Class 2 live demo: Crash course on Bayesian statistics and MCMC algorithms

The team

last updated: 2021-04-23

Introduction

In this demo, we show how to implement the Bayes theorem by brute force with numerical integration, then we use a Metropolis algorithm. We use the survival example with $y = 19$ alive individuals out of $n = 57$ released. A binomial likelihood is assumed, with a beta prior on survival.

Load the packages we will need.

```
library(tidyverse)
```

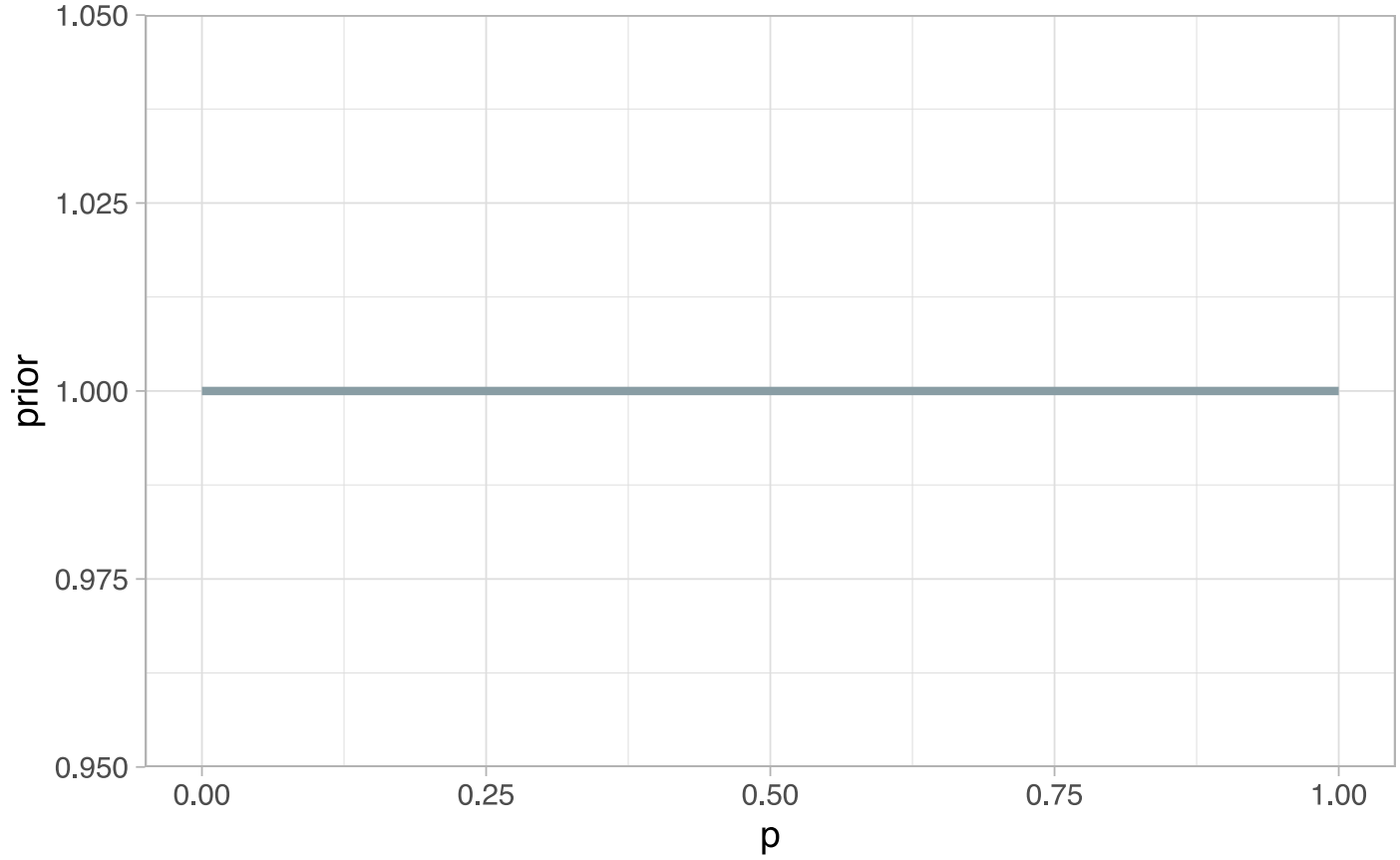
The data.

```
y <- 19 # nb of success
n <- 57 # nb of attempts
```

Brute force via numerical integration

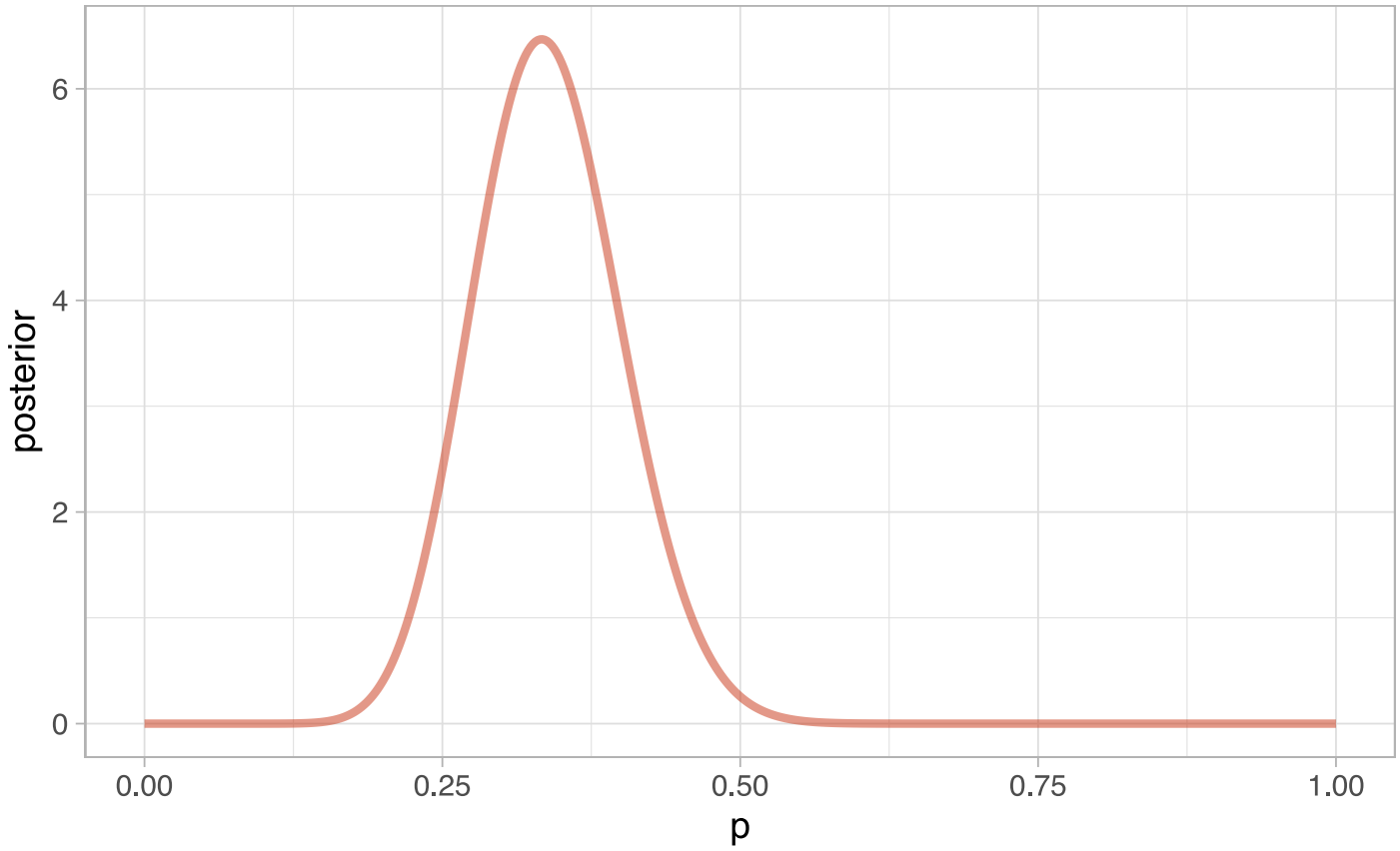
First we defined the prior.

```
a <- 1 # param of the prior
b <- 1 # param of the prior
p <- seq(0,1,.002) # grid
prior <- dbeta(p,a,b) # eval beta prior
dfprior <- data.frame(p = p, prior = prior)
dfprior %>%
  ggplot() +
  geom_line(aes(x = p, y = prior),
            size = 1.5,
            color = wesanderson::wes_palettes$Royal1[1])
```



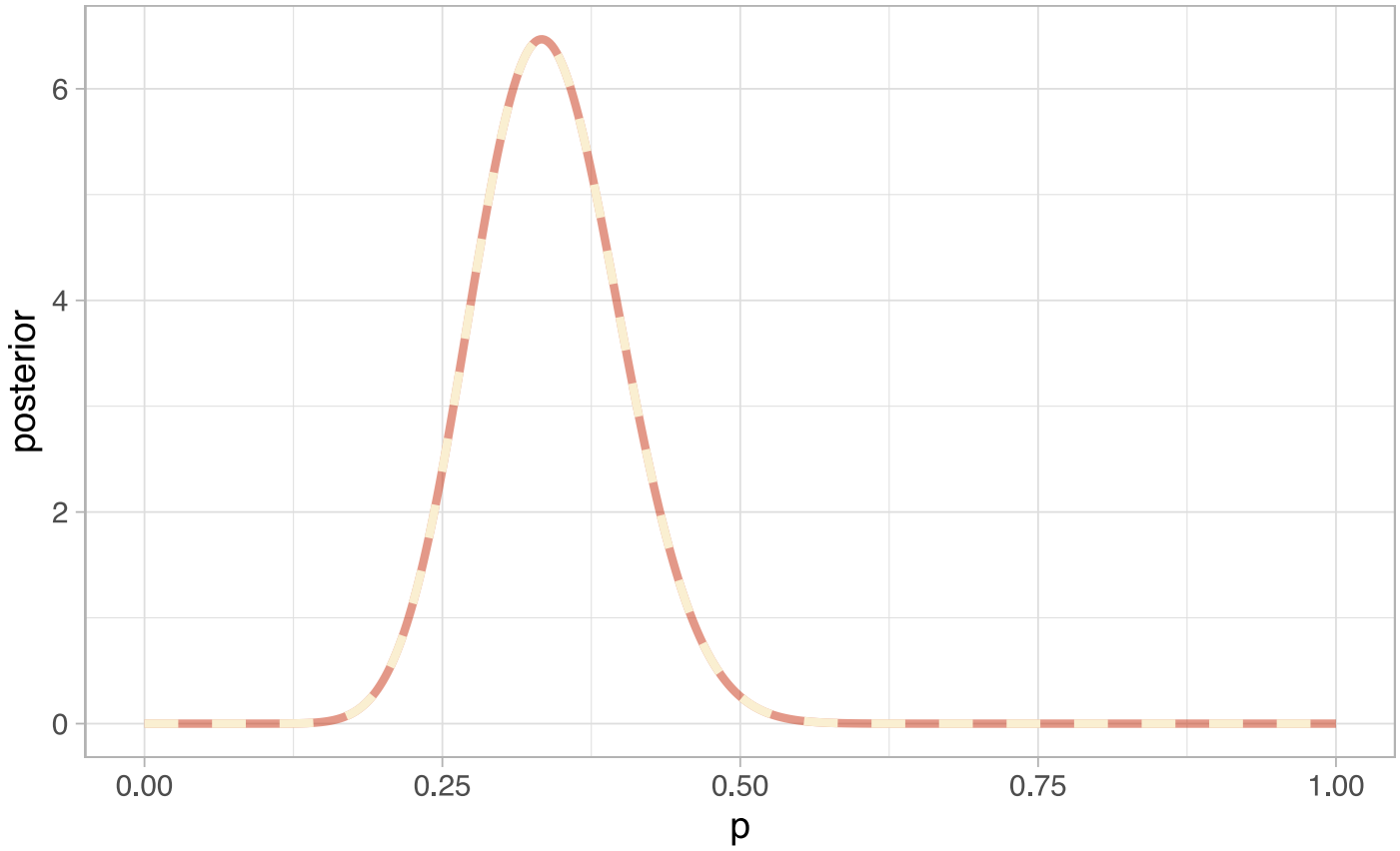
Now we form the numerator and denominator of the Bayes formula.

```
numerator <- function(p) dbinom(y,n,p) * dbeta(p,a,b) # likelihood x prior
denominator <- integrate(numerator,0,1)$value # denominator
numerical_posterior <- data.frame(p = p, posterior = numerator(p)/denominator)
numerical_posterior %>%
  ggplot() +
  geom_line(aes(x = p, y = posterior),
            size = 1.5,
            col = wesanderson::wes_palettes$Royal1[2],
            alpha = 0.5)
```



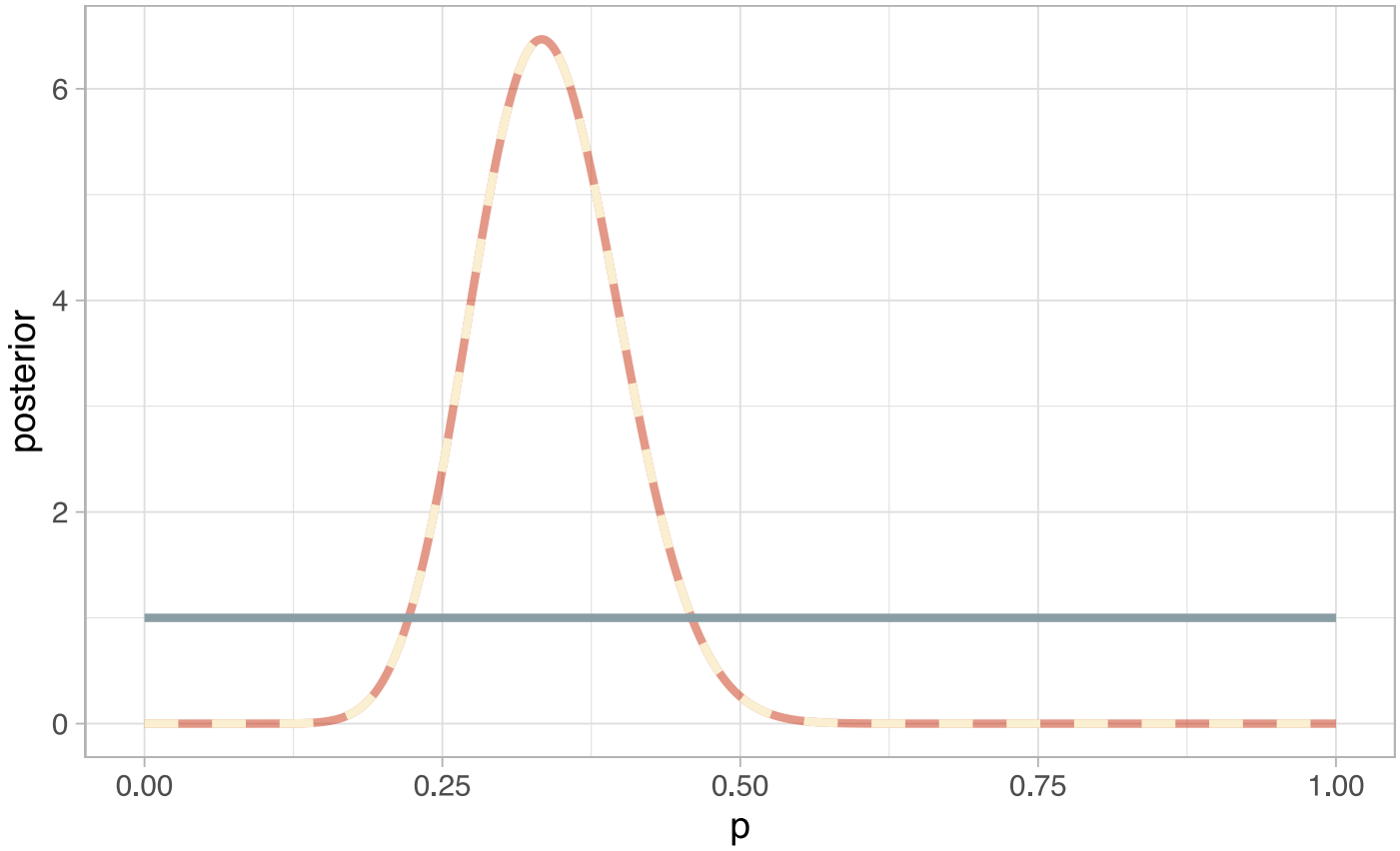
Now we compare the posterior distribution we obtained by numerical integration to the explicit distribution we get through conjugacy (binomial likelihood and beta prior gives beta posterior).

```
explicit_posterior <- dbeta(p, y + a, n - y + b) # beta posterior
dfexpposterior <- data.frame(p = p, explicit_posterior = explicit_posterior)
ggplot() +
  geom_line(data = numerical_posterior,
            aes(x = p, y = posterior),
            size = 1.5,
            col = wesanderson::wes_palettes$Royal1[2],
            alpha = 0.5) +
  geom_line(data = dfexpposterior,
            aes(x = p, y = explicit_posterior),
            size = 1.5,
            col = wesanderson::wes_palettes$Royal1[3],
            linetype = "dashed")
```



Last, we put everything on the same plot.

```
ggplot() +
  geom_line(data = numerical_posterior,
            aes(x = p, y = posterior),
            size = 1.5,
            col = wesanderson::wes_palettes$Royal1[2],
            alpha = 0.5) +
  geom_line(data = dfexpposterior,
            aes(x = p, y = explicit_posterior),
            col = wesanderson::wes_palettes$Royal1[3],
            size = 1.5,
            linetype = "dashed") +
  geom_line(data = dfprior,
            aes(x = p, y = prior),
            col = wesanderson::wes_palettes$Royal1[1],
            size = 1.5)
```



Implement the Metropolis algorithm

In this section, we implement the Metropolis algorithm.

First the data.

```
# survival data, 19 "success" out of 57 "attempts"
survived <- 19
released <- 57
```

Then we define the log-likelihood function, which is binomial. Note that we use the log.

```
# log-likelihood function
loglikelihood <- function(x, p){
  dbinom(x = x, size = released, prob = p, log = TRUE)
}
```

We then define the prior, a uniform distribution between 0 and 1, or a beta distribution with parameters 1 and 1.

```
# prior density
logprior <- function(p){
  dunif(x = p, min = 0, max = 1, log = TRUE)
}
```

Eventually, we form the posterior density distribution, on the log scale. Note that we commented out the denominator because we won't need it in the Metropolis algorithm (it cancels out when we form the acceptance ratio.

```
# posterior density function (log scale)
posterior <- function(x, p){
  loglikelihood(x, p) + logprior(p) # - log(Pr(data))
}
```

Now we need to propose a candidate value for survival at each iteration of the algorithm. To do so, we work on the logit scale, and use a normal distribution to jump away from the current value.

```
# propose candidate value
move <- function(x, away = .2){
  logitx <- log(x / (1 - x))
  logit_candidate <- logitx + rnorm(1, 0, away)
  candidate <- plogis(logit_candidate)
  return(candidate)
}
```

We pick 100 iterations, pre-allocate some memory to store the results and set the seed for reproducibility.

```
# set up the scene
steps <- 100
theta.post <- rep(NA, steps)
set.seed(1234)
```

To start the algorithm, we also need an initial value for survival. It's going to be 0.5 here. This is our first value stored. When running several chains, pick different initial values.

```
# pick starting value (step 1)
inits <- 0.5
theta.post[1] <- inits
```

Now we run the Metropolis algorithm.

```
for (t in 2:steps){ # repeat steps 2-4 (step 5)

  # propose candidate value for prob of success (step 2)
  theta_star <- move(theta.post[t-1])

  # calculate ratio R (step 3)
  pstar <- posterior(survived, p = theta_star)
  pprev <- posterior(survived, p = theta.post[t-1])
  logR <- pstar - pprev # here we see that the denominator log(Pr(data)) cancels out
                        # logR = loglikelihood(survived, theta_star) + logprior(theta_st
ar) - log(Pr(data)) -
                        #        (loglikelihood(survived, theta.post[t-1]) + logprior(th
eta.post[t-1]) - log(Pr(data)))
  R <- exp(logR)

  # decide to accept candidate value or to keep current value (step 4)
  accept <- rbinom(1, 1, prob = min(R, 1))
  theta.post[t] <- ifelse(accept == 1, theta_star, theta.post[t-1])
}
```

Let's have a look to the values we generated from the posterior distribution.

```
head(theta.post)
```

