, 5, 2, 1))

# Survival probability
plot(1:(data$K-1), data$phi, type = "l", xlab = "Yearly Interval", ylab =
"Probability", col = "red", xlim = c(0, data$K), ylim = c(0, 1), lwd = 2,
lty = 1, frame.plot = FALSE, las = 1, main = "Survival probability")
lines(1:(data$K-1), out$mean$phi, type = "l", col = "blue", lwd = 2, lty =
1)
segments(1:(data$K-1), out$summary[11:19,3], 1:(data$K-1),
out$summary[11:19,7], lwd = 2, col = "blue")
# legend(4, 0.15, c('Truth', 'Estimate naïve occupancy model'),
col=c("red", "blue"), lty = c(1, 1), lwd = 2, cex = 0.8)

# Colonization probability
plot(1:(data$K-1), data$gamma, type = "l", xlab = "Yearly Interval", ylab =
"Probability", col = "red", xlim = c(0, data$K), ylim = c(0, 1), lwd = 2,
lty = 1, frame.plot = FALSE, las = 1, main = "Colonization probability")
```

```
lines(1:(data$K-1), out$mean$gamma, type = "l", col = "blue", lwd = 2, lty
= 1)
segments(1:(data$K-1), out$summary[20:28,3], 1:(data$K-1),
out$summary[20:28,7], lwd = 2, col = "blue")
# legend(0, 0.9, c('Truth', 'Estimate naïve occupancy model'), col=c("red",
"blue"), lty = c(1, 1), lwd = 2, cex = 0.8)

# Growth rate
plot(1:(data$K-1), GR, type = "l", xlab = "Yearly Interval", ylab = "Rate",
col = "red", xlim = c(0, data$K), ylim = c(0, 5), lwd = 2, lty = 1,
frame.plot = FALSE, las = 1, main = "Occupancy growth rate")
lines(1:(data$K-1), out$mean$growthr, type = "l", col = "blue", lwd = 2,
lty = 1)
segments(1:(data$K-1), out$summary[29:37,3], 1:(data$K-1),
out$summary[29:37,7], lwd = 2, col = "blue")
abline(h = 1, lwd = 1, col = "black")
# legend(0, 5, c('Truth', 'Estimate naïve occupancy model'), col=c("red",
"blue"), lty = c(1, 1), lwd = 2, cex = 0.8)