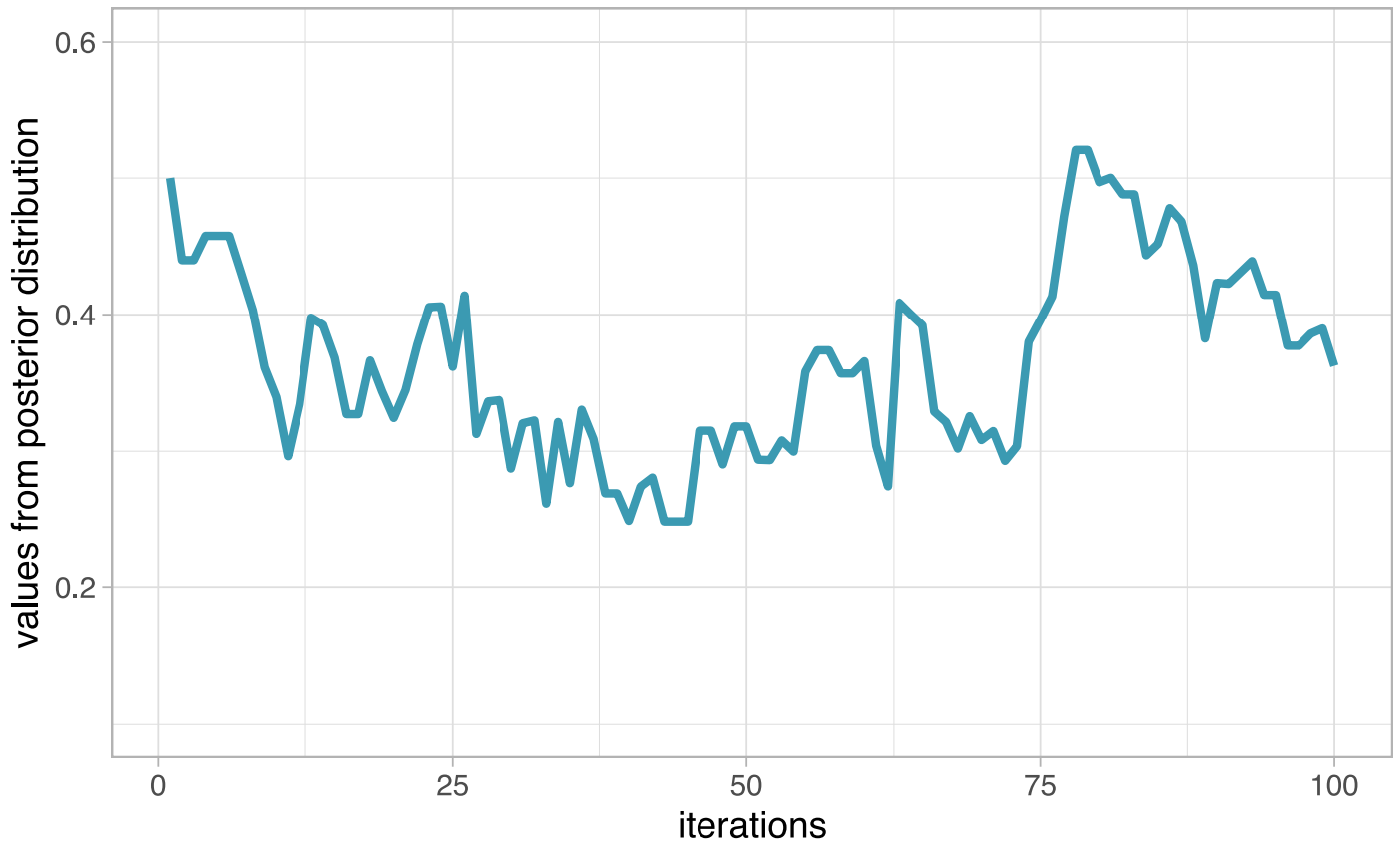
```
[1] 0.5000000 0.4399381 0.4399381 0.4577124 0.4577124 0.4577124
```

```
tail(theta.post)
```

```
[1] 0.4145878 0.3772087 0.3772087 0.3860516 0.3898536 0.3624450
```

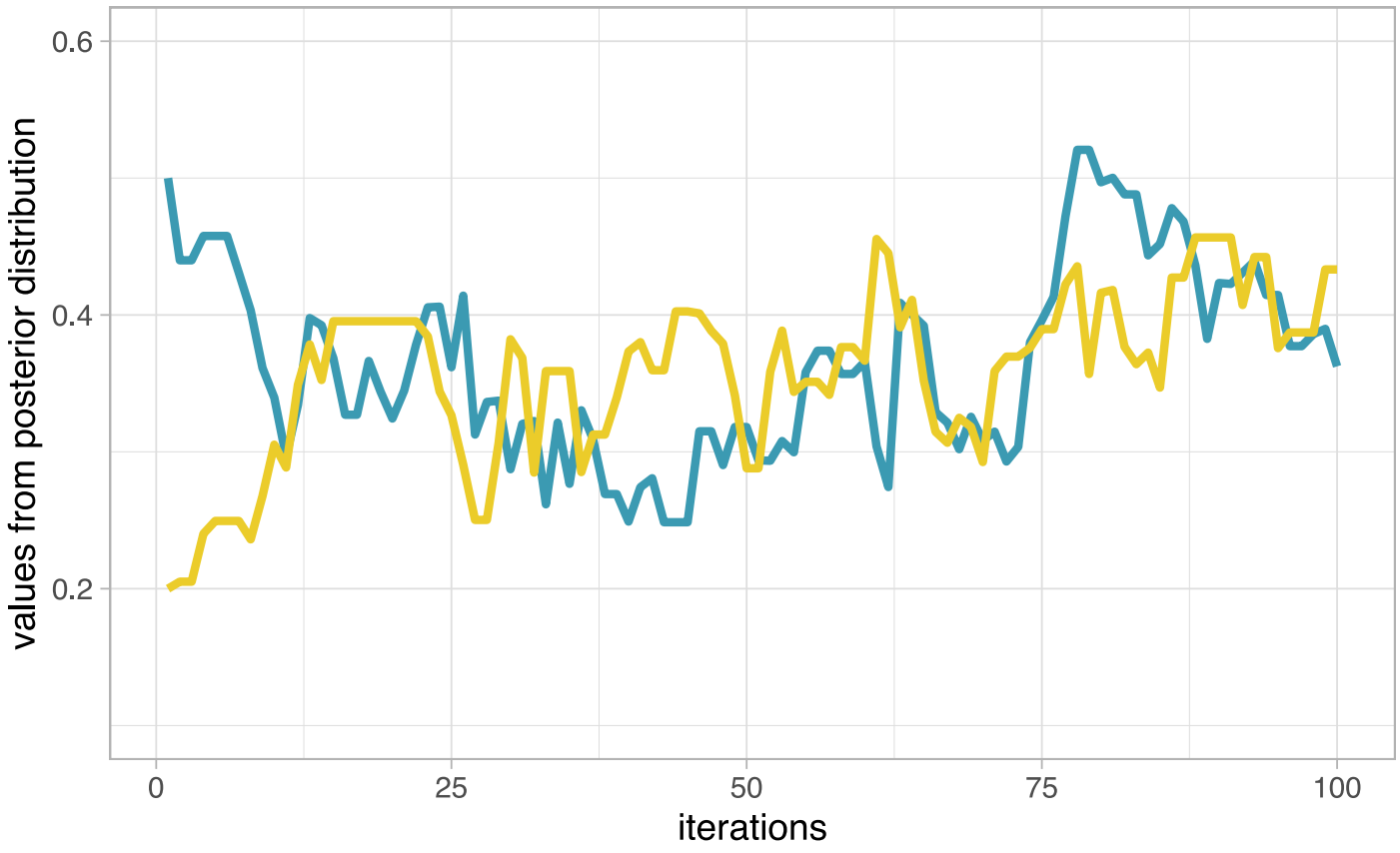
And now, the trace of the chain.

```
df <- data.frame(x = 1:steps, y = theta.post)
df %>%
  ggplot() +
  geom_line(aes(x = x, y = y), size = 1.5, color = wesanderson::wes_palettes$Zissou1[1])
+
  labs(x = "iterations", y = "values from posterior distribution") +
  ylim(0.1, 0.6)
```



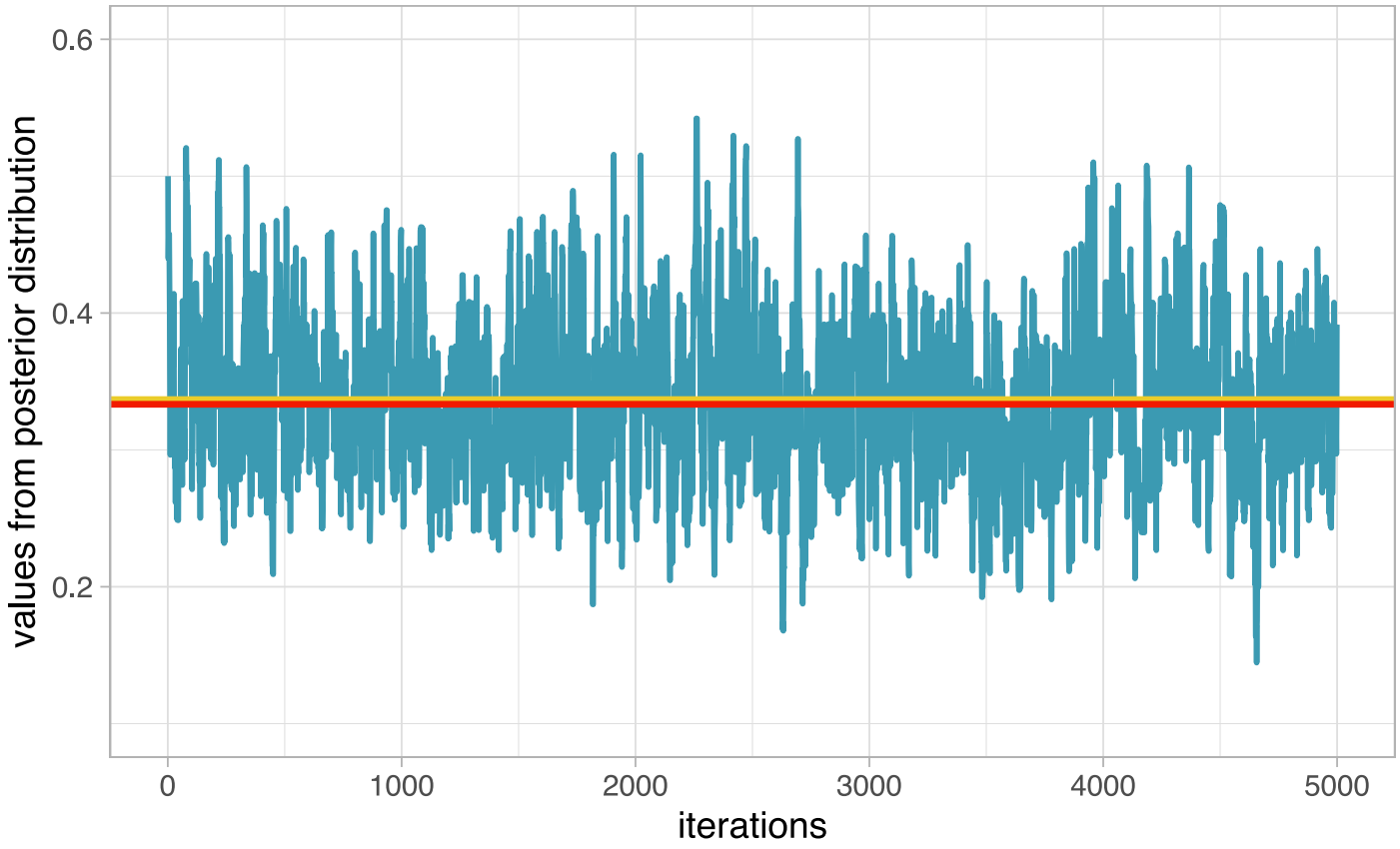
Let's run another chain, now with initial value 0.2 for survival, and visualise the trace of the two chains.

```r
# pick starting value (step 1)
inits <- 0.2
theta.post2 <- rep(NA, steps)
theta.post2[1] <- inits

for (t in 2:steps){ # repeat steps 2-4 (step 5)
  # propose candidate value for prob of success (step 2)
  theta_star <- move(theta.post2[t-1])
  # calculate ratio R (step 3)
  pstar <- posterior(survived, p = theta_star)
  pprev <- posterior(survived, p = theta.post[t-1])
  logR <- pstar - pprev
  R <- exp(logR)

  # decide to accept candidate value or to keep current value (step 4)
  accept <- rbinom(1, 1, prob = min(R, 1))
  theta.post2[t] <- ifelse(accept == 1, theta_star, theta.post2[t-1])
}

df2 <- data.frame(x = 1:steps, y = theta.post2)
  ggplot() +
  geom_line(data = df, aes(x = x, y = y), size = 1.5, color = wesanderson::wes_palettes
$Zissou1[1]) +
  geom_line(data = df2, aes(x = x, y = y), size = 1.5, color = wesanderson::wes_palettes
$Zissou1[3]) +
  labs(x = "iterations", y = "values from posterior distribution") +
  ylim(0.1, 0.6)
```



Last, we re-run these two chains with more iterations, say 5000. We also add the posterior mean in yellow, and the maximum likelihood estimate in red.

```
# set up the scene
steps <- 5000
theta.post <- rep(NA, steps)
set.seed(1234)

# pick starting value (step 1)
inits <- 0.5
theta.post[1] <- inits

for (t in 2:steps){ # repeat steps 2-4 (step 5)

  # propose candidate value for prob of success (step 2)
  theta_star <- move(theta.post[t-1])

  # calculate ratio R (step 3)
  pstar <- posterior(survived, p = theta_star)
  pprev <- posterior(survived, p = theta.post[t-1])
  logR <- pstar - pprev
  R <- exp(logR)

  # decide to accept candidate value or to keep current value (step 4)
  accept <- rbinom(1, 1, prob = min(R, 1))
  theta.post[t] <- ifelse(accept == 1, theta_star, theta.post[t-1])
}

df <- data.frame(x = 1:steps, y = theta.post)
df %>%
  ggplot() +
  geom_line(aes(x = x, y = y), size = 1, color = wesanderson::wes_palettes$Zissou1[1]) +
  labs(x = "iterations", y = "values from posterior distribution") +
  ylim(0.1, 0.6) +
  geom_hline(aes(yintercept = mean(theta.post)),
             color = wesanderson::wes_palettes$Zissou1[3],
             size = 1.2) +
  geom_hline(aes(yintercept = 19/57),
             color = wesanderson::wes_palettes$Zissou1[5],
             size = 1.2)
```


values from posterior distribution
0.6
0.4
0.2
0
1000
2000
3000
4000
5000
iterations

Class 3 live demo: Intro to Nimble

The team

last updated: 2021-05-12

Introduction

In this demo, we show a simple example of how to use nimble. We use the survival example with $y = 19$ alive individuals out of $n = 57$ released. A binomial likelihood is assumed, with a uniform prior on survival.

Prepare everything

First load the nimble package.

```
library(nimble)
```

Then define the model, likelihood and prior.

```
naive.survival.model <- nimbleCode({
  # prior
  phi ~ dunif(0, 1)
  # likelihood
  y ~ dbinom(phi, n)
})
```

Store the constants and data in lists.

```
my.constants <- list(n = 57)
my.data <- list(y = 19)
```

Write a function to pick initial values for survival probability.

```
initial.values <- function() list(phi = runif(1,0,1))
```

Try the function.

```
initial.values()
```

```
$phi
[1] 0.9442041
```

Specify the parameter you'd like to monitor, and for which you'd like to carry out posterior inference.

```
parameters.to.save <- c("phi")
```

Set the details of the MCMC run. Here we will use 5000 iterations, including 1000 for the burn-in period, and run two chains.

```
n.iter <- 5000
n.burnin <- 1000
n.chains <- 2
n.thin <- 1
```

Run nimble

Run nimble.

```
mcmc.output <- nimbleMCMC(code = naive.survival.model,      # model code
                          data = my.data,                   # data
                          constants = my.constants,         # constants
                          inits = initial.values,           # initial values
                          monitors = parameters.to.save,    # parameters to monitor
                          thin = n.thin,                    # thinning interval (default
 = 1)
                          niter = n.iter,                   # nb iterations
                          nburnin = n.burnin,               # length of the burn-in
                          nchains = n.chains)               # nb of chains
```

```
|-------------|-------------|-------------|-------------|
|-----------------------------------------------------|
|-------------|-------------|-------------|-------------|
|-----------------------------------------------------|
```

Post-process results

First, we have a look to the structure of the outputs. We have a list with two elements, each one is a chain with 4000 values (number of iterations minus the iterations we used for the burn-in) for survival.

```
str(mcmc.output)
```

```
List of 2
 $ chain1: num [1:4000, 1] 0.281 0.281 0.281 0.281 0.348 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr "phi"
 $ chain2: num [1:4000, 1] 0.352 0.352 0.352 0.352 0.401 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr "phi"
```

Let's have a look to the first chain.

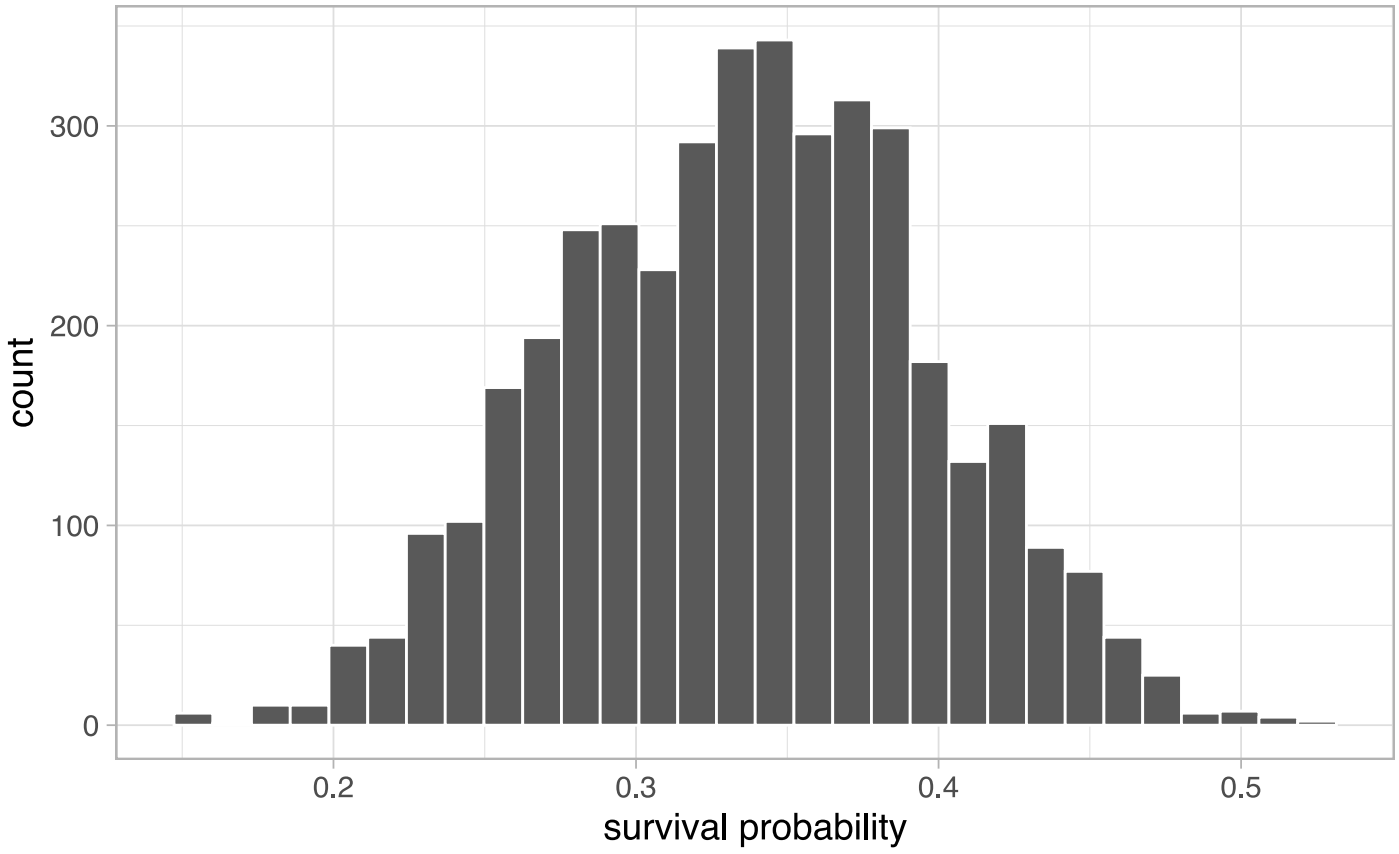
```
head(mcmc.output$chain1)
```

```
          phi
[1,] 0.2808716
[2,] 0.2808716
[3,] 0.2808716
[4,] 0.2808716
[5,] 0.3475681
[6,] 0.3998572
```

We build a histogram of the values of the first chain, generated from the survival posterior distribution.

```
mcmc.output %>%
  as_tibble() %>%
  janitor::clean_names() %>%
  ggplot() +
  geom_histogram(aes(x = chain1[,"phi"]), color = "white") +
  labs(x = "survival probability")
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



There are several packages that can take care of the post-processing for you. We will use `MCMCvis` (https://joss.theoj.org/papers/10.21105/joss.00640) developed by Casey Youngflesh.

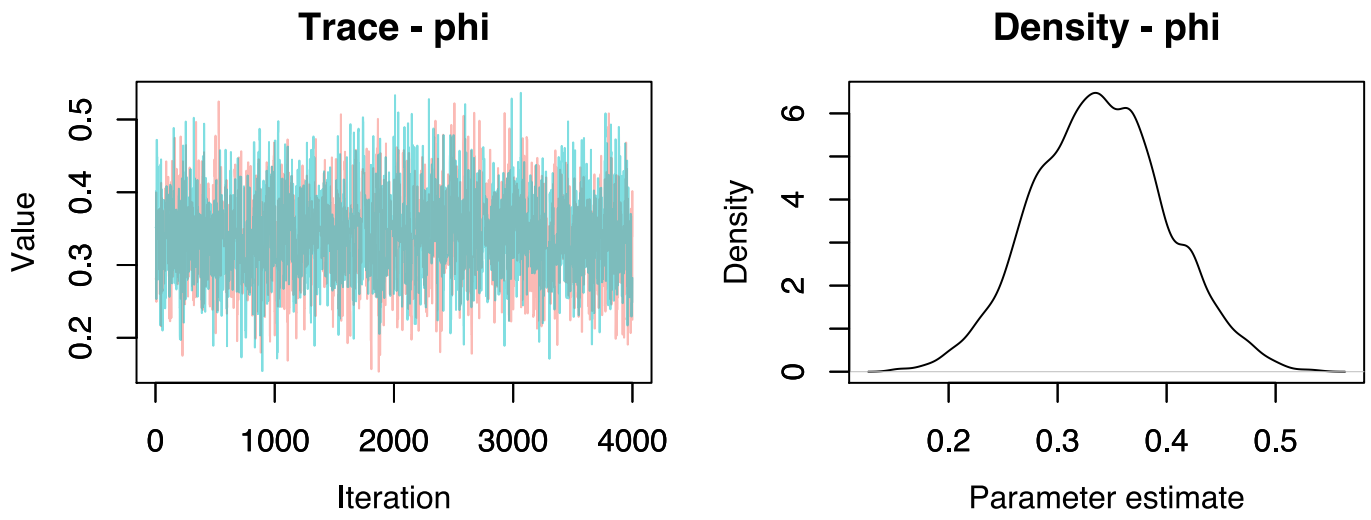
Let's get the numerical summaries of the posterior distribution, along with the R-hat and the effective sample size.