# Turnover rate
plot(1:(data$K-1), TO, type = "l", xlab = "Yearly Interval", ylab =
"Probability", col = "red", xlim = c(0, data$K), ylim = c(0, 1), lwd = 2,
lty = 1, frame.plot = FALSE, las = 1, main = "Turnover rate")
lines(1:(data$K-1), out$mean$turnover, type = "l", col = "blue", lwd = 2,
lty = 1)
segments(1:(data$K-1), out$summary[38:46,3], 1:(data$K-1),
out$summary[38:46,7], lwd = 2, col = "blue")
# legend(0, 1, c('Truth', 'Estimate naïve occupancy model'), col=c("red",
"blue"), lty = c(1, 1), lwd = 2, cex = 0.8)
```



In this plot, the truth is shown in red and the estimates under a traditional metapopulation model (i.e., a dynamic occupancy model without the observation submodel) are shown in blue, along with their 95% credible intervals. The plots shows well that ignoring detection probability in a traditional metapopulation model will always lead to negative bias in the occupancy estimator, if detection probability is less than 1. The biases in other estimands, such as colonization probability, the occupancy-based growth rate or turnover rate, can be substantial, but their sign cannot be predicted, i.e., overestimates and underestimates both occur. Note that the bias in turnover rate has also been called 'pseudo-turnover' in the literature (e.g., Fischer and Stöcklin, *Conservation Biology*, 1997)

Exercise 3

Task: Fit a multi-season, non-dynamic version of the site-occupancy model to the burnet data. That is, treat days as a group and model occupancy independent between successive days (similar to how we modeled abundance in section 12.3). In this way, you commit some pseudoreplication, but treating days as a group allows you to easily model occupancy as a function of temporally varying covariates.

Solution: We show how this model can be coded in the BUGS language. We note that it can be described as a constrained version of the dynamic model, where the colonisation rate is implicitly assumed to be equal to the survival probability; see section 7.4 in the occupancy bible by MacKenzie et al. (2006). We assume that you have an R workspace with all the necessary data in place.

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

# Specify priors
for (k in 1:nyear){
   psi[k] ~ dunif(0, 1)
   p[k] ~ dunif(0, 1)
   }

# Ecological submodel: Define state conditional on parameters
for (i in 1:nsite){
   for (k in 1:nyear){
      z[i,k] ~ dbern(psi[k])
      } #k
   } #i

# Observation model: Define observation conditional on state
for (i in 1:nsite){
   for (j in 1:nrep){
      for (k in 1:nyear){
         muy[i,j,k] <- z[i,k]*p[k]
         y[i,j,k] ~ dbern(muy[i,j,k])
         } #k
      } #j
   } #i

# Derived parameters: Sample occupancy and growth rate
n.occ[1] <- sum(z[1:nsite,1])
```

```
for (k in 2:nyear){
   n.occ[k] <- sum(z[1:nsite,k])
   growthr[k] <- psi[k]/psi[k-1]
   }
}
",fill = TRUE)
sink()

# Bundle data
win.data <- list(y = y, nsite = dim(y)[1], nrep = dim(y)[2], nyear =
dim(y)[3])

# Initial values
zst <- apply(y, c(1, 3), max) # Observed occurrence as inits for z
inits <- function(){ list(z = zst)}

# Parameters monitored
params <- c("psi", "p", "n.occ", "growthr")

# MCMC settings
ni <- 2500
nt <- 4
nb <- 500
nc <- 3

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

# Summarize posteriors
print(out, dig = 2)
Inference for Bugs model at "ImplicitDynocc.txt", fit using WinBUGS,
 3 chains, each with 2500 iterations (first 500 discarded), n.thin = 4
 n.sims = 1500 iterations saved
             mean     sd  2.5%   25%   50%   75%  97.5% Rhat n.eff
psi[1]       0.13   0.20  0.00  0.01  0.04  0.16   0.75 1.09    31
psi[2]       0.15   0.23  0.00  0.01  0.05  0.18   0.88 1.02   110
psi[3]       0.32   0.26  0.03  0.10  0.22  0.50   0.91 1.01   170
psi[4]       0.14   0.05  0.07  0.11  0.13  0.17   0.25 1.00  1500
psi[5]       0.21   0.05  0.12  0.17  0.20  0.24   0.31 1.00  1500
psi[6]       0.21   0.05  0.12  0.18  0.20  0.23   0.31 1.00  1500
psi[7]       0.11   0.05  0.04  0.07  0.10  0.13   0.23 1.00  1500
p[1]         0.25   0.28  0.00  0.02  0.11  0.40   0.94 1.11    26
p[2]         0.22   0.27  0.00  0.02  0.08  0.35   0.91 1.01   140
p[3]         0.15   0.15  0.01  0.04  0.09  0.20   0.59 1.02   120
p[4]         0.61   0.14  0.32  0.52  0.62  0.72   0.84 1.00  1500
p[5]         0.71   0.10  0.49  0.64  0.72  0.78   0.87 1.00  1500
p[6]         0.71   0.10  0.49  0.65  0.72  0.78   0.87 1.00   850
p[7]         0.56   0.18  0.21  0.43  0.58  0.70   0.86 1.01   650
n.occ[1]    11.68  18.94  0.00  0.00  3.00 14.00  71.52 1.24    18
n.occ[2]    13.74  21.81  0.00  0.00  3.00 17.00  83.00 1.03   140
n.occ[3]    29.63  25.35  3.00  8.00 21.00 47.00  87.00 1.01   140
n.occ[4]    12.69   3.24 10.00 11.00 12.00 14.00  21.00 1.00  1200
n.occ[5]    19.02   2.23 17.00 17.00 18.00 20.00  25.00 1.00  1500
n.occ[6]    18.98   2.20 17.00 17.00 18.00 20.00  25.00 1.00  1500
n.occ[7]     9.47   4.28  6.00  7.00  8.00 10.00  20.00 1.00   920
growthr[2]  41.78 489.46  0.01  0.18  1.32  7.35 136.85 1.09    29
growthr[3]  48.29 350.66  0.12  1.04  4.39 19.81 244.10 1.02   100
growthr[4]   1.12   1.42  0.12  0.27  0.62  1.33   4.74 1.01   180
growthr[5]   1.65   0.68  0.66  1.17  1.53  2.03   3.30 1.00  1500
growthr[6]   1.05   0.36  0.52  0.79  0.98  1.23   1.89 1.00  1500
```

```
growthr[7]   0.55   0.30   0.18   0.35   0.48   0.68   1.28 1.00  1500
deviance   171.61  19.72 137.05 157.40 170.20 184.02 214.85 1.00   600
```

Exercise 4

Task: Site-occupancy models represent the only currently available species distribution modeling framework that can estimate true, rather than apparent distributions (Kéry et al., 2010a; Kéry, 2011b). However, modeling occurrence and observation jointly can be difficult in marginal data situations. Devise a simulation study, where you vary the number of sites, occupancy and detection probability as well as the number of replicate visits per site to see that in small-data situations, occupancy estimates will be biased high, and sometimes severely so. Do so in a model with constant detection and occurrence probability. Hint: this is a somewhat larger project.

Solution: Actually, this could be an arbitrarily large project, and, as in exercise 3 in chapter 6, we will only give a sketch of how such a simulation study could be tackled by generating, say, 100 data sets for each of a combination of levels of the four factors Number of sites (R), number of visits (T), occupancy probability (ψ) and detection probability (p). For a full-blown study, you might for instance want to choose the following factor levels: R = (10, 25, 50, 250), T = (2,3,5), ψ = (0.1, 0.2, 0.3) and p = (0.1, 0.2, 0.3). Here, we will illustrate the scenario with R = 250, T = 3, ψ = 0.2 and p = 0.1.

We will again first define arrays to save the results from the data simulation and data analysis routines, second, loop over 100 simulation replicates of the data generation/data analysis cycle and third, summarize the results, i.e., compare what the model told us about the population with what we know about that population. We will first take the data-generation code in section 13.3.1 and package it in a function.

```
data.fn <- function(R = 250, T = 3, psi = 0.2, p = 0.1){
   y <- matrix(NA, nrow = R, ncol = T)
   z <- rbinom(n = R, size = 1, prob = psi)  # Latent occurrence state
   for (j in 1:T){
      y[,j] <- rbinom(n = R, size = 1, prob = z * p)
      }
   n.occ <- sum(z)
   n.occ.obs <- sum(apply(y, 1, max))
   return(list(R=R, T=T, psi=psi, p=p, z=z, y=y, n.occ=n.occ,
n.occ.obs=n.occ.obs))
   }
```

Execute once to try out – it seems to work fine.

```
str(data <- data.fn())
List of 8
 $ R        : num 250
 $ T        : num 3
 $ psi      : num 0.2
 $ p        : num 0.1
 $ z        : num [1:250] 1 1 1 0 0 1 1 0 0 0 ...
 $ y        : num [1:250, 1:3] 0 1 0 0 0 1 0 0 0 0 ...
 $ n.occ    : num 44
 $ n.occ.obs: num 15
```

Then, we package the analysis functions into one single function, which we call `model.fn()`.