```
library(MCMCvis)
MCMCsummary(mcmc.output, round = 2)
```

```
    mean   sd 2.5%  50% 97.5% Rhat n.eff
phi 0.34 0.06 0.22 0.34  0.46    1  1765
```

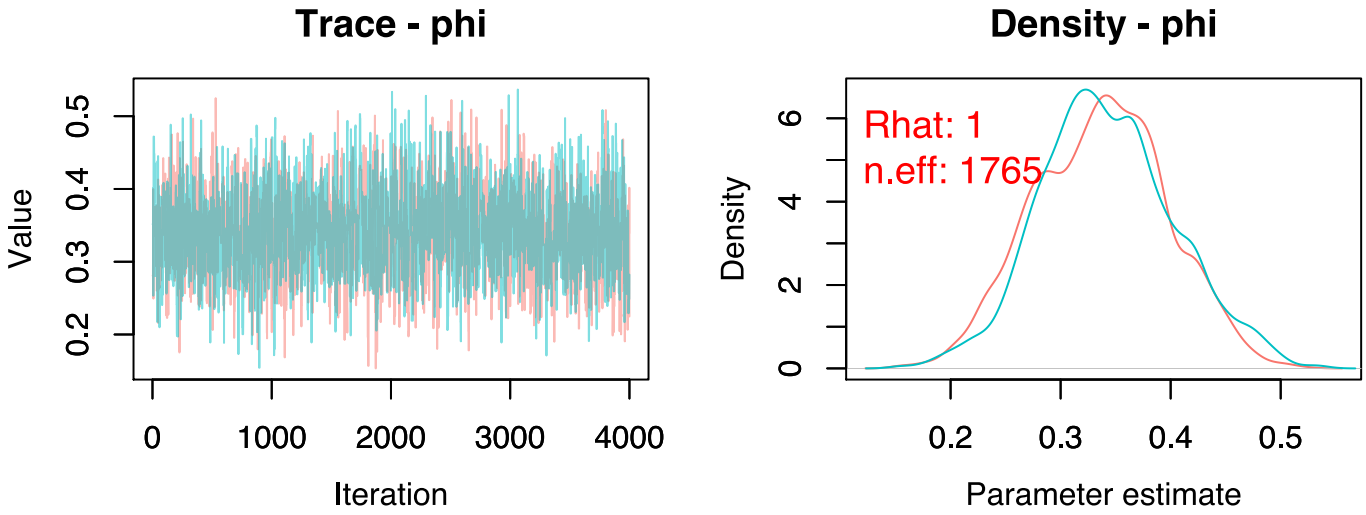
Now the trace plots and (a kernel density estimate of) the posterior distribution of survival.

```
MCMCtrace(mcmc.output,
          pdf = FALSE)
```



You may add the value of R-hat and the effective sample size on the graph.

```
MCMCtrace(mcmc.output,
          pdf = FALSE,
          ind = TRUE,
          Rhat = TRUE,
          n.eff = TRUE)
```



Reproducibility for trouble-shooting and publishing

Being able to reproduce the exactly same initial values and MCMC chains can be very useful for pinpointing the source of errors, and also when publishing your code/workflows.

Here are three easy steps to make your workflow more reproducible:

1) Setting a defined initial seed at the start of your code

```
mySeed <- 0
set.seed(mySeed)
```

2) Using “pre-sampled” initial values

```
initVals <- c(initial.values(), initial.values())
```

```
mcmc.output <- nimbleMCMC(code = naive.survival.model,
                          data = my.data,
                          constants = my.constants,
                          inits = initVals,                # pre-sampled initial values
                          monitors = parameters.to.save,
                          thin = n.thin,
                          niter = n.iter,
                          nburnin = n.burnin,
                          nchains = n.chains)
```

3) Setting the MCMC seed

```
mcmc.output <- nimbleMCMC(code = naive.survival.model,
                          data = my.data,
                          constants = my.constants,
                          inits = initVals,                 # pre-sampled initial values
                          monitors = parameters.to.save,
                          thin = n.thin,
                          niter = n.iter,
                          nburnin = n.burnin,
                          nchains = n.chains,
                          setSeed = mySeed)                 # defined seed
```

Class 4 live demo: Hidden Markov models and capture-recapture data

The team

last updated: 2021-04-23

Introduction

In this demo, we introduce Markov models and hidden Markov models following up on the survival example. Now we'll be using longitudinal data, that is data collected over time on the same individuals. We will also briefly see how to simulate data. Simulating data often proves useful to better understand how your model works, check that you get the right answer by comparing the parameters you used to simulate the data and the estimates you get, and to communicate with others on the model hypotheses and limitations.

Markov chain

Load some useful packages.

```
library(tidyverse)
```

We start with 57 individuals, and we monitor them over 5 occasions.

```
nind <- 57
nocc <- 5
```

For simplicity, we assume that we have a single cohort, that is all individuals enter the study at the same time, here on the first occasion.

```
first <- rep(1, nind) # single cohort
```

We set survival to 0.8, and define z as a matrix with dimensions the number of individuals (rows) and the number of occasions (columns).

```
phi <- 0.8
z <- matrix(NA, nrow = nind, ncol = nocc)
```

Now we simulate the fate of all individuals over time.

```
for (i in 1:nind){ # loop over individuals
  z[i,first[i]] <- 1 # all individuals are alive at first occasion
  for (t in (first[i]+1):nocc){ # loop over time
    z[i,t] <- rbinom(1, 1, phi * z[i,t-1]) # if z[i,t-1] = 1, then z[i,t] ~ dbern(phi)
                                           # if z[i,t-1] = 0, then z[i,t] ~ dbern(0) = 0
(once you're dead, you remain dead)
  }
}
```

The zeros are replaced by twos to match the coding proposed in the lecture. One is for alive, two is for dead.

```
z[z==0] <- 2 # 2 = dead, 1 = alive
```

Name the columns.

```
colnames(z) <- paste0("winter ", 1:nocc)
```

Display the matrix z.

```
z %>%
  as_tibble() %>%
  add_column(id = 1:nind, .before = "winter 1") %>%
  kableExtra::kable() %>%
  kableExtra::scroll_box(width = "100%", height = "400px")
```

id	winter 1	winter 2	winter 3	winter 4	winter 5
1	1	1	1	1	1
2	1	2	2	2	2
3	1	1	2	2	2
4	1	2	2	2	2
5	1	1	2	2	2
6	1	1	1	2	2
7	1	1	1	1	2
8	1	1	1	2	2
9	1	1	1	1	1
10	1	1	1	2	2
11	1	1	2	2	2
12	1	2	2	2	2
13	1	1	1	2	2
14	1	2	2	2	2
15	1	1	2	2	2
16	1	1	1	1	1
17	1	1	1	1	1
18	1	1	2	2	2
19	1	1	1	2	2

Now we're going to fit a Markov model to the data we've just simulated. We load nimble.

```
library(nimble)
```

Then we build the model.

```
markov.survival <- nimbleCode({
  phi ~ dunif(0, 1) # prior
  gamma[1,1] <- phi      # Pr(alive t -> alive t+1)
  gamma[1,2] <- 1 - phi  # Pr(alive t -> dead t+1)
  gamma[2,1] <- 0        # Pr(dead t -> alive t+1)
  gamma[2,2] <- 1        # Pr(dead t -> dead t+1)
  delta[1] <- 1          # Pr(alive t = 1) = 1
  delta[2] <- 0          # Pr(dead t = 1) = 0
  # likelihood
  for (i in 1:N){
    z[i,1] ~ dcat(delta[1:2])
    for (j in 2:T){
      z[i,j] ~ dcat(gamma[z[i,j-1], 1:2])
    }
  }})
```

]

We put the constants in a list.

```
my.constants <- list(N = 57, T = 5)
my.constants
```

```
$N
[1] 57

$T
[1] 5
```

We put the data in a list.

```
my.data <- list(z = z)
my.data
```

```
$z
      winter 1 winter 2 winter 3 winter 4 winter 5
 [1,]        1        1        1        1        1
 [2,]        1        2        2        2        2
 [3,]        1        1        2        2        2
 [4,]        1        2        2        2        2
 [5,]        1        1        2        2        2
 [6,]        1        1        1        2        2
 [7,]        1        1        1        1        2
 [8,]        1        1        1        2        2
 [9,]        1        1        1        1        1
[10,]        1        1        1        2        2
[11,]        1        1        2        2        2
[12,]        1        2        2        2        2
[13,]        1        1        1        2        2
[14,]        1        2        2        2        2
[15,]        1        1        2        2        2
[16,]        1        1        1        1        1
[17,]        1        1        1        1        1
[18,]        1        1        2        2        2
[19,]        1        1        1        2        2
[20,]        1        1        1        1        1
[21,]        1        1        1        1        1
[22,]        1        1        2        2        2
[23,]        1        2        2        2        2
[24,]        1        1        1        1        2
[25,]        1        1        1        2        2
[26,]        1        1        2        2        2
[27,]        1        1        1        1        1
[28,]        1        1        1        1        1
[29,]        1        1        1        1        1
[30,]        1        1        2        2        2
[31,]        1        1        1        1        1
[32,]        1        2        2        2        2
[33,]        1        1        1        1        1
[34,]        1        1        1        1        2
[35,]        1        1        2        2        2
[36,]        1        2        2        2        2
[37,]        1        1        2        2        2
[38,]        1        1        1        2        2
[39,]        1        1        1        1        1
[40,]        1        1        1        1        1
[41,]        1        1        1        1        1
[42,]        1        1        1        1        1
[43,]        1        1        1        1        2
[44,]        1        1        1        1        1
[45,]        1        2        2        2        2
[46,]        1        1        2        2        2
[47,]        1        1        1        1        2
[48,]        1        1        1        1        1
[49,]        1        2        2        2        2
[50,]        1        1        2        2        2
[51,]        1        1        1        2        2
```

```
[52,]        1        1        1        1        1
[53,]        1        2        2        2        2
[54,]        1        1        2        2        2
[55,]        1        1        1        1        1
[56,]        1        2        2        2        2
[57,]        1        1        1        1        1
```

We write a function that generates initial values for survival.

```
initial.values <- function() list(phi = runif(1,0,1))
initial.values()
```

```
$phi
[1] 0.208142
```

We specify that we'd like to monitor survival.

```
parameters.to.save <- c("phi")
parameters.to.save
```

```
[1] "phi"
```

Let's specify a few details about the chains.

```
n.iter <- 5000
n.burnin <- 1000
n.chains <- 2
```

And now, run nimble.

```
mcmc.output <- nimbleMCMC(code = markov.survival,
                          constants = my.constants,
                          data = my.data,
                          inits = initial.values,
                          monitors = parameters.to.save,
                          niter = n.iter,
                          nburnin = n.burnin,
                          nchains = n.chains)
```

```
|-------------|-------------|-------------|-------------|
|-----------------------------------------------------|
|-------------|-------------|-------------|-------------|
|-----------------------------------------------------|
```

```
library(MCMCvis)
MCMCsummary(mcmc.output, round = 2)
```

```
    mean   sd 2.5%  50% 97.5% Rhat n.eff
phi 0.77 0.03  0.7 0.77  0.83    1  1724
```

The posterior mean is close to the value of survival we used to simulate the data $\phi = 0.8$. Great!

Note that you should be able to write the model in a more efficient way with matrices and vectors.

```
markov.survival <- nimbleCode({
  phi ~ dunif(0, 1) # prior
  gamma[1:2,1:2] <- matrix(c(phi, 0, 1 - phi, 1), nrow = 2)
  delta[1:2] <- c(1, 0)
  # likelihood
  for (i in 1:N){
    z[i,1] ~ dcat(delta[1:2])
    for (j in 2:T){
      z[i,j] ~ dcat(gamma[z[i,j-1], 1:2])
    }
  }})
```

]

Hiden Markov chain

On top of the matrix of alive/dead states, we add the detection process. First we set the detection probability to $p = 0.6$.

```
p <- 0.6
```

Then we say for now that the matrix of detections and non-detections is just z in which dead individuals are not detected.

```
y <- z
y[z==2] <- 0
```

Now for alive individuals, those for which $y = z = 1$, we say they're detected with probability 1.

```
y[y==1] <- rbinom(n = sum(y==1), 1, p)
```

Let's get the number of individuals that have at least a detection.

```
nobs <- sum(apply(y,1,sum) != 0)
nobs
```

```
[1] 50
```

Before going any further, we need to get rid of the 7 individuals that have never been detected.

```
y <- y[apply(y,1,sum)!=0, ]
```

Let's get the occasion of first capture for each individual.

```
first <- apply(y, 1, function(x) min(which(x != 0)))
```

For convenience, we will replace the 0s before first detection by NAs.

```
for (i in 1:nobs){
  if(first[i] > 1) y[i, 1:(first[i]-1)] <- NA
}
y %>%
  as_tibble() %>%
  add_column(id = 1:nobs, .before = "winter 1") %>%
  kableExtra::kable() %>%
  kableExtra::scroll_box(width = "100%", height = "300px")
```

id	winter 1	winter 2	winter 3	winter 4	winter 5
1	1	1	0	1	0
2	1	0	0	0	0
3	1	0	0	0	0
4	1	0	0	0	0
5	1	0	0	0	0
6	1	0	1	0	0
7	1	1	0	0	0
8	NA	NA	1	1	0
9	1	0	1	0	0
10	1	1	0	0	0
11	1	0	0	0	0
12	1	1	1	0	0
13	1	1	0	0	0
14	1	1	1	1	0

Now we're ready to fit our HMM to the data we simulated. As usual, we first define the model.

```
hmm.survival <- nimbleCode({
  phi ~ dunif(0, 1) # prior survival
  p ~ dunif(0, 1) # prior detection
  # likelihood
  gamma[1,1] <- phi      # Pr(alive t -> alive t+1)
  gamma[1,2] <- 1 - phi  # Pr(alive t -> dead t+1)
  gamma[2,1] <- 0        # Pr(dead t -> alive t+1)
  gamma[2,2] <- 1        # Pr(dead t -> dead t+1)
  delta[1] <- 1          # Pr(alive t = 1) = 1
  delta[2] <- 0          # Pr(dead t = 1) = 0
  omega[1,1] <- 1 - p    # Pr(alive t -> non-detected t)
  omega[1,2] <- p        # Pr(alive t -> detected t)
  omega[2,1] <- 1        # Pr(dead t -> non-detected t)
  omega[2,2] <- 0        # Pr(dead t -> detected t)
  for (i in 1:N){
    z[i,first[i]] ~ dcat(delta[1:2]) # initial state prob
    for (j in (first[i]+1):T){
      z[i,j] ~ dcat(gamma[z[i,j-1], 1:2]) # z_t given z_(t-1)
      y[i,j] ~ dcat(omega[z[i,j], 1:2])   # y_t given z_t
    }
  }
})
```

Then put the constants in a list.

```
my.constants <- list(N = nrow(y),      # nb of individuals
                     T = 5,            # nb of occasions
                     first = first)    # occasions of first capture
my.constants
```

```
$N
[1] 50

$T
[1] 5

$first
 [1] 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 2 1 2 1 1 1 2 2 1 1 1 1 1 2 1 1 1 1 3 2 2 3
[39] 1 1 1 1 3 1 1 1 1 1 2 3
```

Now the data in a list. Note that we add 1 to the data to have 1 for non-detections and 2 for detections. You may use the coding you prefer of course, you will just need to adjust the Ω and Γ matrices in the model above.

```
my.data <- list(y = y + 1)
```

Specify initial values. For the latent states, we go for the easy way, and say that all individuals are alive through the study period.

```
zinits <- y + 1 # non-detection -> alive
zinits[zinits == 2] <- 1 # dead -> alive
initial.values <- function() list(phi = runif(1,0,1),
                                  p = runif(1,0,1),
                                  z = zinits)
```

Specify the parameters we'd like to monitor.

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

```
[1] "phi" "p"
```

MCMC detaisl.

```
n.iter <- 5000
n.burnin <- 1000
n.chains <- 2
```

At last, let's run nimble.

```
mcmc.output <- nimbleMCMC(code = hmm.survival,                # model code
                          constants = my.constants,           # constants
                          data = my.data,                     # data
                          inits = initial.values,             # initial values
                          monitors = parameters.to.save,      # parameters to monitor
                          niter = n.iter,                     # nb of iterations
                          nburnin = n.burnin,                 # length of the burn-in per
iod
                          nchains = n.chains)                 # nb of chains
```

```
|-------------|-------------|-------------|-------------|
|-----------------------------------------------------|
|-------------|-------------|-------------|-------------|
|-----------------------------------------------------|
```

We display the results. The posterior means of detection and survival are close to the parameters we used to simulate the data.

```
library(MCMCvis)
MCMCsummary(mcmc.output, round = 2)
```

```
    mean   sd 2.5%  50% 97.5% Rhat n.eff
p   0.54 0.07 0.41 0.54  0.68 1.03   447
phi 0.77 0.05 0.68 0.77  0.87 1.01   576
```

Class 5 live demo: Survival estimation

The team

last updated: 2021-05-15

Introduction

We are going to analyze real capture-recapture data. The data concern the European Dipper (Cinclus cinclus). Captures were carried out for 7 years (1981-1987) in eastern France by Gilbert Marzolin who kindly provided us with the data. The data consist of initial markings and recaptures of almost 300 breeding adults each year during the March-June period. Birds were at least 1 year old when initially banded.

Load nimble first.

```
library(nimble)
```

Data

Read in the data.

```
dipper <- read_csv("dipper.csv")
dipper %>%
  kableExtra::kable() %>%
  kableExtra::scroll_box(width = "100%", height = "400px")
```

year_1981	year_1982	year_1983	year_1984	year_1985	year_1986	year_1987	sex	wing_length
1	1	1	1	1	1	0	M	95
1	1	1	1	1	0	0	F	88
1	1	1	1	0	0	0	M	94
1	1	1	1	0	0	0	F	85
1	1	0	1	1	1	0	F	86
1	1	0	0	0	0	0	M	97
1	1	0	0	0	0	0	M	96
1	1	0	0	0	0	0	M	98
1	1	0	0	0	0	0	M	96
1	1	0	0	0	0	0	F	89
1	1	0	0	0	0	0	F	86
1	0	1	0	0	0	0	M	98
1	0	1	0	0	0	0	F	92
1	0	0	0	0	0	0	M	97
1	0	0	0	0	0	0	M	96
1	0	0	0	0	0	0	M	95
1	0	0	0	0	0	0	M	98
1	0	0	0	0	0	0	M	96
1	0	0	0	0	0	0	F	91

Format the data.

```
y <- dipper %>%
  select(year_1981:year_1987) %>%
  as.matrix()
head(y)
```

```
     year_1981 year_1982 year_1983 year_1984 year_1985 year_1986 year_1987
[1,]         1         1         1         1         1         1         0
[2,]         1         1         1         1         1         0         0
[3,]         1         1         1         1         0         0         0
[4,]         1         1         1         1         0         0         0
[5,]         1         1         0         1         1         1         0
[6,]         1         1         0         0         0         0         0
```

CJS models

We start the session by fitting models with or without a time effect on survival and recapture probabilities.

A model with constant parameters.

```
hmm.phip <- nimbleCode({
  phi ~ dunif(0, 1) # prior survival
  p ~ dunif(0, 1) # prior detection
  # likelihood
  gamma[1,1] <- phi      # Pr(alive t -> alive t+1)
  gamma[1,2] <- 1 - phi  # Pr(alive t -> dead t+1)
  gamma[2,1] <- 0        # Pr(dead t -> alive t+1)
  gamma[2,2] <- 1        # Pr(dead t -> dead t+1)
  delta[1] <- 1          # Pr(alive t = 1) = 1
  delta[2] <- 0          # Pr(dead t = 1) = 0
  omega[1,1] <- 1 - p    # Pr(alive t -> non-detected t)
  omega[1,2] <- p        # Pr(alive t -> detected t)
  omega[2,1] <- 1        # Pr(dead t -> non-detected t)
  omega[2,2] <- 0        # Pr(dead t -> detected t)
  for (i in 1:N){
    z[i,first[i]] ~ dcat(delta[1:2])
    for (j in (first[i]+1):T){
      z[i,j] ~ dcat(gamma[z[i,j-1], 1:2])
      y[i,j] ~ dcat(omega[z[i,j], 1:2])
    }
  }
})
```

Get the occasion of first capture for all individuals.

```
first <- apply(y, 1, function(x) min(which(x !=0)))
```

A list with constants.

```
my.constants <- list(N = nrow(y),
                     T = ncol(y),
                     first = first)
my.constants
```

```
$N
[1] 294

$T
[1] 7

$first
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [38] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3
 [75] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
[112] 3 3 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
[149] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5
[186] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 6 6
[223] 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 7 7 7 7
[260] 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
```

Now the data in a list. Note that we add 1 to the data to have 1 for non-detections and 2 for detections. You may use the coding you prefer of course, you will just need to adjust the Ω and Γ matrices in the model above.

```
my.data <- list(y = y + 1)
```

Specify initial values. For the latent states, we go for the easy way, and say that all individuals are alive through the study period.

```
zinits <- y + 1 # non-detection -> alive
zinits[zinits == 2] <- 1 # dead -> alive
initial.values <- function() list(phi = runif(1,0,1),
                                  p = runif(1,0,1),
                                  z = zinits)
initial.values()
```

```
$phi
[1] 0.03299212

$p
[1] 0.1289726

$z
       year_1981 year_1982 year_1983 year_1984 year_1985 year_1986 year_1987
  [1,]         1         1         1         1         1         1         1
  [2,]         1         1         1         1         1         1         1
  [3,]         1         1         1         1         1         1         1
  [4,]         1         1         1         1         1         1         1
  [5,]         1         1         1         1         1         1         1
  [6,]         1         1         1         1         1         1         1
  [7,]         1         1         1         1         1         1         1
  [8,]         1         1         1         1         1         1         1
  [9,]         1         1         1         1         1         1         1
 [10,]         1         1         1         1         1         1         1
 [11,]         1         1         1         1         1         1         1
 [12,]         1         1         1         1         1         1         1
 [13,]         1         1         1         1         1         1         1
 [14,]         1         1         1         1         1         1         1
 [15,]         1         1         1         1         1         1         1
 [16,]         1         1         1         1         1         1         1
 [17,]         1         1         1         1         1         1         1
 [18,]         1         1         1         1         1         1         1
 [19,]         1         1         1         1         1         1         1
 [20,]         1         1         1         1         1         1         1
 [21,]         1         1         1         1         1         1         1
 [22,]         1         1         1         1         1         1         1
 [23,]         1         1         1         1         1         1         1
 [24,]         1         1         1         1         1         1         1
 [25,]         1         1         1         1         1         1         1
 [26,]         1         1         1         1         1         1         1
 [27,]         1         1         1         1         1         1         1
 [28,]         1         1         1         1         1         1         1
 [29,]         1         1         1         1         1         1         1
 [30,]         1         1         1         1         1         1         1
 [31,]         1         1         1         1         1         1         1
 [32,]         1         1         1         1         1         1         1
 [33,]         1         1         1         1         1         1         1
 [34,]         1         1         1         1         1         1         1
 [35,]         1         1         1         1         1         1         1
 [36,]         1         1         1         1         1         1         1
 [37,]         1         1         1         1         1         1         1
 [38,]         1         1         1         1         1         1         1
 [39,]         1         1         1         1         1         1         1
 [40,]         1         1         1         1         1         1         1
 [41,]         1         1         1         1         1         1         1
 [42,]         1         1         1         1         1         1         1
 [43,]         1         1         1         1         1         1         1
 [44,]         1         1         1         1         1         1         1
 [45,]         1         1         1         1         1         1         1
```

```
[46,]    1    1    1    1    1    1    1
[47,]    1    1    1    1    1    1    1
[48,]    1    1    1    1    1    1    1
[49,]    1    1    1    1    1    1    1
[50,]    1    1    1    1    1    1    1
[51,]    1    1    1    1    1    1    1
[52,]    1    1    1    1    1    1    1
[53,]    1    1    1    1    1    1    1
[54,]    1    1    1    1    1    1    1
[55,]    1    1    1    1    1    1    1
[56,]    1    1    1    1    1    1    1
[57,]    1    1    1    1    1    1    1
[58,]    1    1    1    1    1    1    1
[59,]    1    1    1    1    1    1    1
[60,]    1    1    1    1    1    1    1
[61,]    1    1    1    1    1    1    1
[62,]    1    1    1    1    1    1    1
[63,]    1    1    1    1    1    1    1
[64,]    1    1    1    1    1    1    1
[65,]    1    1    1    1    1    1    1
[66,]    1    1    1    1    1    1    1
[67,]    1    1    1    1    1    1    1
[68,]    1    1    1    1    1    1    1
[69,]    1    1    1    1    1    1    1
[70,]    1    1    1    1    1    1    1
[71,]    1    1    1    1    1    1    1
[72,]    1    1    1    1    1    1    1
[73,]    1    1    1    1    1    1    1
[74,]    1    1    1    1    1    1    1
[75,]    1    1    1    1    1    1    1
[76,]    1    1    1    1    1    1    1
[77,]    1    1    1    1    1    1    1
[78,]    1    1    1    1    1    1    1
[79,]    1    1    1    1    1    1    1
[80,]    1    1    1    1    1    1    1
[81,]    1    1    1    1    1    1    1
[82,]    1    1    1    1    1    1    1
[83,]    1    1    1    1    1    1    1
[84,]    1    1    1    1    1    1    1
[85,]    1    1    1    1    1    1    1
[86,]    1    1    1    1    1    1    1
[87,]    1    1    1    1    1    1    1
[88,]    1    1    1    1    1    1    1
[89,]    1    1    1    1    1    1    1
[90,]    1    1    1    1    1    1    1
[91,]    1    1    1    1    1    1    1
[92,]    1    1    1    1    1    1    1
[93,]    1    1    1    1    1    1    1
[94,]    1    1    1    1    1    1    1
[95,]    1    1    1    1    1    1    1
[96,]    1    1    1    1    1    1    1
[97,]    1    1    1    1    1    1    1
[98,]    1    1    1    1    1    1    1
[99,]    1    1    1    1    1    1    1
```

```
[100,]    1    1    1    1    1    1    1
[101,]    1    1    1    1    1    1    1
[102,]    1    1    1    1    1    1    1
[103,]    1    1    1    1    1    1    1
[104,]    1    1    1    1    1    1    1
[105,]    1    1    1    1    1    1    1
[106,]    1    1    1    1    1    1    1
[107,]    1    1    1    1    1    1    1
[108,]    1    1    1    1    1    1    1
[109,]    1    1    1    1    1    1    1
[110,]    1    1    1    1    1    1    1
[111,]    1    1    1    1    1    1    1
[112,]    1    1    1    1    1    1    1
[113,]    1    1    1    1    1    1    1
[114,]    1    1    1    1    1    1    1
[115,]    1    1    1    1    1    1    1
[116,]    1    1    1    1    1    1    1
[117,]    1    1    1    1    1    1    1
[118,]    1    1    1    1    1    1    1
[119,]    1    1    1    1    1    1    1
[120,]    1    1    1    1    1    1    1
[121,]    1    1    1    1    1    1    1
[122,]    1    1    1    1    1    1    1
[123,]    1    1    1    1    1    1    1
[124,]    1    1    1    1    1    1    1
[125,]    1    1    1    1    1    1    1
[126,]    1    1    1    1    1    1    1
[127,]    1    1    1    1    1    1    1
[128,]    1    1    1    1    1    1    1
[129,]    1    1    1    1    1    1    1
[130,]    1    1    1    1    1    1    1
[131,]    1    1    1    1    1    1    1
[132,]    1    1    1    1    1    1    1
[133,]    1    1    1    1    1    1    1
[134,]    1    1    1    1    1    1    1
[135,]    1    1    1    1    1    1    1
[136,]    1    1    1    1    1    1    1
[137,]    1    1    1    1    1    1    1
[138,]    1    1    1    1    1    1    1
[139,]    1    1    1    1    1    1    1
[140,]    1    1    1    1    1    1    1
[141,]    1    1    1    1    1    1    1
[142,]    1    1    1    1    1    1    1
[143,]    1    1    1    1    1    1    1
[144,]    1    1    1    1    1    1    1
[145,]    1    1    1    1    1    1    1
[146,]    1    1    1    1    1    1    1
[147,]    1    1    1    1    1    1    1
[148,]    1    1    1    1    1    1    1
[149,]    1    1    1    1    1    1    1
[150,]    1    1    1    1    1    1    1
[151,]    1    1    1    1    1    1    1
[152,]    1    1    1    1    1    1    1
[153,]    1    1    1    1    1    1    1
```

[154,]	1	1	1	1	1	1	1
[155,]	1	1	1	1	1	1	1
[156,]	1	1	1	1	1	1	1
[157,]	1	1	1	1	1	1	1
[158,]	1	1	1	1	1	1	1
[159,]	1	1	1	1	1	1	1
[160,]	1	1	1	1	1	1	1
[161,]	1	1	1	1	1	1	1
[162,]	1	1	1	1	1	1	1
[163,]	1	1	1	1	1	1	1
[164,]	1	1	1	1	1	1	1
[165,]	1	1	1	1	1	1	1
[166,]	1	1	1	1	1	1	1
[167,]	1	1	1	1	1	1	1
[168,]	1	1	1	1	1	1	1
[169,]	1	1	1	1	1	1	1
[170,]	1	1	1	1	1	1	1
[171,]	1	1	1	1	1	1	1
[172,]	1	1	1	1	1	1	1
[173,]	1	1	1	1	1	1	1
[174,]	1	1	1	1	1	1	1
[175,]	1	1	1	1	1	1	1
[176,]	1	1	1	1	1	1	1
[177,]	1	1	1	1	1	1	1
[178,]	1	1	1	1	1	1	1
[179,]	1	1	1	1	1	1	1
[180,]	1	1	1	1	1	1	1
[181,]	1	1	1	1	1	1	1
[182,]	1	1	1	1	1	1	1
[183,]	1	1	1	1	1	1	1
[184,]	1	1	1	1	1	1	1
[185,]	1	1	1	1	1	1	1
[186,]	1	1	1	1	1	1	1
[187,]	1	1	1	1	1	1	1
[188,]	1	1	1	1	1	1	1
[189,]	1	1	1	1	1	1	1
[190,]	1	1	1	1	1	1	1
[191,]	1	1	1	1	1	1	1
[192,]	1	1	1	1	1	1	1
[193,]	1	1	1	1	1	1	1
[194,]	1	1	1	1	1	1	1
[195,]	1	1	1	1	1	1	1
[196,]	1	1	1	1	1	1	1
[197,]	1	1	1	1	1	1	1
[198,]	1	1	1	1	1	1	1
[199,]	1	1	1	1	1	1	1
[200,]	1	1	1	1	1	1	1
[201,]	1	1	1	1	1	1	1
[202,]	1	1	1	1	1	1	1
[203,]	1	1	1	1	1	1	1
[204,]	1	1	1	1	1	1	1
[205,]	1	1	1	1	1	1	1
[206,]	1	1	1	1	1	1	1
[207,]	1	1	1	1	1	1	1

```
[208,]    1    1    1    1    1    1    1
[209,]    1    1    1    1    1    1    1
[210,]    1    1    1    1    1    1    1
[211,]    1    1    1    1    1    1    1
[212,]    1    1    1    1    1    1    1
[213,]    1    1    1    1    1    1    1
[214,]    1    1    1    1    1    1    1
[215,]    1    1    1    1    1    1    1
[216,]    1    1    1    1    1    1    1
[217,]    1    1    1    1    1    1    1
[218,]    1    1    1    1    1    1    1
[219,]    1    1    1    1    1    1    1
[220,]    1    1    1    1    1    1    1
[221,]    1    1    1    1    1    1    1
[222,]    1    1    1    1    1    1    1
[223,]    1    1    1    1    1    1    1
[224,]    1    1    1    1    1    1    1
[225,]    1    1    1    1    1    1    1
[226,]    1    1    1    1    1    1    1
[227,]    1    1    1    1    1    1    1
[228,]    1    1    1    1    1    1    1
[229,]    1    1    1    1    1    1    1
[230,]    1    1    1    1    1    1    1
[231,]    1    1    1    1    1    1    1
[232,]    1    1    1    1    1    1    1
[233,]    1    1    1    1    1    1    1
[234,]    1    1    1    1    1    1    1
[235,]    1    1    1    1    1    1    1
[236,]    1    1    1    1    1    1    1
[237,]    1    1    1    1    1    1    1
[238,]    1    1    1    1    1    1    1
[239,]    1    1    1    1    1    1    1
[240,]    1    1    1    1    1    1    1
[241,]    1    1    1    1    1    1    1
[242,]    1    1    1    1    1    1    1
[243,]    1    1    1    1    1    1    1
[244,]    1    1    1    1    1    1    1
[245,]    1    1    1    1    1    1    1
[246,]    1    1    1    1    1    1    1
[247,]    1    1    1    1    1    1    1
[248,]    1    1    1    1    1    1    1
[249,]    1    1    1    1    1    1    1
[250,]    1    1    1    1    1    1    1
[251,]    1    1    1    1    1    1    1
[252,]    1    1    1    1    1    1    1
[253,]    1    1    1    1    1    1    1
[254,]    1    1    1    1    1    1    1
[255,]    1    1    1    1    1    1    1
[256,]    1    1    1    1    1    1    1
[257,]    1    1    1    1    1    1    1
[258,]    1    1    1    1    1    1    1
[259,]    1    1    1    1    1    1    1
[260,]    1    1    1    1    1    1    1
[261,]    1    1    1    1    1    1    1
```

```
[262,]    1    1    1    1    1    1    1
[263,]    1    1    1    1    1    1    1
[264,]    1    1    1    1    1    1    1
[265,]    1    1    1    1    1    1    1
[266,]    1    1    1    1    1    1    1
[267,]    1    1    1    1    1    1    1
[268,]    1    1    1    1    1    1    1
[269,]    1    1    1    1    1    1    1
[270,]    1    1    1    1    1    1    1
[271,]    1    1    1    1    1    1    1
[272,]    1    1    1    1    1    1    1
[273,]    1    1    1    1    1    1    1
[274,]    1    1    1    1    1    1    1
[275,]    1    1    1    1    1    1    1
[276,]    1    1    1    1    1    1    1
[277,]    1    1    1    1    1    1    1
[278,]    1    1    1    1    1    1    1
[279,]    1    1    1    1    1    1    1
[280,]    1    1    1    1    1    1    1
[281,]    1    1    1    1    1    1    1
[282,]    1    1    1    1    1    1    1
[283,]    1    1    1    1    1    1    1
[284,]    1    1    1    1    1    1    1
[285,]    1    1    1    1    1    1    1
[286,]    1    1    1    1    1    1    1
[287,]    1    1    1    1    1    1    1
[288,]    1    1    1    1    1    1    1
[289,]    1    1    1    1    1    1    1
[290,]    1    1    1    1    1    1    1
[291,]    1    1    1    1    1    1    1
[292,]    1    1    1    1    1    1    1
[293,]    1    1    1    1    1    1    1
[294,]    1    1    1    1    1    1    1
```

Some information that we now pass as initial value info (observations of alive) are actually known states, and could also be passed as data – in which case the initial values have to be 0.

Specify the parameters we wish to monitor.

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

```
[1] "phi" "p"
```

MCMC details.

```
n.iter <- 2500
n.burnin <- 1000
n.chains <- 2
```

At last, let's run nimble.

```
mcmc.phip <- nimbleMCMC(code = hmm.phip,
                        constants = my.constants,
                        data = my.data,
                        inits = initial.values,
                        monitors = parameters.to.save,
                        niter = n.iter,
                        nburnin = n.burnin,
                        nchains = n.chains)
```

```
defining model...
```

```
building model...
```

```
setting data and initial values...
```

```
running calculate on model (any error reports that follow may simply reflect missing val
ues in model variables) ...
checking model sizes and dimensions...
checking model calculations...
model building finished.
compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilat
ion details.
compilation finished.
running chain 1...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

```
running chain 2...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

Examine the results.

```
library(MCMCvis)
MCMCsummary(mcmc.phip, round = 2)
```

```
    mean   sd 2.5%  50% 97.5% Rhat n.eff
p   0.89 0.03 0.83 0.90  0.94 1.01   262
phi 0.56 0.03 0.52 0.56  0.61 1.01   489
```

Now a model with time-varying survival probabilities, and constant detection.

```
hmm.phitp <- nimbleCode({
  for (t in 1:(T-1)){
    phi[t] ~ dunif(0, 1) # prior survival
    gamma[1,1,t] <- phi[t]      # Pr(alive t -> alive t+1)
    gamma[1,2,t] <- 1 - phi[t]  # Pr(alive t -> dead t+1)
    gamma[2,1,t] <- 0        # Pr(dead t -> alive t+1)
    gamma[2,2,t] <- 1        # Pr(dead t -> dead t+1)
  }
  p ~ dunif(0, 1) # prior detection
  delta[1] <- 1          # Pr(alive t = 1) = 1
  delta[2] <- 0          # Pr(dead t = 1) = 0
  omega[1,1] <- 1 - p    # Pr(alive t -> non-detected t)
  omega[1,2] <- p        # Pr(alive t -> detected t)
  omega[2,1] <- 1        # Pr(dead t -> non-detected t)
  omega[2,2] <- 0        # Pr(dead t -> detected t)
  # likelihood
  for (i in 1:N){
    z[i,first[i]] ~ dcat(delta[1:2])
    for (j in (first[i]+1):T){
      z[i,j] ~ dcat(gamma[z[i,j-1], 1:2, j-1])
      y[i,j] ~ dcat(omega[z[i,j], 1:2])
    }
  }
})
```

The initial values.

```
initial.values <- function() list(phi = runif(my.constants$T-1,0,1),
                                  p = runif(1,0,1),
                                  z = zinits)
```

Run nimble.

```
mcmc.phitp <- nimbleMCMC(code = hmm.phitp,
                  constants = my.constants,
                  data = my.data,
                  inits = initial.values,
                  monitors = parameters.to.save,
                  niter = n.iter,
                  nburnin = n.burnin,
                  nchains = n.chains)
```

```
defining model...
```

```
building model...
```

```
setting data and initial values...
```

```
running calculate on model (any error reports that follow may simply reflect missing val
ues in model variables) ...
checking model sizes and dimensions...
checking model calculations...
model building finished.
compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilat
ion details.
compilation finished.
running chain 1...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

```
running chain 2...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

Display the results.

```
MCMCsummary(object = mcmc.phitp, round = 2)
```

```
       mean   sd 2.5%  50% 97.5% Rhat n.eff
p      0.89 0.03 0.82 0.89  0.94 1.02   255
phi[1] 0.62 0.11 0.41 0.62  0.81 1.01   567
phi[2] 0.46 0.06 0.34 0.46  0.59 1.00   652
phi[3] 0.49 0.06 0.37 0.49  0.60 1.00   561
phi[4] 0.62 0.06 0.51 0.62  0.73 1.01   572
phi[5] 0.61 0.05 0.50 0.61  0.71 1.01   546
phi[6] 0.59 0.06 0.48 0.58  0.70 1.02   428
```

Now a model with time-varying detection and constant survival.

```
hmm.phipt <- nimbleCode({
  phi ~ dunif(0, 1) # prior survival
  gamma[1,1] <- phi      # Pr(alive t -> alive t+1)
  gamma[1,2] <- 1 - phi  # Pr(alive t -> dead t+1)
  gamma[2,1] <- 0        # Pr(dead t -> alive t+1)
  gamma[2,2] <- 1        # Pr(dead t -> dead t+1)
  delta[1] <- 1          # Pr(alive t = 1) = 1
  delta[2] <- 0          # Pr(dead t = 1) = 0
  for (t in 1:(T-1)){
    p[t] ~ dunif(0, 1) # prior detection
    omega[1,1,t] <- 1 - p[t]    # Pr(alive t -> non-detected t)
    omega[1,2,t] <- p[t]        # Pr(alive t -> detected t)
    omega[2,1,t] <- 1        # Pr(dead t -> non-detected t)
    omega[2,2,t] <- 0        # Pr(dead t -> detected t)
  }
  # likelihood
  for (i in 1:N){
    z[i,first[i]] ~ dcat(delta[1:2])
    for (j in (first[i]+1):T){
      z[i,j] ~ dcat(gamma[z[i,j-1], 1:2])
      y[i,j] ~ dcat(omega[z[i,j], 1:2, j-1])
    }
  }
})
```

Initial values.

```
initial.values <- function() list(phi = runif(1,0,1),
                                  p = runif(my.constants$T-1,0,1),
                                  z = zinits)
```

Run nimble.

```
mcmc.phipt <- nimbleMCMC(code = hmm.phipt,
                         constants = my.constants,
                         data = my.data,
                         inits = initial.values,
                         monitors = parameters.to.save,
                         niter = n.iter,
                         nburnin = n.burnin,
                         nchains = n.chains)
```

```
defining model...
```

```
building model...
```

```
setting data and initial values...
```

```
running calculate on model (any error reports that follow may simply reflect missing val
ues in model variables) ...
checking model sizes and dimensions...
checking model calculations...
model building finished.
compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilat
ion details.
compilation finished.
running chain 1...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

```
running chain 2...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

Display the results.

```
MCMCsummary(object = mcmc.phipt, round = 2)
```

```
     mean   sd 2.5%  50% 97.5% Rhat n.eff
p[1] 0.75 0.11 0.50 0.76  0.93 1.03   540
p[2] 0.86 0.08 0.67 0.87  0.98 1.02   393
p[3] 0.84 0.07 0.70 0.85  0.97 1.01   400
p[4] 0.89 0.05 0.78 0.89  0.96 1.01   432
p[5] 0.91 0.04 0.82 0.92  0.98 1.04   381
p[6] 0.90 0.07 0.75 0.92  0.99 1.01   161
phi  0.56 0.03 0.51 0.56  0.61 1.00   522
```

Eventually, the CJS model with both time-varying survival and recapture probabilities.

```
hmm.phitpt <- nimbleCode({
  delta[1] <- 1                    # Pr(alive t = 1) = 1
  delta[2] <- 0                    # Pr(dead t = 1) = 0
  for (t in 1:(T-1)){ # loop over time
    phi[t] ~ dunif(0, 1)           # prior survival
    gamma[1,1,t] <- phi[t]         # Pr(alive t -> alive t+1)
    gamma[1,2,t] <- 1 - phi[t]     # Pr(alive t -> dead t+1)
    gamma[2,1,t] <- 0              # Pr(dead t -> alive t+1)
    gamma[2,2,t] <- 1              # Pr(dead t -> dead t+1)
    p[t] ~ dunif(0, 1)             # prior detection
    omega[1,1,t] <- 1 - p[t]       # Pr(alive t -> non-detected t)
    omega[1,2,t] <- p[t]           # Pr(alive t -> detected t)
    omega[2,1,t] <- 1              # Pr(dead t -> non-detected t)
    omega[2,2,t] <- 0              # Pr(dead t -> detected t)
  }
  # likelihood
  for (i in 1:N){
    z[i,first[i]] ~ dcat(delta[1:2])
    for (j in (first[i]+1):T){
      z[i,j] ~ dcat(gamma[z[i,j-1], 1:2, j-1])
      y[i,j] ~ dcat(omega[z[i,j], 1:2, j-1])
    }
  }
})
```

Initial values.

```
initial.values <- function() list(phi = runif(my.constants$T-1,0,1),
                                  p = runif(my.constants$T-1,0,1),
                                  z = zinits)
```

Run nimble.