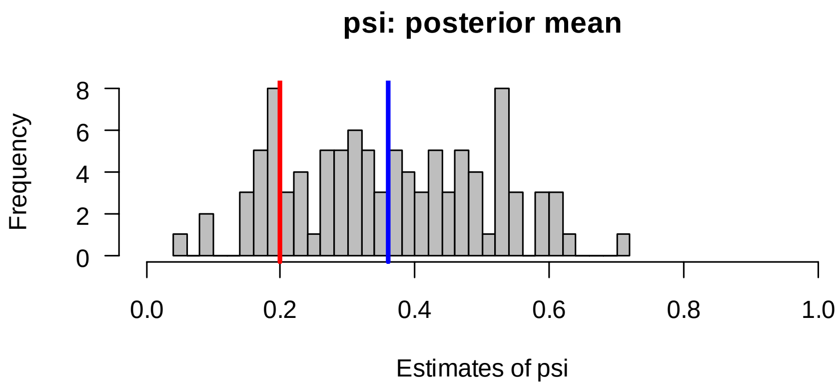
```
# Define a function to do data augmentation, run analysis using Mh and
return results all at once
model.fn <- function(ni = 1200, nt = 2, nb = 200, nc = 3, data.file = data,
debg = FALSE){
# Function arguments:
# ni/nt/nb/nc -- MCMC settings
# data.file -- name of the object with the simulated data
# debg -- setting of DEBUG argument in bugs()

# Specify model in BUGS language
sink("model.txt")
cat("
model {
psi ~ dunif(0, 1)
p ~ dunif(0, 1)
for (i in 1:R){
   z[i] ~ dbern(psi)
   p.eff[i] <- z[i] * p
   for (j in 1:T){
      y[i,j] ~ dbern(p.eff[i])
      } #j
   } #i
occ.fs <- sum(z[])
}
",fill = TRUE)
sink()

# Bundle data
win.data <- list(y = data$y, R = data$R, T = data$T)

# Initial values
zst <- apply(data$y, 1, max)  # Observed occurrence as starting values for z
inits <- function() list(z = zst)

# Parameters monitored
params <- c("psi", "p", "occ.fs")

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

# Return stuff
return(post.estimates = out$summary)
}
```

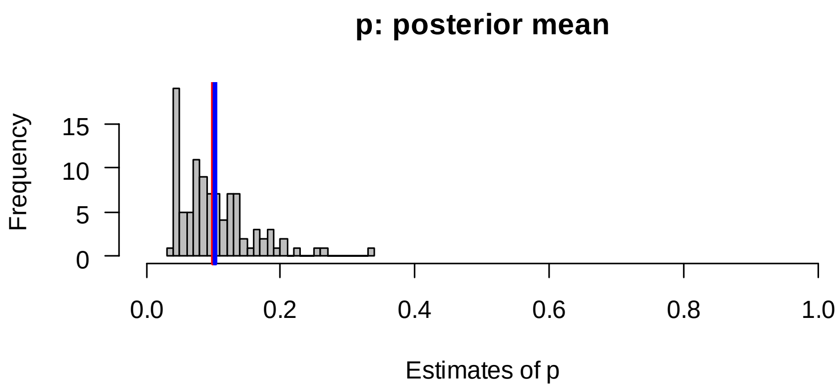
Try out a single data simulation/data analysis cycle.

```
data <- data.fn(R = 250, T = 3, psi = 0.2, p = 0.1)
estimates <- model.fn(ni = 2500, nt = 2, nb = 500, nc = 3, data.file =
data, debg = TRUE)
```

That seems to work. Now we run 100 simulation replicates for the single chosen design point with R = 250, T = 3, ψ = 0.2 and p = 0.1. The next block of code could be repeated for each design point of a larger, genuine simulation exercise. Or better still, the block of code could itself be put in a loop over the design points of the simulation.

```
# Set up data structures to hold the results
simreps <- 100
data.sets <- array(NA, dim = c(250, 3, simreps))
solutions <- array(NA, dim = c(4, 9, simreps))
rownames(solutions) <- rownames(estimates)
colnames(solutions) <- colnames(estimates)

# Run data generation/data analysis cycle simrep times
for (i in 1:simreps){
   cat(paste("\n\n*** SimRep", i, "***\n"))
   data <- data.fn(R = 250, T = 3, psi = 0.2, p = 0.1)
   data.sets[,,i] <- data$y
   estimates <- model.fn(ni = 2500, nt = 2, nb = 500, nc = 3, data.file =
data,
   debg = FALSE)
   solutions[,,i] <- estimates
   }

# Summarize simulation results
par(mfrow = c(2, 1))
hist(solutions[1,1,], breaks = 25, col = "grey", xlab = "Estimates of psi",
main = "psi: posterior mean", xlim = c(0, 1), las = 1)
abline(v = 0.2, col = "red", lwd = 3)
abline(v = mean(solutions[1,1,]), col = "blue", lwd = 3)

hist(solutions[2,1,], breaks = 25, col = "grey", xlab = "Estimates of p",
main = "p: posterior mean", xlim = c(0, 1), las = 1)
abline(v = 0.1, col = "red", lwd = 3)
abline(v = mean(solutions[2,1,]), col = "blue", lwd = 3)
```