```
mcmc.phitpt <- nimbleMCMC(code = hmm.phitpt,
                          constants = my.constants,
                          data = my.data,
                          inits = initial.values,
                          monitors = parameters.to.save,
                          niter = n.iter,
                          nburnin = n.burnin,
                          nchains = n.chains)
```

```
defining model...
```

```
building model...
```

```
setting data and initial values...
```

```
running calculate on model (any error reports that follow may simply reflect missing val
ues in model variables) ...
checking model sizes and dimensions...
checking model calculations...
model building finished.
compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilat
ion details.
compilation finished.
running chain 1...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

```
running chain 2...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

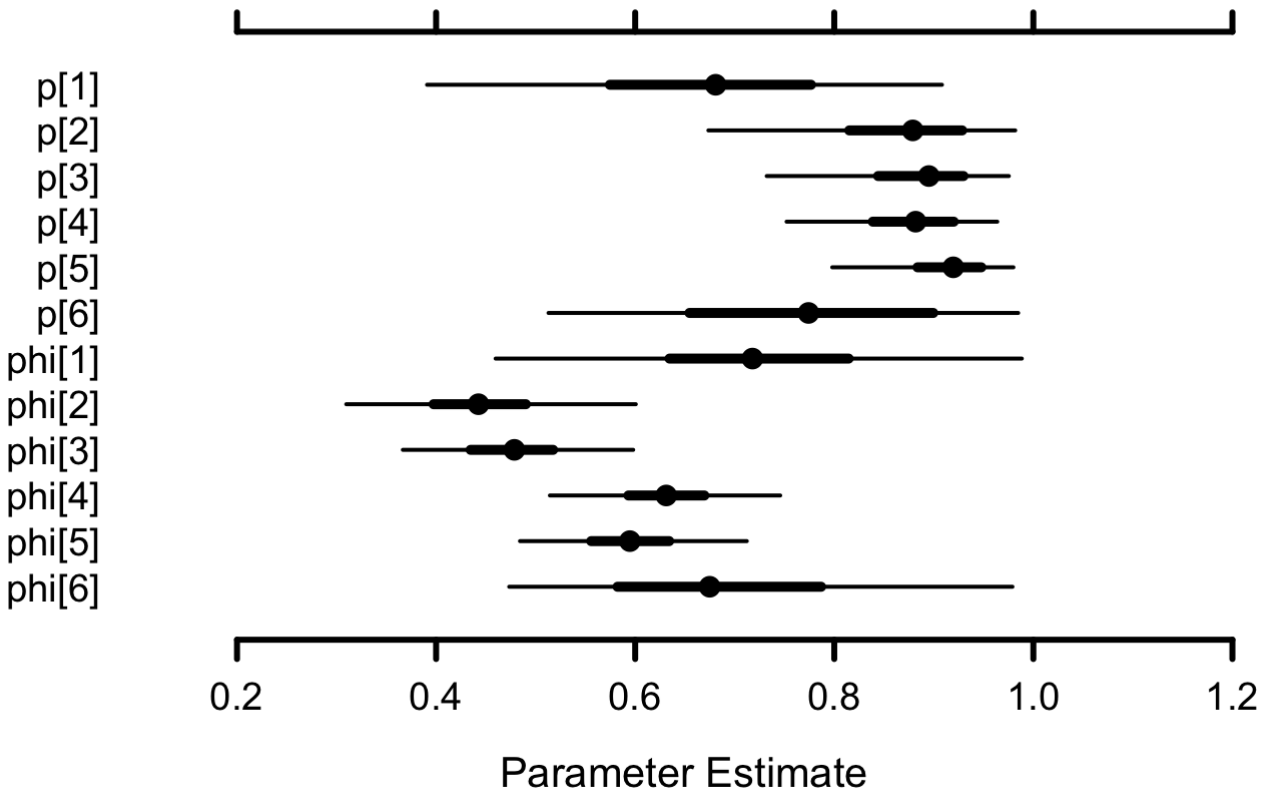
Display the numerical summaries. Note the small effective sample size for the last survival and recapture probabilities.

```
MCMCsummary(object = mcmc.phitpt, round = 2)
```

```
       mean   sd 2.5%  50% 97.5% Rhat n.eff
p[1]   0.67 0.14 0.39 0.68  0.91 1.01   280
p[2]   0.87 0.08 0.67 0.88  0.98 1.01   269
p[3]   0.88 0.06 0.73 0.90  0.98 1.02   294
p[4]   0.88 0.06 0.75 0.88  0.96 1.01   305
p[5]   0.91 0.05 0.80 0.92  0.98 1.02   234
p[6]   0.77 0.14 0.51 0.77  0.98 2.13    40
phi[1] 0.72 0.13 0.46 0.72  0.99 1.01   246
phi[2] 0.45 0.07 0.31 0.44  0.60 1.01   457
phi[3] 0.48 0.06 0.37 0.48  0.60 1.00   470
phi[4] 0.63 0.06 0.51 0.63  0.75 1.00   492
phi[5] 0.60 0.06 0.48 0.59  0.71 1.03   457
phi[6] 0.69 0.14 0.47 0.67  0.98 2.10    36
```

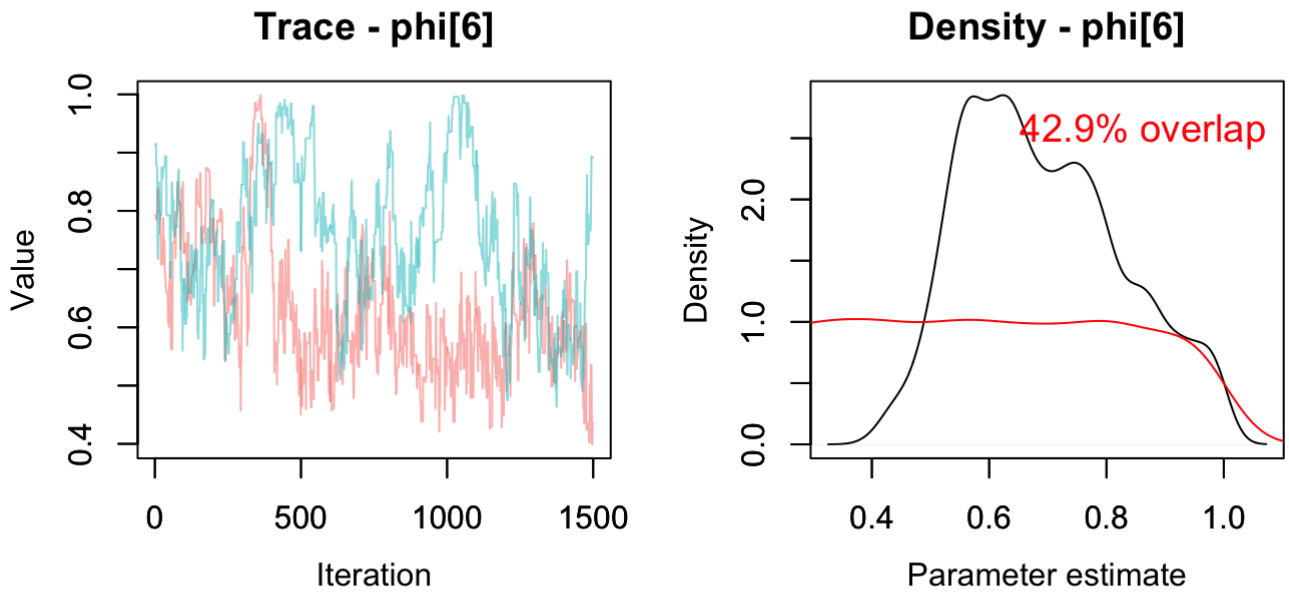
Caterpillar plot of the estimates.

```
MCMCplot(object = mcmc.phitpt)
```



Let's focus for a minute on the last survival probability. See how mixing is bad and the overlap with the prior is big. This parameter is redundant, and it can be shown that only the product of ϕ_6 and p_7 can be estimated.

```
priors <- runif(3000, 0, 1)
MCMCtrace(object = mcmc.phitpt,
          ISB = FALSE,
          exact = TRUE,
          params = c("phi[6]"),
          pdf = FALSE,
          priors = priors)
```



Model selection with WAIC

We re-run the four models above, but now we make sure we monitor the z and set the WAIC argument to `TRUE` in `nimbleMCMC()`.

```
my.constants <- list(N = nrow(y), T = ncol(y), first = first)
my.data <- list(y = y + 1)
initial.values <- function() list(phi = runif(1,0,1),
                                  p = runif(1,0,1),
                                  z = zinits)
parameters.to.save <- c("phi", "p", "z")
mcmc.phip <- nimbleMCMC(code = hmm.phip,
                        constants = my.constants,
                        data = my.data,
                        inits = initial.values,
                        monitors = parameters.to.save,
                        niter = n.iter,
                        nburnin = n.burnin,
                        nchains = n.chains,
                        WAIC = TRUE)
```

```
defining model...
```

```
building model...
```

```
setting data and initial values...
```

```
running calculate on model (any error reports that follow may simply reflect missing val
ues in model variables) ...
checking model sizes and dimensions...
checking model calculations...
model building finished.
Monitored nodes are valid for WAIC.
compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilat
ion details.
compilation finished.
running chain 1...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

```
running chain 2...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

```
initial.values <- function() list(phi = runif(my.constants$T-1,0,1),
                                  p = runif(1,0,1),
                                  z = zinits)
mcmc.phitp <- nimbleMCMC(code = hmm.phitp,
                         constants = my.constants,
                         data = my.data,
                         inits = initial.values,
                         monitors = parameters.to.save,
                         niter = n.iter,
                         nburnin = n.burnin,
                         nchains = n.chains,
                         WAIC = TRUE)
```

```
defining model...
building model...
setting data and initial values...
running calculate on model (any error reports that follow may simply reflect missing val
ues in model variables) ...
checking model sizes and dimensions...
checking model calculations...
model building finished.
Monitored nodes are valid for WAIC.
compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilat
ion details.
compilation finished.
running chain 1...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

```
running chain 2...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

```
initial.values <- function() list(phi = runif(1,0,1),
                                  p = runif(my.constants$T-1,0,1),
                                  z = zinits)
mcmc.phipt <- nimbleMCMC(code = hmm.phipt,
                         constants = my.constants,
                         data = my.data,
                         inits = initial.values,
                         monitors = parameters.to.save,
                         niter = n.iter,
                         nburnin = n.burnin,
                         nchains = n.chains,
                         WAIC = TRUE)
```

```
defining model...
building model...
setting data and initial values...
running calculate on model (any error reports that follow may simply reflect missing val
ues in model variables) ...
checking model sizes and dimensions...
checking model calculations...
model building finished.
Monitored nodes are valid for WAIC.
compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilat
ion details.
compilation finished.
running chain 1...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

```
running chain 2...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

```
initial.values <- function() list(phi = runif(my.constants$T-1,0,1),
                                  p = runif(my.constants$T-1,0,1),
                                  z = zinits)
mcmc.phitpt <- nimbleMCMC(code = hmm.phitpt,
                          constants = my.constants,
                          data = my.data,
                          inits = initial.values,
                          monitors = parameters.to.save,
                          niter = n.iter,
                          nburnin = n.burnin,
                          nchains = n.chains,
                          WAIC = TRUE)
```

```
defining model...
building model...
setting data and initial values...
running calculate on model (any error reports that follow may simply reflect missing val
ues in model variables) ...
checking model sizes and dimensions...
checking model calculations...
model building finished.
Monitored nodes are valid for WAIC.
compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilat
ion details.
compilation finished.
running chain 1...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

```
running chain 2...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

Now we report model ranking.

```
data.frame(model = c("(phi,p)",
                     "(phit,p)",
                     "(phi,pt)",
                     "(phit,pt)"),
           WAIC = c(mcmc.phip$WAIC,
             mcmc.phitp$WAIC,
             mcmc.phipt$WAIC,
             mcmc.phitpt$WAIC))
```

```
      model     WAIC
1   (phi,p) 267.4950
2  (phit,p) 274.3351
3  (phi,pt) 269.4981
4 (phit,pt) 311.2689
```

Add a temporal covariate

Now we'd like to add a temporal covariate to try and explain annual variation in survival. We pick water flow in river. We specify the relationship on the logit scale.

```
hmm.phiflowp <- nimbleCode({
  delta[1] <- 1           # Pr(alive t = 1) = 1
  delta[2] <- 0           # Pr(dead t = 1) = 0
  for (t in 1:(T-1)){
    logit(phi[t]) <- beta[1] + beta[2] * flow[t]
    gamma[1,1,t] <- phi[t]      # Pr(alive t -> alive t+1)
    gamma[1,2,t] <- 1 - phi[t]  # Pr(alive t -> dead t+1)
    gamma[2,1,t] <- 0           # Pr(dead t -> alive t+1)
    gamma[2,2,t] <- 1           # Pr(dead t -> dead t+1)
  }
  p ~ dunif(0, 1) # prior detection
  omega[1,1] <- 1 - p    # Pr(alive t -> non-detected t)
  omega[1,2] <- p        # Pr(alive t -> detected t)
  omega[2,1] <- 1        # Pr(dead t -> non-detected t)
  omega[2,2] <- 0        # Pr(dead t -> detected t)
  beta[1] ~ dnorm(0, 1.5) # prior intercept
  beta[2] ~ dnorm(0, 1.5) # prior slope
  # likelihood
  for (i in 1:N){
    z[i,first[i]] ~ dcat(delta[1:2])
    for (j in (first[i]+1):T){
      z[i,j] ~ dcat(gamma[z[i,j-1], 1:2, j-1])
      y[i,j] ~ dcat(omega[z[i,j], 1:2])
    }
  }
})
```

We only take the values we need, and standardize the covariate.

```
# water flow in L/s
water_flow <- c(443, 1114, 529, 434, 627, 466, 730) # 1981, 1982, ..., 1987
water_flow <- water_flow[-7]
water_flow_st <- (water_flow - mean(water_flow))/sd(water_flow)
```

Constants in a list.

```
my.constants <- list(N = nrow(y),
                     T = ncol(y),
                     first = first,
                     flow = water_flow_st)
my.constants
```

```
$N
[1] 294

$T
[1] 7

$first
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [38] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3
 [75] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
[112] 3 3 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
[149] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5
[186] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 6
[223] 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 7 7 7 7
[260] 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7

$flow
[1] -0.61028761  1.96250602 -0.28054059 -0.64479602  0.09521765 -0.52209945
```

Initial values.

```
initial.values <- function() list(beta = rnorm(2,0,1),
                                  p = runif(1,0,1),
                                  z = zinits)
```

Parameters to be monitored.

```
parameters.to.save <- c("beta", "p", "phi")
parameters.to.save
```

```
[1] "beta" "p"    "phi"
```

Run nimble.

```
mcmc.phiflowp <- nimbleMCMC(code = hmm.phiflowp,
                            constants = my.constants,
                            data = my.data,
                            inits = initial.values,
                            monitors = parameters.to.save,
                            niter = n.iter,
                            nburnin = n.burnin,
                            nchains = n.chains)
```

```
defining model...
```

```
building model...
```

```
setting data and initial values...
```

```
running calculate on model (any error reports that follow may simply reflect missing val
ues in model variables) ...
checking model sizes and dimensions...
checking model calculations...
model building finished.
compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilat
ion details.
compilation finished.
running chain 1...
```

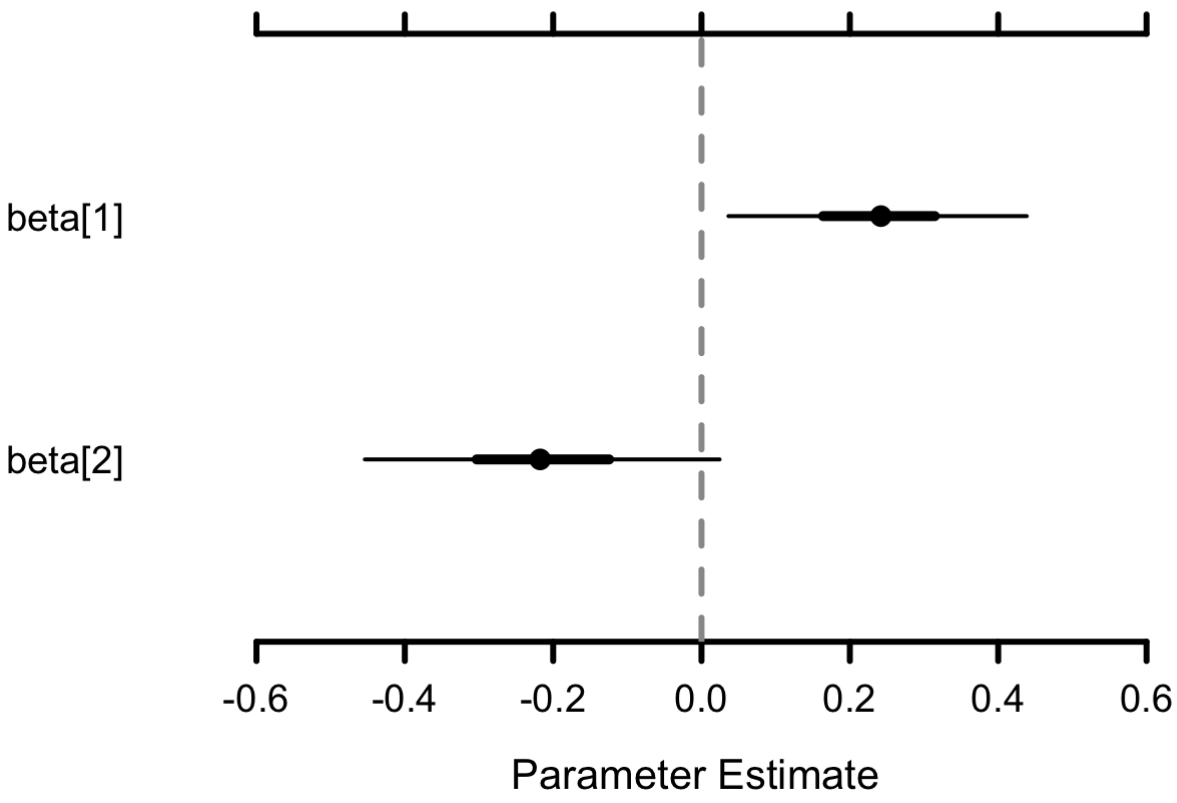
```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

```
running chain 2...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

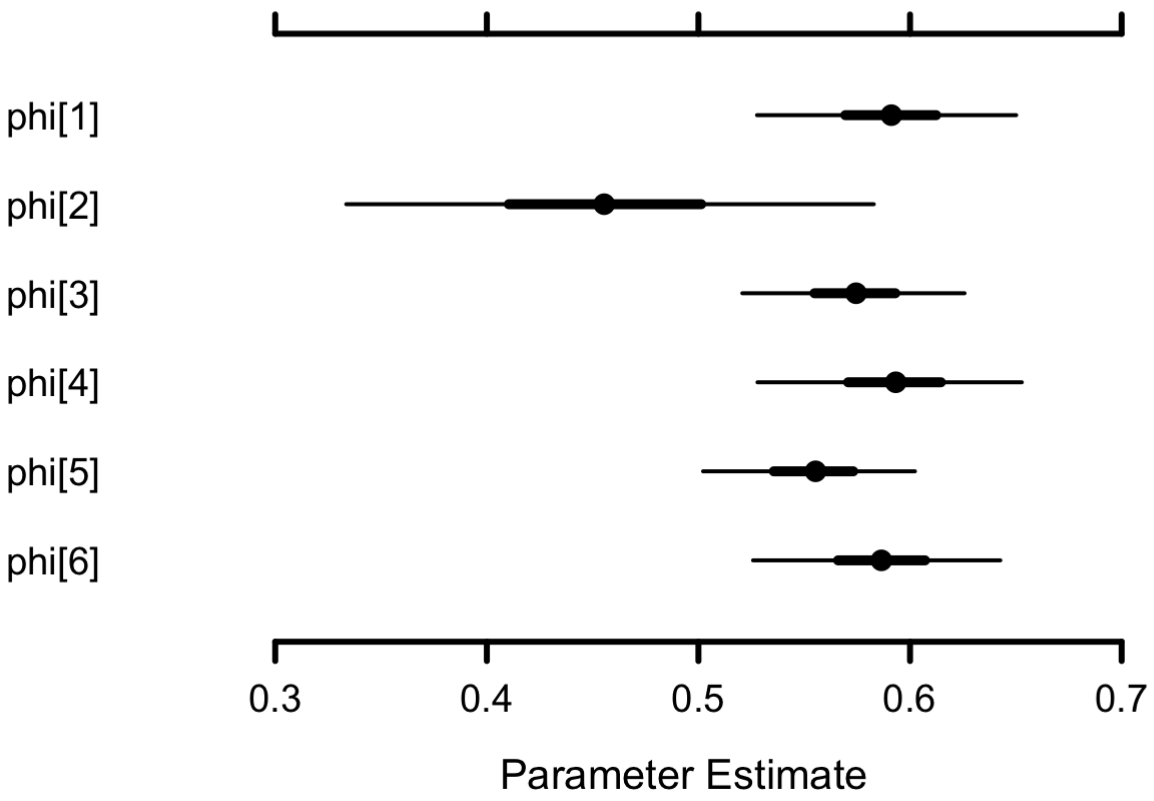
Caterpillar plot of the regression parameters. The posterior distribution of the slope is centered on negative values, suggesting the as water flow increases, survival decreases.

```
MCMCplot(object = mcmc.phiflowp, params = "beta", ISB = TRUE)
```



Caterpillar plot of the survival estimates. Survival between 1982 and 1983 seems to have been affected highly by a huge water flow compared to the other years.

```
MCMCplot(object = mcmc.phiflowp, params = "phi", ISB = TRUE)
```



Yearly random effects

We may wish to allow for extra variation in the survival vs. water flow relationship. To do so, we consider a yearly random effect. The prior on the standard deviation of the random effect is uniform between 0 and 10.