As before, red is the truth and blue is the mean of the estimates. We see that the estimate of *p* is on average not biased, but the distribution is skewed. The estimator of ψ , however, is usually biased high, with the mean of the 100 estimates (0.36) almost double the true value (0.2).

Exercise 5

Task: In the multistate occupancy model, add an effect of Julian date on detection probability of hooting adults and begging young, i.e., p_2, $p_{3,2}$ and $p_{3,3}$. Don't forget to standardize the covariate.

Solution: We want to model the effect of a covariate that differs by day; therefore, we need to start with model 2 in section 13.6., modify that and call it model 3. As usual, we assume that you have all the necessary data in your R workspace already.

We start with preparing the data. We put the dates into a matrix, standardize and replace the NA's with zeroes.

```
# Bundle data
y <- as.matrix(owls[, 2:6])
y <- y + 1
DATE <- as.matrix(owls[, 7:11])
mn.date <- mean(DATE, na.rm = TRUE)
sd.date <- sd(c(DATE), na.rm = TRUE)
```

```
DATE <- (DATE  - mn.date)/ sd.date
DATE[is.na(DATE)]<- 0
win.data <- list(y = y, DATE = DATE, R = dim(y)[1], T = dim(y)[2])
```

Then, we specify the model. Some changes are necessary if we want to model the detection probabilities as functions of Julian date. Since not all sites were visited at the same date (the variable DATE is a matrix, not a vector), we have to add a site dimension to the detection parameters, to the observation matrix and to the observation equation. Then we have to write the detection probabilities as a linear function of DATE. For the detection probability p2 this is straightforward: as usual we use the logit link function to specify the linear model. For the detection probabilities p3 there is some twist, since each of them must be between 0 and 1 *and* all must sum to 1. We therefore use the multinomial logit function to formulate the linear models (see also chapter 9.6.3 for an analogous case).

```
# Specifiy model in BUGS language
sink("model3.bug")
cat("
model {

# Priors
psi ~ dunif(0, 1)
r ~ dunif(0,1 )

for (t in 1:T){
   for (s in 1:R){
      # linear models on logit and multinomial logit scale
      logit(p2[s,t]) <- int.p2 + beta.p2 * DATE[s,t]
      lp3[2,s,t] <- int.p32 + beta.p32 * DATE[s,t]
      lp3[3,s,t] <- int.p33 + beta.p33 * DATE[s,t]

      p3[1,s,t] <- 1-p3[2,s,t]-p3[3,s,t]  # calculate last p3
      p3[2,s,t] <- exp(lp3[2,s,t]) / (1 + exp(lp3[2,s,t]) +
exp(lp3[3,s,t]))   # backtransformation to {0,1} scale
      p3[3,s,t] <- exp(lp3[3,s,t]) / (1 + exp(lp3[2,s,t]) +
exp(lp3[3,s,t]))   # backtransformation to {0,1} scale
      } # s
   } #t
int.p2 ~ dnorm(0, 0.01)
beta.p2 ~ dnorm(0, 0.01)
int.p32 ~ dnorm(0, 0.01)
beta.p32 ~ dnorm(0, 0.01)
int.p33 ~ dnorm(0, 0.01)
beta.p33 ~ dnorm(0, 0.01)

# Define state vector
for (s in 1:R){
   phi[s,1] <- 1 - psi              # Prob. of non-occupation