```
hmm.phiflowREpt <- nimbleCode({
  delta[1] <- 1           # Pr(alive t = 1) = 1
  delta[2] <- 0           # Pr(dead t = 1) = 0
  for (t in 1:(T-1)){
    logit(phi[t]) <- beta[1] + beta[2] * flow[t] + eps[t] # eps is random effect
    eps[t] ~ dnorm(0, sd = sdeps)
    gamma[1,1,t] <- phi[t]      # Pr(alive t -> alive t+1)
    gamma[1,2,t] <- 1 - phi[t]  # Pr(alive t -> dead t+1)
    gamma[2,1,t] <- 0           # Pr(dead t -> alive t+1)
    gamma[2,2,t] <- 1           # Pr(dead t -> dead t+1)
    p[t] ~ dunif(0, 1)          # prior detection
    omega[1,1,t] <- 1 - p[t]    # Pr(alive t -> non-detected t)
    omega[1,2,t] <- p[t]        # Pr(alive t -> detected t)
    omega[2,1,t] <- 1           # Pr(dead t -> non-detected t)
    omega[2,2,t] <- 0           # Pr(dead t -> detected t)
  }
  beta[1] ~ dnorm(0, 1.5) # prior intercept
  beta[2] ~ dnorm(0, 1.5) # prior slope
  sdeps ~ dunif(0,10)
  # likelihood
  for (i in 1:N){
    z[i,first[i]] ~ dcat(delta[1:2])
    for (j in (first[i]+1):T){
      z[i,j] ~ dcat(gamma[z[i,j-1], 1:2, j-1])
      y[i,j] ~ dcat(omega[z[i,j], 1:2, j-1])
    }
  }
})
```

Initial values.

```
initial.values <- function() list(beta = rnorm(2,0,1),
                                  p = runif(my.constants$T-1,0,1),
                                  sdeps = runif(1,0,3),
                                  z = zinits)
```

Parameters to be monitored.

```
parameters.to.save <- c("beta", "p", "phi", "sdeps")
```

MCMC details. Note that we've increased the number of iterations and the length of the burn-in period.

```
n.iter <- 10000
n.burnin <- 5000
n.chains <- 2
```

Run nimble.

```
mcmc.phiflowREpt <- nimbleMCMC(code = hmm.phiflowREpt,
                               constants = my.constants,
                               data = my.data,
                               inits = initial.values,
                               monitors = parameters.to.save,
                               niter = n.iter,
                               nburnin = n.burnin,
                               nchains = n.chains)
```

```
defining model...
```

```
building model...
```

```
setting data and initial values...
```

```
running calculate on model (any error reports that follow may simply reflect missing val
ues in model variables) ...
checking model sizes and dimensions... This model is not fully initialized. This is not
an error. To see which variables are not initialized, use model$initializeInfo(). For mo
re information on model initialization, see help(modelInitialization).
checking model calculations...
```

```
NAs were detected in model variables: eps, logProb_eps, phi, gamma, logProb_z.
```

```
model building finished.
compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilat
ion details.
compilation finished.
running chain 1...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

```
running chain 2...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

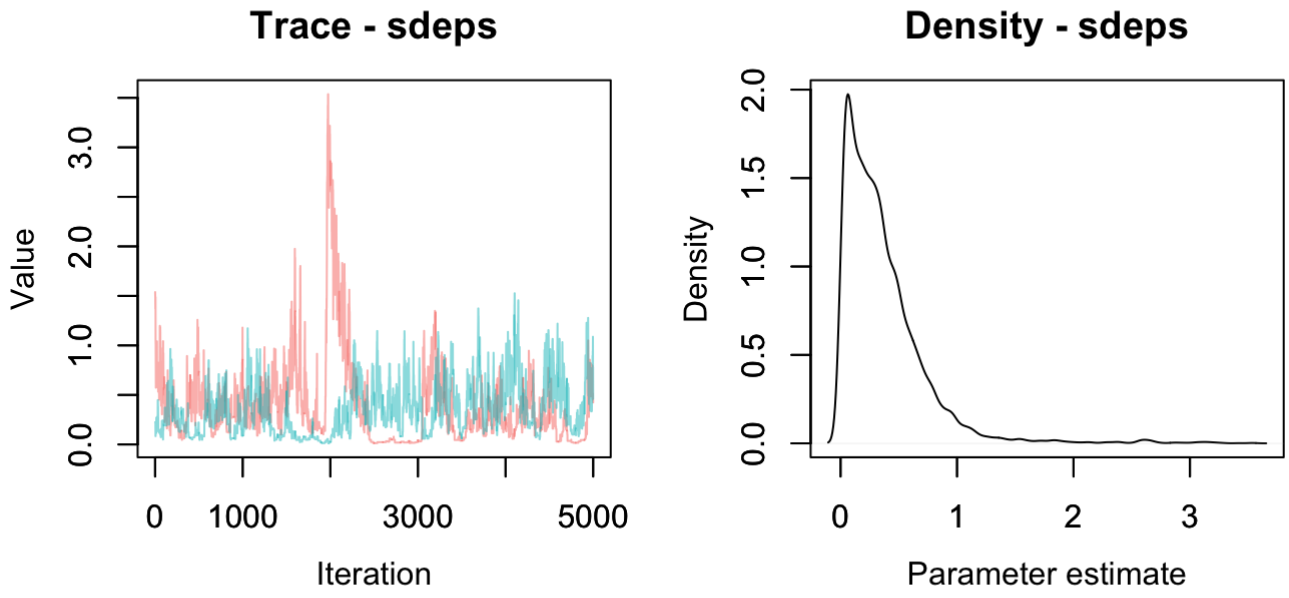
Display outputs. Seems that the water flow effect is not so important anymore.

```
MCMCsummary(object = mcmc.phiflowREpt, round = 2)
```

```
         mean   sd  2.5%   50% 97.5% Rhat n.eff
beta[1]  0.31 0.22 -0.12  0.30  0.79 1.16   210
beta[2] -0.25 0.23 -0.77 -0.24  0.19 1.04   314
p[1]     0.70 0.13  0.44  0.71  0.92 1.00  1438
p[2]     0.88 0.08  0.68  0.89  0.98 1.05   982
p[3]     0.87 0.07  0.71  0.88  0.97 1.01  1078
p[4]     0.88 0.05  0.76  0.88  0.96 1.00  1224
p[5]     0.91 0.05  0.79  0.91  0.98 1.00  1012
p[6]     0.84 0.10  0.62  0.85  0.99 1.04   223
phi[1]   0.64 0.09  0.50  0.63  0.84 1.08   353
phi[2]   0.45 0.07  0.32  0.45  0.59 1.02  1546
phi[3]   0.53 0.06  0.41  0.54  0.63 1.16   465
phi[4]   0.62 0.05  0.52  0.62  0.72 1.02  1315
phi[5]   0.59 0.05  0.50  0.58  0.69 1.06  1122
phi[6]   0.62 0.08  0.50  0.61  0.81 1.05   237
sdeps    0.37 0.37  0.02  0.28  1.21 1.33    89
```

Trace plots for the standard deviation of the random effect.

```
MCMCtrace(object = mcmc.phiflowREpt, params = "sdeps", pdf = FALSE)
```



Add a discrete individual covariate (aka group)

OK now we're gonna illustrate how to have a discrete individual covariate, which we often call group. Here we will consider the sex of individuals. There are two methods to include sex as a covariate.

First, let us define the covariate sex that takes value 0 if the individual is a male, and 1 if it is a female. We put these values in a vector sex.

```
sex <- if_else(dipper$sex == "M", 0, 1)
sex
```

```
  [1] 0 1 0 1 1 0 0 0 0 1 1 0 1 0 0 0 0 0 1 1 1 1 1 1 1 0 1 1 0 1 1 0 0 0 0 0 0
 [38] 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
 [75] 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1
[112] 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 0 0 0 1 1 1 1 0 0 0 0 0 0 1 1 1 1
[149] 1 1 0 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0
[186] 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1
[223] 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
[260] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Now we write the model, and write the survival as a function of the covariate sex on the logit scale $\text{logit}(\phi_i) = \beta_1 + \beta_2 * \text{sex}_i$. We need to make the matrix Γ individual specific. For convenience we also define $\phi_{\text{male}} = \beta_1$ with $\text{sex}_i = 0$ and $\phi_{\text{female}} = \beta_1 + \beta_2$ with $\text{sex}_i = 1$.

```
hmm.phisexp <- nimbleCode({
    p ~ dunif(0, 1) # prior detection
    omega[1,1] <- 1 - p    # Pr(alive t -> non-detected t)
    omega[1,2] <- p        # Pr(alive t -> detected t)
    omega[2,1] <- 1        # Pr(dead t -> non-detected t)
    omega[2,2] <- 0        # Pr(dead t -> detected t)
  for (i in 1:N){ # loop over individuals
    logit(phi[i]) <- beta[1] + beta[2] * sex[i]
    gamma[1,1,i] <- phi[i]      # Pr(alive t -> alive t+1)
    gamma[1,2,i] <- 1 - phi[i]  # Pr(alive t -> dead t+1)
    gamma[2,1,i] <- 0           # Pr(dead t -> alive t+1)
    gamma[2,2,i] <- 1           # Pr(dead t -> dead t+1)
  }
  beta[1] ~ dnorm(mean = 0, sd = 1.5)
  beta[2] ~ dnorm(mean = 0, sd = 1.5)
  phi_male <- 1/(1+exp(-beta[1]))
  phi_female <- 1/(1+exp(-(beta[1]+beta[2])))
  delta[1] <- 1          # Pr(alive t = 1) = 1
  delta[2] <- 0          # Pr(dead t = 1) = 0
  # likelihood
  for (i in 1:N){
    z[i,first[i]] ~ dcat(delta[1:2])
    for (j in (first[i]+1):T){
      z[i,j] ~ dcat(gamma[z[i,j-1], 1:2, i])
      y[i,j] ~ dcat(omega[z[i,j], 1:2])
    }
  }
})
```

Constants in a list.

```
my.constants <- list(N = nrow(y),
                     T = ncol(y),
                     first = first,
                     sex = sex)
```

Data in a list.

```
my.data <- list(y = y + 1)
```

Initial values.

```
initial.values <- function() list(beta = rnorm(2,0,1),
                                  p = runif(1,0,1),
                                  z = zinits)
```

Parameters to monitor.

```
parameters.to.save <- c("beta", "p", "phi_male", "phi_female")
```

MCMC details.

```
n.iter <- 2500
n.burnin <- 1000
n.chains <- 2
```

Run nimble.

```
mcmc.phisexp <- nimbleMCMC(code = hmm.phisexp,
                           constants = my.constants,
                           data = my.data,
                           inits = initial.values,
                           monitors = parameters.to.save,
                           niter = n.iter,
                           nburnin = n.burnin,
                           nchains = n.chains)
```

```
defining model...
```

```
building model...
```

```
setting data and initial values...
```

```
running calculate on model (any error reports that follow may simply reflect missing val
ues in model variables) ...
checking model sizes and dimensions...
checking model calculations...
model building finished.
compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilat
ion details.
compilation finished.
running chain 1...
```

```
|-------------|-------------|-------------|-------------|
|-----------------------------------------------------|
```

```
running chain 2...
```

```
|-------------|-------------|-------------|-------------|
|-----------------------------------------------------|
```

Display results.

```
MCMCsummary(object = mcmc.phisexp, round = 2)
```

```
            mean   sd  2.5%   50% 97.5% Rhat n.eff
beta[1]     0.29 0.15  0.00  0.28  0.58 1.07   190
beta[2]    -0.07 0.20 -0.47 -0.07  0.32 1.05   227
p           0.89 0.03  0.83  0.90  0.94 1.00   230
phi_female  0.55 0.03  0.48  0.55  0.62 1.00   678
phi_male    0.57 0.04  0.50  0.57  0.64 1.07   191
```

Another method to include a group effect is to use nested indexing. Let's use a covariate sex that contains 1s and 2s, indicating the sex of each individual: 1 if male, and 2 if female. E.g. for individual $i = 2$, `beta[sex[i]]` gives `beta[sex[2]]` which will be `beta[1]` or `beta[2]` depending on whether sex[2] is 1 or 2.

```
sex <- if_else(dipper$sex == "M", 1, 2)
sex
```

```
  [1] 1 2 1 2 2 1 1 1 1 2 2 1 2 1 1 1 1 1 2 2 2 2 2 2 2 1 2 2 1 2 2 1 1 1 1 1 1
 [38] 1 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1
 [75] 2 1 1 1 1 2 2 1 1 1 1 1 1 1 1 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2
[112] 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 2 2 1 1 1 2 2 2 2 1 1 1 1 1 1 2 2 2 2
[149] 2 2 1 2 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 1
[186] 1 1 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 2
[223] 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1
[260] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

Write the model.

```
hmm.phisexpt.ni <- nimbleCode({
    p ~ dunif(0, 1) # prior detection
    omega[1,1] <- 1 - p    # Pr(alive t -> non-detected t)
    omega[1,2] <- p        # Pr(alive t -> detected t)
    omega[2,1] <- 1        # Pr(dead t -> non-detected t)
    omega[2,2] <- 0        # Pr(dead t -> detected t)
  for (i in 1:N){
    phi[i] <- beta[sex[i]]
    gamma[1,1,i] <- phi[i]      # Pr(alive t -> alive t+1)
    gamma[1,2,i] <- 1 - phi[i]  # Pr(alive t -> dead t+1)
    gamma[2,1,i] <- 0        # Pr(dead t -> alive t+1)
    gamma[2,2,i] <- 1        # Pr(dead t -> dead t+1)
  }
  beta[1] ~ dunif(0,1)
  beta[2] ~ dunif(0,1)
  delta[1] <- 1          # Pr(alive t = 1) = 1
  delta[2] <- 0          # Pr(dead t = 1) = 0
  # likelihood
  for (i in 1:N){
    z[i,first[i]] ~ dcat(delta[1:2])
    for (j in (first[i]+1):T){
      z[i,j] ~ dcat(gamma[z[i,j-1], 1:2, i])
      y[i,j] ~ dcat(omega[z[i,j], 1:2])
    }
  }
})
```

My constants.

```
my.constants <- list(N = nrow(y),
                     T = ncol(y),
                     first = first,
                     sex = sex)  # beta[1] male survival
                                 # beta[2] female survival
```

Parameters to monitor.

```
parameters.to.save <- c("beta", "p")
```

Initial values.

```
initial.values <- function() list(beta = runif(2,0,1),
                                  p = runif(1,0,1),
                                  z = zinits)
```

Run nimble.

```
mcmc.phisexp.ni <- nimbleMCMC(code = hmm.phisexpt.ni,
                              constants = my.constants,
                              data = my.data,
                              inits = initial.values,
                              monitors = parameters.to.save,
                              niter = n.iter,
                              nburnin = n.burnin,
                              nchains = n.chains)
```

```
defining model...
```

```
building model...
```

```
setting data and initial values...
```

```
running calculate on model (any error reports that follow may simply reflect missing val
ues in model variables) ...
checking model sizes and dimensions...
checking model calculations...
model building finished.
compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilat
ion details.
compilation finished.
running chain 1...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

```
running chain 2...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

Dislpay results. Compare with the other method above, the estimates are very similar.

```
MCMCsummary(object = mcmc.phisexp.ni, round = 2)
```

```
        mean   sd 2.5%  50% 97.5% Rhat n.eff
beta[1] 0.57 0.04 0.50 0.57  0.64 1.01   615
beta[2] 0.55 0.03 0.48 0.55  0.62 1.01   575
p       0.90 0.03 0.83 0.90  0.95 1.01   258
```

Add a continuous individual covariate

Besides discrete individual covariates, you might want to have continuous individual covariates, e.g. wing length in the dipper case study. Note that we're considering an individual trait that takes the same value whatever the occasion. If we were to have time-varying individual covariate in the model, we would have to do something about missing values of the covariate when an individual is not recaptured. The easiest way to cope with time-varying individual covariate is to discretize and treat levels of the covariates as states. More in the next live demo. Back to wing length. We first standardize the covariate.

```
wing.length.st <- as.vector(scale(dipper$wing_length))
head(wing.length.st)
```

```
[1]  0.7581335 -0.8671152  0.5259551 -1.5636504 -1.3314720  1.2224903
```

Now we write the model. Basically we replace sex by wing length in the first method we used in the previous section. Easy.

```
hmm.phiwlp <- nimbleCode({
    p ~ dunif(0, 1) # prior detection
    omega[1,1] <- 1 - p    # Pr(alive t -> non-detected t)
    omega[1,2] <- p        # Pr(alive t -> detected t)
    omega[2,1] <- 1        # Pr(dead t -> non-detected t)
    omega[2,2] <- 0        # Pr(dead t -> detected t)
  for (i in 1:N){
    logit(phi[i]) <- beta[1] + beta[2] * winglength[i]
    gamma[1,1,i] <- phi[i]      # Pr(alive t -> alive t+1)
    gamma[1,2,i] <- 1 - phi[i]  # Pr(alive t -> dead t+1)
    gamma[2,1,i] <- 0           # Pr(dead t -> alive t+1)
    gamma[2,2,i] <- 1           # Pr(dead t -> dead t+1)
  }
  beta[1] ~ dnorm(mean = 0, sd = 1.5)
  beta[2] ~ dnorm(mean = 0, sd = 1.5)
  delta[1] <- 1          # Pr(alive t = 1) = 1
  delta[2] <- 0          # Pr(dead t = 1) = 0
  # likelihood
  for (i in 1:N){
    z[i,first[i]] ~ dcat(delta[1:2])
    for (j in (first[i]+1):T){
      z[i,j] ~ dcat(gamma[z[i,j-1], 1:2, i])
      y[i,j] ~ dcat(omega[z[i,j], 1:2])
    }
  }
})
```

Constants in a list.

```
my.constants <- list(N = nrow(y),
                     T = ncol(y),
                     first = first,
                     winglength = wing.length.st)
```

Initial values.

```
initial.values <- function() list(beta = rnorm(2,0,1),
                                  p = runif(1,0,1),
                                  z = zinits)
```

Run nimble.

```
mcmc.phiwlp <- nimbleMCMC(code = hmm.phiwlp,
                          constants = my.constants,
                          data = my.data,
                          inits = initial.values,
                          monitors = parameters.to.save,
                          niter = n.iter,
                          nburnin = n.burnin,
                          nchains = n.chains)
```

```
defining model...
```

```
building model...
```

```
setting data and initial values...
```

```
running calculate on model (any error reports that follow may simply reflect missing val
ues in model variables) ...
checking model sizes and dimensions...
checking model calculations...
model building finished.
compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilat
ion details.
compilation finished.
running chain 1...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

```
running chain 2...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

Numerical summaries. Wing length does not seem to explain much individual-to-individual variation in survival.

```
MCMCsummary(mcmc.phiwlp, round = 2)
```

```
         mean   sd  2.5%   50% 97.5% Rhat n.eff
beta[1]  0.24 0.10  0.02  0.25  0.45 1.01   528
beta[2] -0.02 0.09 -0.21 -0.02  0.15 1.00   674
p        0.90 0.03  0.84  0.90  0.95 1.00   259
```

Let's plot the relationship. First, we gather the values generated from the posterior distribution of the regression parameters in the two chains.

```
beta1 <- c(mcmc.phiwlp$chain1[,'beta[1]'], mcmc.phiwlp$chain2[,'beta[1]'])
beta2 <- c(mcmc.phiwlp$chain1[,'beta[2]'], mcmc.phiwlp$chain2[,'beta[2]'])
```

Then we define a grid of values for wing length, and predict survival for each MCMC iteration.

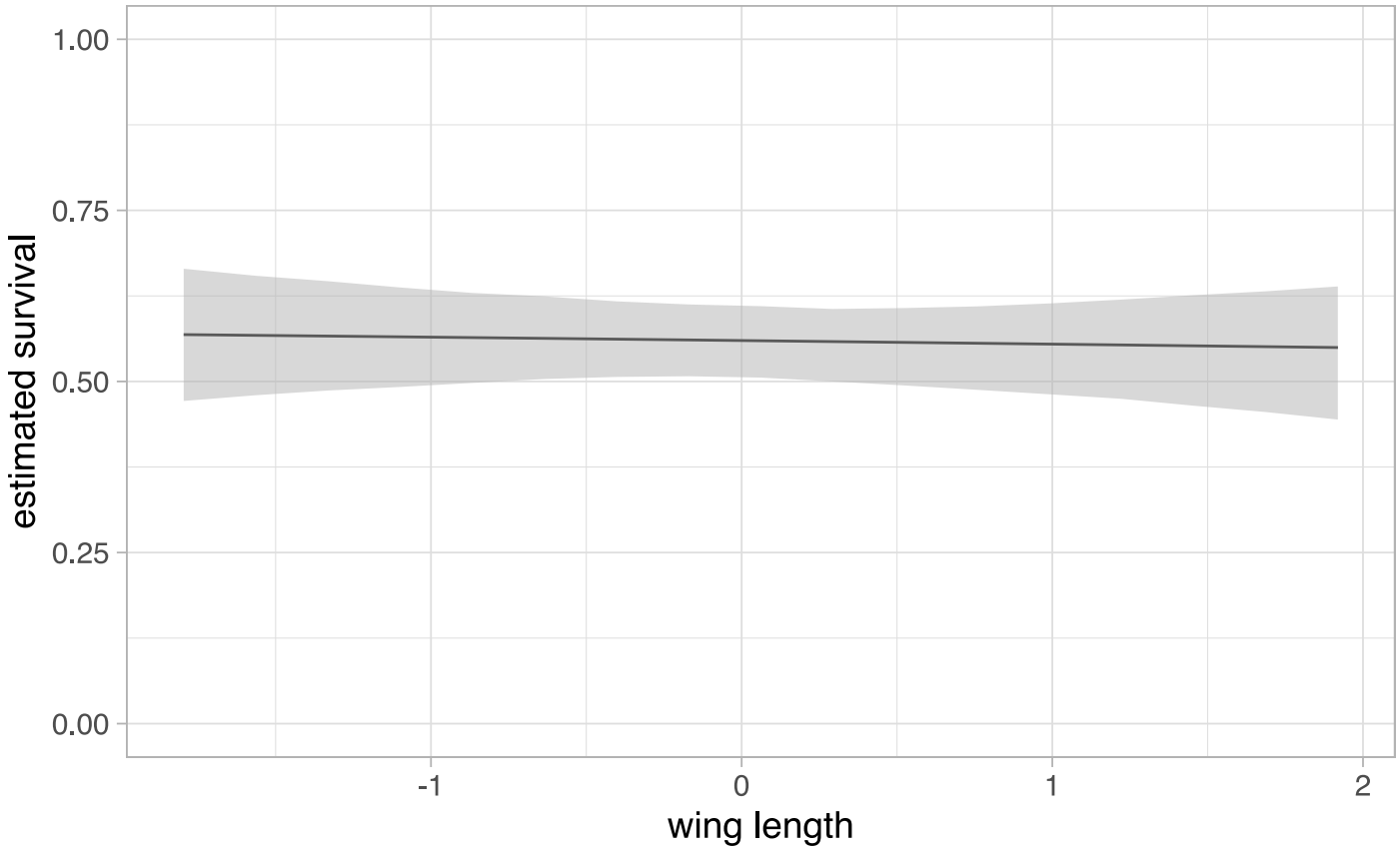
```
predicted_survival <- matrix(NA, nrow = length(beta1), ncol = length(my.constants$wingle
ngth))
for (i in 1:length(beta1)){
  for (j in 1:length(my.constants$winglength)){
    predicted_survival[i,j] <- plogis(beta1[i] + beta2[i] * my.constants$winglength[j])
  }
}
```

Now we calculate posterior mean and the credible interval. Note the ordering.