   phi[s,2] <- psi * (1-r)        # Prob. of occupancy without repro.
   phi[s,3] <- psi * r              # Prob. of occupancy and repro
   }

# Define observation matrix
# Order of indices: true state, time, observed state
for (s in 1:R){
   for (t in 1:T){
      p[1,s,t,1] <- 1
      p[1,s,t,2] <- 0
      p[1,s,t,3] <- 0
      p[2,s,t,1] <- 1-p2[s,t]
```

```
      p[2,s,t,2] <- p2[s,t]
      p[2,s,t,3] <- 0
      p[3,s,t,1] <- p3[1,s,t]
      p[3,s,t,2] <- p3[2,s,t]
      p[3,s,t,3] <- p3[3,s,t]
      } #t
   } #s

# State-space likelihood
# State equation: model of true states (z)
for (s in 1:R){
   z[s] ~ dcat(phi[s,])
   }

# Observation equation
for (s in 1:R){
   for (t in 1:T){
      y[s,t] ~ dcat(p[z[s],s,t,])
      } #t
   } #s

# Derived quantities
for (s in 1:R){
   occ1[s] <- equals(z[s], 1)
   occ2[s] <- equals(z[s], 2)
   occ3[s] <- equals(z[s], 3)
   }
n.occ[1] <- sum(occ1[]) # Sites in state 1
n.occ[2] <- sum(occ2[]) # Sites in state 2
n.occ[3] <- sum(occ3[]) # Sites in state 3
}
",fill=TRUE)
sink()

# Initial values
zst <- apply(y, 1, max, na.rm = TRUE)
zst[zst == "-Inf"] <- 1
inits <- function(){list(z = zst, beta.p32 = 0, beta.p33 = 0)}

# Parameters monitored
params <- c("r", "psi", "n.occ", "int.p2", "beta.p2", "int.p32",
"beta.p32", "int.p33", "beta.p33")

# MCMC settings
ni <- 5000
nt <- 1
nb <- 2000
nc <- 3

# Call WinBUGS from R (BRT < 1 min)
out3 <- bugs(win.data, inits, params, "model3.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(out3, dig = 3)
             mean    sd   2.5%     25%    50%    75%  97.5%  Rhat n.eff
r           0.594 0.184  0.253   0.460  0.584  0.725  0.961 1.001  9000
psi         0.464 0.129  0.252   0.375  0.449  0.534  0.782 1.003  1200
n.occ[1]   21.501 4.472  9.000  20.000 22.000 24.000 28.000 1.008   650
n.occ[2]    7.241 3.708  0.000   5.000  7.000  9.000 15.000 1.002  2500
n.occ[3]   11.258 4.362  5.000   8.000 10.000 14.000 22.000 1.002  2300
```

```
int.p2   -2.348 5.004 -15.840  -3.566 -1.118  0.271  4.058 1.002  6100
beta.p2  -7.170 6.398 -21.910 -10.230 -6.070 -3.517  5.302 1.005   460
int.p32  -3.950 3.239 -12.440  -5.392 -2.848 -1.646 -0.325 1.003  1000
beta.p32 -3.377 2.628 -10.270  -4.390 -2.553 -1.613 -0.254 1.003   840
int.p33  -0.999 0.809  -2.594  -1.539 -0.992 -0.435  0.542 1.001  8300
beta.p33  1.004 1.050  -0.690   0.282  0.865  1.587  3.436 1.001  9000
deviance 46.928 9.438  33.520  39.410 45.215 53.272 68.070 1.003   940
```

We notice that the estimated slope parameters of the detection probabilities related to hooting owls (p2, p32) are negative, while the detection probability related with begging young is increasing. This result makes intuitively sense. However, the credible intervals of beta.p2 and beta.p33 include zero, thus these seasonal trends are not strongly supported by the data.

That's it. **Bravo, you're through!**