```
mean_survival <- apply(predicted_survival, 2, mean)
lci <- apply(predicted_survival, 2, quantile, prob = 2.5/100)
uci <- apply(predicted_survival, 2, quantile, prob = 97.5/100)
ord <- order(my.constants$winglength)
df <- data.frame(wing_length = my.constants$winglength[ord],
                 survival = mean_survival[ord],
                 lci = lci[ord],
                 uci = uci[ord])
```

Now time to visualize.

```
df %>%
  ggplot() +
  aes(x = wing_length, y = survival) +
  geom_line() +
  geom_ribbon(aes(ymin = lci, ymax = uci), fill = "grey70", alpha = 0.5) +
  ylim(0,1) +
  labs(x = "wing length", y = "estimated survival")
```



Add an individual random effect on top of the individual covariate

We add an individual random effect.

```
hmm.phiwlrep <- nimbleCode({
    p ~ dunif(0, 1) # prior detection
    omega[1,1] <- 1 - p    # Pr(alive t -> non-detected t)
    omega[1,2] <- p        # Pr(alive t -> detected t)
    omega[2,1] <- 1        # Pr(dead t -> non-detected t)
    omega[2,2] <- 0        # Pr(dead t -> detected t)
  for (i in 1:N){
    logit(phi[i]) <- beta[1] + beta[2] * winglength[i] + eps[i]
    eps[i] ~ dnorm(mean = 0, sd = sdeps)
    gamma[1,1,i] <- phi[i]      # Pr(alive t -> alive t+1)
    gamma[1,2,i] <- 1 - phi[i]  # Pr(alive t -> dead t+1)
    gamma[2,1,i] <- 0           # Pr(dead t -> alive t+1)
    gamma[2,2,i] <- 1           # Pr(dead t -> dead t+1)
  }
  beta[1] ~ dnorm(mean = 0, sd = 1.5)
  beta[2] ~ dnorm(mean = 0, sd = 1.5)
  sdeps ~ dunif(0, 10)
  delta[1] <- 1          # Pr(alive t = 1) = 1
  delta[2] <- 0          # Pr(dead t = 1) = 0
  # likelihood
  for (i in 1:N){
    z[i,first[i]] ~ dcat(delta[1:2])
    for (j in (first[i]+1):T){
      z[i,j] ~ dcat(gamma[z[i,j-1], 1:2, i])
      y[i,j] ~ dcat(omega[z[i,j], 1:2])
    }
  }
})
```

Initial values.

```
initial.values <- function() list(beta = rnorm(2,0,1.5),
                                  sdeps = runif(1,0,3),
                                  p = runif(1,0,1),
                                  z = zinits)
```

Parameters to be monitored.

```
parameters.to.save <- c("beta", "sdeps", "p")
```

MCMC details. Note that we increase the number of iterations and the length of the burn-in period.

```
n.iter <- 10000
n.burnin <- 5000
n.chains <- 2
```

Run nimble.

```
mcmc.phiwlrep <- nimbleMCMC(code = hmm.phiwlrep,
                            constants = my.constants,
                            data = my.data,
                            inits = initial.values,
                            monitors = parameters.to.save,
                            niter = n.iter,
                            nburnin = n.burnin,
                            nchains = n.chains)
```

```
defining model...
```

```
building model...
```

```
setting data and initial values...
```

```
running calculate on model (any error reports that follow may simply reflect missing val
ues in model variables) ...
checking model sizes and dimensions... This model is not fully initialized. This is not
an error. To see which variables are not initialized, use model$initializeInfo(). For mo
re information on model initialization, see help(modelInitialization).
checking model calculations...
```

```
NAs were detected in model variables: eps, logProb_eps, phi, gamma, logProb_z.
```

```
model building finished.
compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilat
ion details.
compilation finished.
running chain 1...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

```
running chain 2...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

Numerical summaries.

```
MCMCsummary(mcmc.phiwlrep, round = 2)
```

```
        mean   sd  2.5%  50% 97.5% Rhat n.eff
beta[1] 0.18 0.12 -0.07 0.19  0.41 1.07   700
beta[2] 0.00 0.11 -0.21 0.00  0.21 1.02  1626
p       0.90 0.03  0.83 0.90  0.95 1.00   863
sdeps   0.55 0.22  0.12 0.53  1.03 2.07    16
```

Let's try something else. We reparameterize by non-centering.

```
hmm.phiwlrep <- nimbleCode({
    p ~ dunif(0, 1) # prior detection
    omega[1,1] <- 1 - p    # Pr(alive t -> non-detected t)
    omega[1,2] <- p        # Pr(alive t -> detected t)
    omega[2,1] <- 1        # Pr(dead t -> non-detected t)
    omega[2,2] <- 0        # Pr(dead t -> detected t)
  for (i in 1:N){
    logit(phi[i]) <- beta[1] + beta[2] * winglength[i] + sdeps * eps[i]
    eps[i] ~ dnorm(mean = 0, sd = 1)
    gamma[1,1,i] <- phi[i]      # Pr(alive t -> alive t+1)
    gamma[1,2,i] <- 1 - phi[i]  # Pr(alive t -> dead t+1)
    gamma[2,1,i] <- 0           # Pr(dead t -> alive t+1)
    gamma[2,2,i] <- 1           # Pr(dead t -> dead t+1)
  }
  beta[1] ~ dnorm(mean = 0, sd = 1.5)
  beta[2] ~ dnorm(mean = 0, sd = 1.5)
  sdeps ~ dunif(0, 10)
  delta[1] <- 1          # Pr(alive t = 1) = 1
  delta[2] <- 0          # Pr(dead t = 1) = 0
  # likelihood
  for (i in 1:N){
    z[i,first[i]] ~ dcat(delta[1:2])
    for (j in (first[i]+1):T){
      z[i,j] ~ dcat(gamma[z[i,j-1], 1:2, i])
      y[i,j] ~ dcat(omega[z[i,j], 1:2])
    }
  }
})
```

Initial values.

```
initial.values <- function() list(beta = rnorm(2,0,1.5),
                                  sdeps = runif(1,0,3),
                                  p = runif(1,0,1),
                                  z = zinits)
```

Parameters to be monitored.

```
parameters.to.save <- c("beta", "sdeps", "p")
```

MCMC details. Note that we increase the number of iterations and the length of the burn-in period.

```
n.iter <- 10000
n.burnin <- 5000
n.chains <- 2
```

Run nimble.

```
mcmc.phiwlrep <- nimbleMCMC(code = hmm.phiwlrep,
                            constants = my.constants,
                            data = my.data,
                            inits = initial.values,
                            monitors = parameters.to.save,
                            niter = n.iter,
                            nburnin = n.burnin,
                            nchains = n.chains)
```

```
defining model...
```

```
building model...
```

```
setting data and initial values...
```

```
running calculate on model (any error reports that follow may simply reflect missing val
ues in model variables) ...
checking model sizes and dimensions... This model is not fully initialized. This is not
an error. To see which variables are not initialized, use model$initializeInfo(). For mo
re information on model initialization, see help(modelInitialization).
checking model calculations...
```

```
NAs were detected in model variables: eps, logProb_eps, phi, gamma, logProb_z.
```

```
model building finished.
compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilat
ion details.
compilation finished.
running chain 1...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

```
running chain 2...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

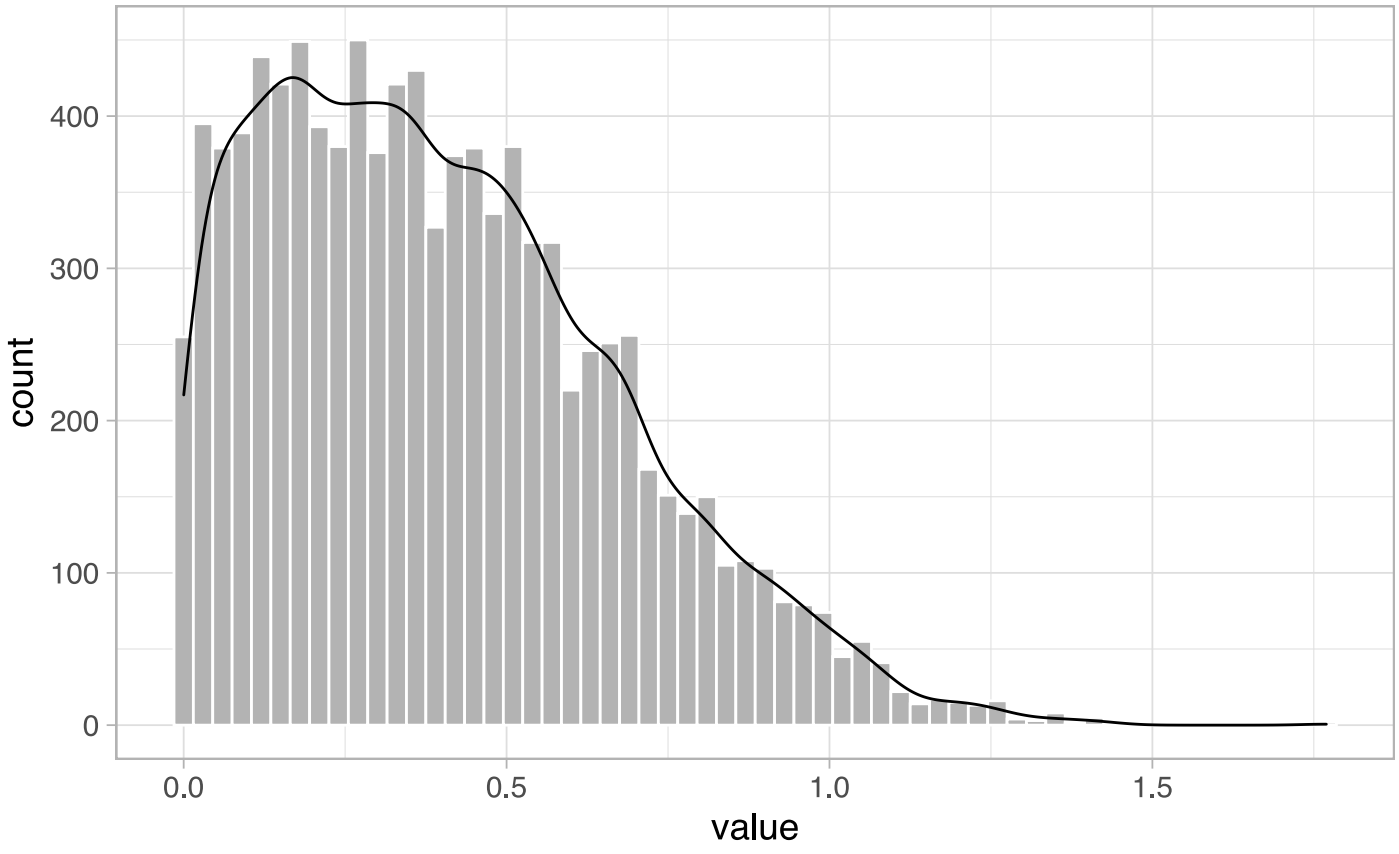
Numerical summaries. Much better.

```
MCMCsummary(mcmc.phiwlrep, round = 2)
```

```
         mean   sd  2.5%   50% 97.5% Rhat n.eff
beta[1]  0.21 0.12 -0.04  0.21  0.44 1.00   575
beta[2] -0.01 0.10 -0.21 -0.02  0.19 1.01  1729
p        0.90 0.03  0.83  0.90  0.95 1.00   811
sdeps    0.40 0.27  0.01  0.36  1.01 1.01   165
```

Let's plot the posterior distribution of the standard deviation of the individual random effect.

```
sdeps <- c(mcmc.phiwlrep$chain1[,"sdeps"], mcmc.phiwlrep$chain2[,"sdeps"])
sdeps %>%
  as_tibble() %>%
  ggplot() +
  aes(x = value) +
  geom_histogram(color = "white", binwidth = .03, fill = "gray70") +
  geom_density(aes(y = .03 * ..count..))
```



Class 6 live demo: Transition estimation

The team

last updated: 2021-05-14

Introduction

In this demo, we illustrate how to fit multistate models to capture-recapture data. We first consider states as sites and estimate movements. Next we treat states as breeding states for studying life-history trade-offs.

```
library(tidyverse)
library(nimble)
library(MCMCvis)
```

Multisite model with 2 sites

We are going to analyze real capture-recapture data on > 20,000 Canada geese (Branta canadensis) marked and resighted in the east coast of the US in the mid-Atlantic (New York, Pennsylvania, New Jersey), Chesapeake (Delaware, Maryland, Virginia), and Carolinas (North and South Carolina). The data were kindly provided by Jay Hestbeck. We analyse a subset of 500 geese from the original dataset, we'll get back to the whole dataset in the last live demo.

Read in the data.

```
geese <- read_csv("geese.csv", col_names = TRUE)
y <- as.matrix(geese)
head(y)
```

```
     year_1984 year_1985 year_1986 year_1987 year_1988 year_1989
[1,]         0         2         2         0         0         0
[2,]         0         0         0         0         0         2
[3,]         0         0         0         1         0         0
[4,]         0         0         2         0         0         0
[5,]         0         3         0         0         3         2
[6,]         0         0         0         2         0         0
```

For simplicity, we start by considering only 2 sites, dropping Carolinas.

```
y2 <- y
y2[y2==3] <- 0 # act as if there was no detection in site 3 Carolinas
mask <- apply(y2, 1, sum)
y2 <- y2[mask!=0,] # remove rows w/ 0s only
head(y2)
```

```
     year_1984 year_1985 year_1986 year_1987 year_1988 year_1989
[1,]         0         2         2         0         0         0
[2,]         0         0         0         0         0         2
[3,]         0         0         0         1         0         0
[4,]         0         0         2         0         0         0
[5,]         0         0         0         0         0         2
[6,]         0         0         0         2         0         0
```

Get occasion of first capture.

```
get.first <- function(x) min(which(x != 0))
first <- apply(y2, 1, get.first)
first
```

```
  [1] 2 6 4 3 6 4 1 5 4 2 2 3 1 5 4 6 3 4 5 2 5 5 4 3 4 4 4 3 4 1 2 4 3 2 2 4 5
 [38] 2 3 2 2 2 1 1 5 5 2 3 3 2 3 3 4 4 4 6 2 3 1 2 2 3 2 2 4 4 3 2 4 3 3 3 4 2
 [75] 2 3 2 4 5 2 3 2 5 4 2 3 3 2 3 1 2 3 4 2 6 3 2 1 4 5 3 1 3 3 5 1 1 6 2 1 4
[112] 2 2 3 2 4 1 5 4 4 5 5 3 1 4 3 5 3 2 5 3 3 3 2 2 4 5 2 3 3 1 5 5 2 3 1 6 3
[149] 3 5 2 3 2 2 3 3 3 1 4 3 2 1 5 3 2 1 3 3 1 4 3 2 4 4 4 2 1 4 1 1 3 3 3 3 3
[186] 3 3 3 1 3 2 3 3 3 4 3 2 1 3 4 4 2 2 2 2 3 3 2 2 5 4 3 1 2 2 3 6 3 3 6 2 1
[223] 4 2 3 6 2 3 5 2 5 1 3 5 6 1 2 3 1 3 5 3 1 3 5 4 2 2 5 3 3 4 1 5 4 2 5 3 6
[260] 4 1 4 2 2 2 4 3 1 3 5 3 2 1 3 1 1 5 3 3 3 3 1 6 4 1 1 5 5 3 2 2 6 4 5 2 4
[297] 2 4 1 3 4 2 4 3 2 4 1 4 1 3 2 1 2 2 2 4 4 5 4 3 3 3 4 4 4 3 5 3 3 2 3 4 3
[334] 4 5 3 6 2 2 2 2 3 2 2 5 3 2 3 3 4 4 2 5 2 3 3 4 5 5 5 4 2 4 2 4 1 3 2 3 6
[371] 2 2 4 3 1 2 2 1 1 3 3 3 3 6 3 1 2 5 1 4 1 2 2 3 4 1 2 3 4 1 2 4 2 5 2 5 2
[408] 5 4 2 6 2 1 1 2 3 4
```

Write the model.

```
multisite <- nimbleCode({

  # ----------------------------------------------
  # Parameters:
  # phiA: survival probability site A
  # phiB: survival probability site B
  # psiAB: movement probability from site A to site B
  # psiBA: movement probability from site B to site A
  # pA: recapture probability site A
  # pB: recapture probability site B
  # piA: prop of being in site A at initial capture
  # ----------------------------------------------
  # States (z):
  # 1 alive at A
  # 2 alive at B
  # 3 dead
  # Observations (y):
  # 1 not seen
  # 2 seen at site A
  # 3 seen at site B
  # ----------------------------------------------

  # priors
  phiA ~ dunif(0, 1)
  phiB ~ dunif(0, 1)
  psiAB ~ dunif(0, 1)
  psiBA ~ dunif(0, 1)
  pA ~ dunif(0, 1)
  pB ~ dunif(0, 1)
  piA ~ dunif(0, 1)

  # probabilities of state z(t+1) given z(t)
  gamma[1,1] <- phiA * (1 - psiAB)
  gamma[1,2] <- phiA * psiAB
  gamma[1,3] <- 1 - phiA
  gamma[2,1] <- phiB * psiBA
  gamma[2,2] <- phiB * (1 - psiBA)
  gamma[2,3] <- 1 - phiB
  gamma[3,1] <- 0
  gamma[3,2] <- 0
  gamma[3,3] <- 1

  delta[1] <- piA          # Pr(alive in A t = 1)
  delta[2] <- 1 - piA      # Pr(alive in B t = 1)
  delta[3] <- 0            # Pr(dead t = 1) = 0

  # probabilities of y(t) given z(t)
  omega[1,1] <- 1 - pA     # Pr(alive A t -> non-detected t)
  omega[1,2] <- pA         # Pr(alive A t -> detected A t)
  omega[1,3] <- 0          # Pr(alive A t -> detected B t)
  omega[2,1] <- 1 - pB     # Pr(alive B t -> non-detected t)
  omega[2,2] <- 0          # Pr(alive B t -> detected A t)
  omega[2,3] <- pB         # Pr(alive B t -> detected B t)
```

```
  omega[3,1] <- 1           # Pr(dead t -> non-detected t)
  omega[3,2] <- 0           # Pr(dead t -> detected A t)
  omega[3,3] <- 0           # Pr(dead t -> detected B t)

  # likelihood
  for (i in 1:N){
    # latent state at first capture
    z[i,first[i]] ~ dcat(delta[1:3])
    for (t in (first[i]+1):K){
      # draw z(t) given z(t-1)
      z[i,t] ~ dcat(gamma[z[i,t-1],1:3])
      # draw y(t) given z(t)
      y[i,t] ~ dcat(omega[z[i,t],1:3])
    }
  }
})
```

Data in a list. Remember to add 1.

```
my.data <- list(y = y2 + 1)
```

Constants in a list.

```
my.constants <- list(first = first,
                     K = ncol(y2),
                     N = nrow(y2))
```

Initial values.

```
zinits <- y2 # say states are observations, detections in A or B are taken as alive in s
ame sites
zinits[zinits==0] <- sample(c(1,2), sum(zinits==0), replace = TRUE) # non-detections bec
ome alive in site A or B
initial.values <- function(){list(phiA = runif(1, 0, 1),
                                  phiB = runif(1, 0, 1),
                                  psiAB = runif(1, 0, 1),
                                  psiBA = runif(1, 0, 1),
                                  pA = runif(1, 0, 1),
                                  pB = runif(1, 0, 1),
                                  piA = runif(1, 0, 1),
                                  z = zinits)}
```

Parameters to monitor.

```
parameters.to.save <- c("phiA", "phiB","psiAB", "psiBA", "pA", "pB", "piA")
parameters.to.save
```

```
[1] "phiA"  "phiB"  "psiAB" "psiBA" "pA"    "pB"    "piA"
```

MCMC settings.

```
n.iter <- 10000
n.burnin <- 5000
n.chains <- 2
```

Run nimble.

```
mcmc.multisite <- nimbleMCMC(code = multisite,
                             constants = my.constants,
                             data = my.data,
                             inits = initial.values,
                             monitors = parameters.to.save,
                             niter = n.iter,
                             nburnin = n.burnin,
                             nchains = n.chains)
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

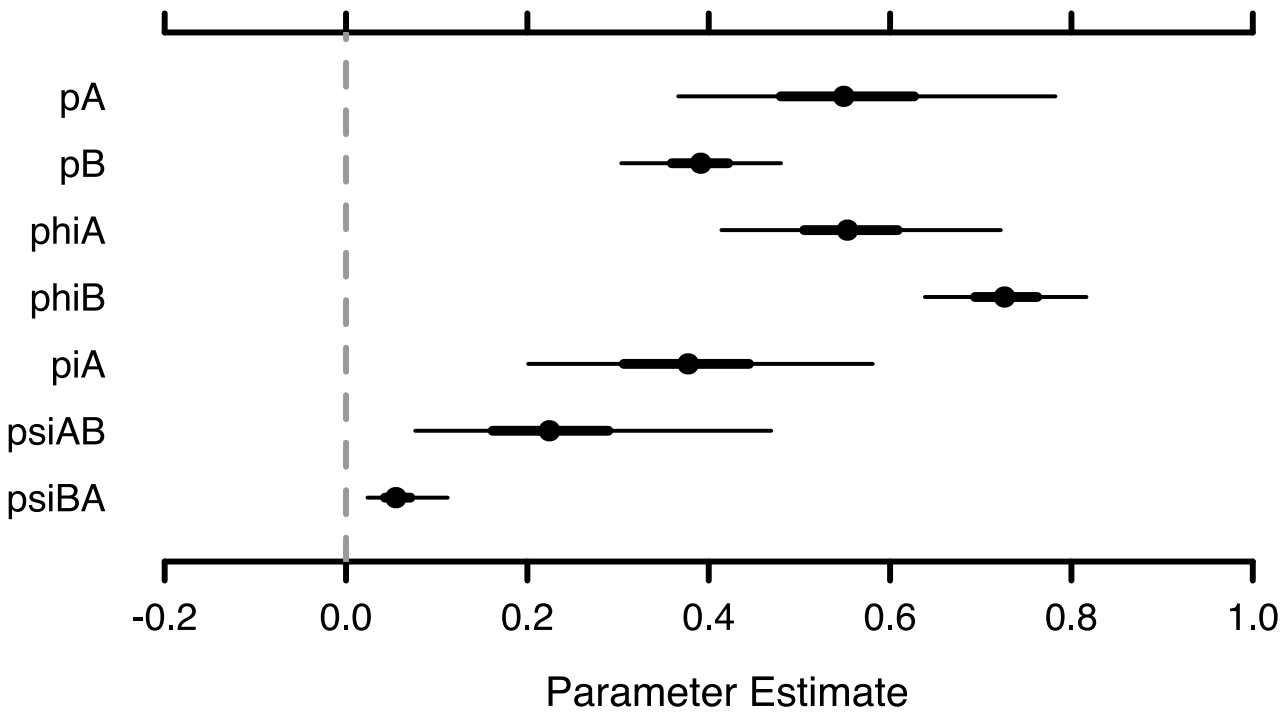
Let's inspect the results. First the numerical summaries.

```
MCMCsummary(mcmc.multisite, round = 2)
```

```
      mean   sd 2.5%  50% 97.5% Rhat n.eff
pA    0.56 0.11 0.37 0.55  0.78 1.01   226
pB    0.39 0.04 0.30 0.39  0.48 1.00   207
phiA  0.56 0.08 0.41 0.55  0.72 1.01   208
phiB  0.73 0.05 0.64 0.73  0.82 1.00   180
piA   0.38 0.10 0.20 0.38  0.58 1.00    85
psiAB 0.23 0.10 0.08 0.22  0.47 1.01   191
psiBA 0.06 0.02 0.02 0.06  0.11 1.02   410
```

Second, a caterpillar plot of the parameter estimates.

```
MCMCplot(mcmc.multisite)
```



Actually, the initial state of a goose is known exactly. It is alive at site of initial capture. Therefore, we don't need to try and estimate the probability of initial states. Let's rewrite the model.

```
multisite <- nimbleCode({

  # ----------------------------------------------
  # Parameters:
  # phiA: survival probability site A
  # phiB: survival probability site B
  # psiAB: movement probability from site A to site B
  # psiBA: movement probability from site B to site A
  # pA: recapture probability site A
  # pB: recapture probability site B
  # ----------------------------------------------
  # States (z):
  # 1 alive at A
  # 2 alive at B
  # 3 dead
  # Observations (y):
  # 1 not seen
  # 2 seen at A
  # 3 seen at B
  # ----------------------------------------------

  # priors
  phiA ~ dunif(0, 1)
  phiB ~ dunif(0, 1)
  psiAB ~ dunif(0, 1)
  psiBA ~ dunif(0, 1)
  pA ~ dunif(0, 1)
  pB ~ dunif(0, 1)

  # probabilities of state z(t+1) given z(t)
  gamma[1,1] <- phiA * (1 - psiAB)
  gamma[1,2] <- phiA * psiAB
  gamma[1,3] <- 1 - phiA
  gamma[2,1] <- phiB * psiBA
  gamma[2,2] <- phiB * (1 - psiBA)
  gamma[2,3] <- 1 - phiB
  gamma[3,1] <- 0
  gamma[3,2] <- 0
  gamma[3,3] <- 1

  # probabilities of y(t) given z(t)
  omega[1,1] <- 1 - pA     # Pr(alive A t -> non-detected t)
  omega[1,2] <- pA         # Pr(alive A t -> detected A t)
  omega[1,3] <- 0          # Pr(alive A t -> detected B t)
  omega[2,1] <- 1 - pB     # Pr(alive B t -> non-detected t)
  omega[2,2] <- 0          # Pr(alive B t -> detected A t)
  omega[2,3] <- pB         # Pr(alive B t -> detected B t)
  omega[3,1] <- 1          # Pr(dead t -> non-detected t)
  omega[3,2] <- 0          # Pr(dead t -> detected A t)
  omega[3,3] <- 0          # Pr(dead t -> detected B t)

  # likelihood
  for (i in 1:N){
```

```
    # latent state at first capture
    z[i,first[i]] <- y[i,first[i]] - 1 # if seen at site A (y = 2), state is alive in A
 (y - 1 = z = 1) with prob = 1
    for (t in (first[i]+1):K){         # if seen at site B (y = 3), state is alive in A
 (y - 1 = z = 2) with prob = 1
      # draw (t) given z(t-1)
      z[i,t] ~ dcat(gamma[z[i,t-1],1:3])
      # draw y(t) given z(t)
      y[i,t] ~ dcat(omega[z[i,t],1:3])
    }
  }
})
```

Initial values without p_A.

```
zinits <- y2
zinits[zinits==0] <- sample(c(1,2), sum(zinits==0), replace = TRUE)
initial.values <- function(){list(phiA = runif(1, 0, 1),
                                  phiB = runif(1, 0, 1),
                                  psiAB = runif(1, 0, 1),
                                  psiBA = runif(1, 0, 1),
                                  pA = runif(1, 0, 1),
                                  pB = runif(1, 0, 1),
                                  z = zinits)}
```

Parameters to monitor.

```
parameters.to.save <- c("phiA", "phiB","psiAB", "psiBA", "pA", "pB")
parameters.to.save
```

```
[1] "phiA"  "phiB"  "psiAB" "psiBA" "pA"    "pB"
```

Run nimble.

```
mcmc.multisite <- nimbleMCMC(code = multisite,
                             constants = my.constants,
                             data = my.data,
                             inits = initial.values,
                             monitors = parameters.to.save,
                             niter = n.iter,
                             nburnin = n.burnin,
                             nchains = n.chains)
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

Have a look to the results. Note that convergence is better.

```
MCMCsummary(mcmc.multisite, round = 2)
```

```
      mean   sd 2.5%  50% 97.5% Rhat n.eff
pA    0.52 0.09 0.35 0.52  0.70 1.01   280
pB    0.40 0.04 0.32 0.40  0.49 1.01   323
phiA  0.61 0.05 0.51 0.61  0.72 1.02   400
phiB  0.69 0.04 0.63 0.69  0.77 1.01   341
psiAB 0.27 0.06 0.16 0.26  0.39 1.00   598
psiBA 0.07 0.02 0.04 0.07  0.12 1.01   488
```

Multisite model with 3 sites

Now we proceed with the analysis of the 3 sites. Two methods exist to force the movement probabilities from a site to sum to 1 and to be between 0 and 1 at the same time.

Multinomial logit link

Get the date of first capture.

```
get.first <- function(x) min(which(x != 0))
first <- apply(y, 1, get.first)
```

Write the model.

```
multisite <- nimbleCode({

  # ------------------------------------------------
  # Parameters:
  # phiA: survival probability site A
  # phiB: survival probability site B
  # phiC: survival probability site B
  # psiAA: movement probability from site A to site A (reference)
  # psiAB = psiA[1]: movement probability from site A to site B
  # psiAC = psiA[2]: movement probability from site A to site C
  # psiBA = psiB[1]: movement probability from site B to site A
  # psiBB: movement probability from site B to site B (reference)
  # psiBC = psiB[2]: movement probability from site B to site C
  # psiCA = psiC[1]: movement probability from site C to site A
  # psiCB = psiC[2]: movement probability from site C to site B
  # psiCC: movement probability from site C to site C (reference)
  # pA: recapture probability site A
  # pB: recapture probability site B
  # pC: recapture probability site C
  # ------------------------------------------------
  # States (z):
  # 1 alive at A
  # 2 alive at B
  # 2 alive at C
  # 3 dead
  # Observations (y):
  # 1 not seen
  # 2 seen at A
  # 3 seen at B
  # 3 seen at C
  # ------------------------------------------------

  # Priors
  phiA ~ dunif(0, 1)
  phiB ~ dunif(0, 1)
  phiC ~ dunif(0, 1)
  pA ~ dunif(0, 1)
  pB ~ dunif(0, 1)
  pC ~ dunif(0, 1)
  # transitions: multinomial logit
  # normal priors on logit of all but one transition probs
  for (i in 1:2){
    lpsiA[i] ~ dnorm(0, sd = 1000)
    lpsiB[i] ~ dnorm(0, sd = 1000)
    lpsiC[i] ~ dnorm(0, sd = 1000)
  }
  # constrain the transitions such that their sum is < 1
  for (i in 1:2){
    psiA[i] <- exp(lpsiA[i]) / (1 + exp(lpsiA[1]) + exp(lpsiA[2]))
    psiB[i] <- exp(lpsiB[i]) / (1 + exp(lpsiB[1]) + exp(lpsiB[2]))
    psiC[i] <- exp(lpsiC[i]) / (1 + exp(lpsiC[1]) + exp(lpsiC[2]))
  }
  # last transition probability
```

```
  psiA[3] <- 1 - psiA[1] - psiA[2]
  psiB[3] <- 1 - psiB[1] - psiB[2]
  psiC[3] <- 1 - psiC[1] - psiC[2]

  # probabilities of state z(t+1) given z(t)
  gamma[1,1] <- phiA * psiA[1]
  gamma[1,2] <- phiA * psiA[2]
  gamma[1,3] <- phiA * psiA[3]
  gamma[1,4] <- 1 - phiA
  gamma[2,1] <- phiB * psiB[1]
  gamma[2,2] <- phiB * psiB[2]
  gamma[2,3] <- phiB * psiB[3]
  gamma[2,4] <- 1 - phiB
  gamma[3,1] <- phiC * psiC[1]
  gamma[3,2] <- phiC * psiC[2]
  gamma[3,3] <- phiC * psiC[3]
  gamma[3,4] <- 1 - phiC
  gamma[4,1] <- 0
  gamma[4,2] <- 0
  gamma[4,3] <- 0
  gamma[4,4] <- 1

  # probabilities of y(t) given z(t)
  omega[1,1] <- 1 - pA     # Pr(alive A t -> non-detected t)
  omega[1,2] <- pA         # Pr(alive A t -> detected A t)
  omega[1,3] <- 0          # Pr(alive A t -> detected B t)
  omega[1,4] <- 0          # Pr(alive A t -> detected C t)
  omega[2,1] <- 1 - pB     # Pr(alive B t -> non-detected t)
  omega[2,2] <- 0          # Pr(alive B t -> detected A t)
  omega[2,3] <- pB         # Pr(alive B t -> detected B t)
  omega[2,4] <- 0          # Pr(alive B t -> detected C t)
  omega[3,1] <- 1 - pC     # Pr(alive C t -> non-detected t)
  omega[3,2] <- 0          # Pr(alive C t -> detected A t)
  omega[3,3] <- 0          # Pr(alive C t -> detected B t)
  omega[3,4] <- pC         # Pr(alive C t -> detected C t)
  omega[4,1] <- 1          # Pr(dead t -> non-detected t)
  omega[4,2] <- 0          # Pr(dead t -> detected A t)
  omega[4,3] <- 0          # Pr(dead t -> detected B t)
  omega[4,4] <- 0          # Pr(dead t -> detected C t)

  # likelihood
  for (i in 1:N){
    # latent state at first capture
    z[i,first[i]] <- y[i,first[i]] - 1
    for (t in (first[i]+1):K){
      # z(t) given z(t-1)
      z[i,t] ~ dcat(gamma[z[i,t-1],1:4])
      # y(t) given z(t)
      y[i,t] ~ dcat(omega[z[i,t],1:4])
    }
  }
})
```

List of data.

```
my.data <- list(y = y + 1)
```

List of constants.

```
my.constants <- list(first = first,
                     K = ncol(y),
                     N = nrow(y))
```

Initial values.

```
zinits <- y
zinits[zinits==0] <- sample(c(1,2,3), sum(zinits==0), replace = TRUE)
initial.values <- function(){list(phiA = runif(1, 0, 1),
                                  phiB = runif(1, 0, 1),
                                  phiC = runif(1, 0, 1),
                                  lpsiA = rnorm(2, 0, 10),
                                  lpsiB = rnorm(2, 0, 10),
                                  lpsiC = rnorm(2, 0, 10),
                                  pA = runif(1, 0, 1),
                                  pB = runif(1, 0, 1),
                                  pC = runif(1, 0, 1),
                                  z = zinits)}
```

Parameters to monitor.

```
parameters.to.save <- c("phiA", "phiB", "phiC", "psiA", "psiB", "psiC","pA", "pB", "pC")
parameters.to.save
```

```
[1] "phiA" "phiB" "phiC" "psiA" "psiB" "psiC" "pA"   "pB"   "pC"
```

Run nimble.

```
mcmc.multisite <- nimbleMCMC(code = multisite,
                             constants = my.constants,
                             data = my.data,
                             inits = initial.values,
                             monitors = parameters.to.save,
                             niter = n.iter,
                             nburnin = n.burnin,
                             nchains = n.chains)
```

```
|-------------|-------------|-------------|-------------|
|-----------------------------------------------------|
|-------------|-------------|-------------|-------------|
|-----------------------------------------------------|
```

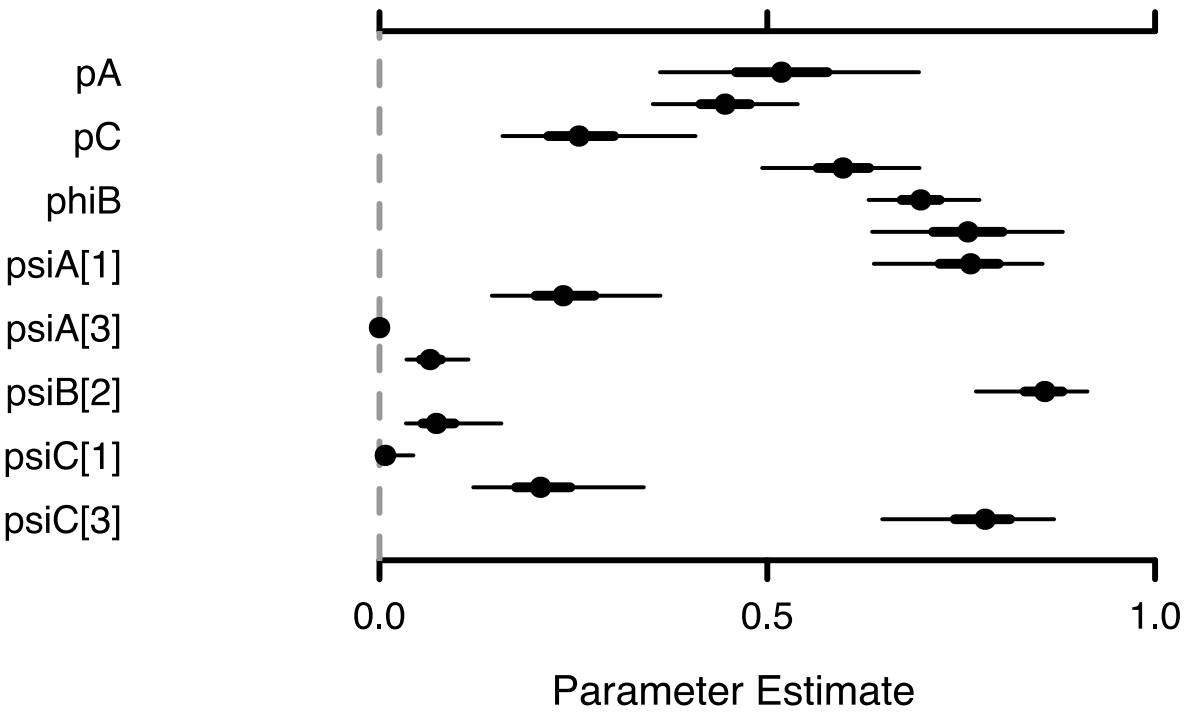
Inspect the results.

```
MCMCsummary(mcmc.multisite, round = 2)
```

```
        mean   sd 2.5%  50% 97.5% Rhat n.eff
pA      0.52 0.09 0.36 0.52  0.70 1.00   292
pB      0.45 0.05 0.35 0.45  0.54 1.04   300
pC      0.26 0.06 0.16 0.26  0.41 1.01   215
phiA    0.60 0.05 0.49 0.60  0.70 1.01   514
phiB    0.70 0.04 0.63 0.70  0.77 1.03   293
phiC    0.76 0.06 0.64 0.76  0.88 1.01   274
psiA[1] 0.76 0.06 0.64 0.76  0.85 1.00   560
psiA[2] 0.24 0.06 0.14 0.24  0.36 1.00   567
psiA[3] 0.00 0.00 0.00 0.00  0.01 1.32    28
psiB[1] 0.07 0.02 0.03 0.07  0.11 1.00   644
psiB[2] 0.85 0.04 0.77 0.86  0.91 1.03   255
psiB[3] 0.08 0.03 0.03 0.07  0.16 1.07   139
psiC[1] 0.01 0.01 0.00 0.01  0.04 1.01  1132
psiC[2] 0.21 0.06 0.12 0.21  0.34 1.02   468
psiC[3] 0.77 0.06 0.65 0.78  0.87 1.02   445
```

Caterpillar plot.

```
MCMCplot(mcmc.multisite)
```



Dirichlet prior

Another method is to use Dirichlet priors for the movement parameters. Let's write the model.

```
multisite <- nimbleCode({

  # ------------------------------------------------
  # Parameters:
  # phiA: survival probability site A
  # phiB: survival probability site B
  # phiC: survival probability site B
  # psiAA = psiA[1]: movement probability from site A to site A (reference)
  # psiAB = psiA[2]: movement probability from site A to site B
  # psiAC = psiA[3]: movement probability from site A to site C
  # psiBA = psiB[1]: movement probability from site B to site A
  # psiBB = psiB[2]: movement probability from site B to site B (reference)
  # psiBC = psiB[3]: movement probability from site B to site C
  # psiCA = psiC[1]: movement probability from site C to site A
  # psiCB = psiC[2]: movement probability from site C to site B
  # psiCC = psiC[3]: movement probability from site C to site C (reference)
  # pA: recapture probability site A
  # pB: recapture probability site B
  # pC: recapture probability site C
  # ------------------------------------------------
  # States (z):
  # 1 alive at A
  # 2 alive at B
  # 2 alive at C
  # 3 dead
  # Observations (y):
  # 1 not seen
  # 2 seen at A
  # 3 seen at B
  # 3 seen at C
  # ------------------------------------------------

  # Priors
  phiA ~ dunif(0, 1)
  phiB ~ dunif(0, 1)
  phiC ~ dunif(0, 1)
  pA ~ dunif(0, 1)
  pB ~ dunif(0, 1)
  pC ~ dunif(0, 1)
  # transitions: Dirichlet priors
  psiA[1:3] ~ ddirch(alpha[1:3])
  psiB[1:3] ~ ddirch(alpha[1:3])
  psiC[1:3] ~ ddirch(alpha[1:3])

  # probabilities of state z(t+1) given z(t)
  gamma[1,1] <- phiA * psiA[1]
  gamma[1,2] <- phiA * psiA[2]
  gamma[1,3] <- phiA * psiA[3]
  gamma[1,4] <- 1 - phiA
  gamma[2,1] <- phiB * psiB[1]
  gamma[2,2] <- phiB * psiB[2]
  gamma[2,3] <- phiB * psiB[3]
  gamma[2,4] <- 1 - phiB
```

```
  gamma[3,1] <- phiC * psiC[1]
  gamma[3,2] <- phiC * psiC[2]
  gamma[3,3] <- phiC * psiC[3]
  gamma[3,4] <- 1 - phiC
  gamma[4,1] <- 0
  gamma[4,2] <- 0
  gamma[4,3] <- 0
  gamma[4,4] <- 1

  # probabilities of y(t) given z(t)
  omega[1,1] <- 1 - pA     # Pr(alive A t -> non-detected t)
  omega[1,2] <- pA         # Pr(alive A t -> detected A t)
  omega[1,3] <- 0          # Pr(alive A t -> detected B t)
  omega[1,4] <- 0          # Pr(alive A t -> detected C t)
  omega[2,1] <- 1 - pB     # Pr(alive B t -> non-detected t)
  omega[2,2] <- 0          # Pr(alive B t -> detected A t)
  omega[2,3] <- pB         # Pr(alive B t -> detected B t)
  omega[2,4] <- 0          # Pr(alive B t -> detected C t)
  omega[3,1] <- 1 - pC     # Pr(alive C t -> non-detected t)
  omega[3,2] <- 0          # Pr(alive C t -> detected A t)
  omega[3,3] <- 0          # Pr(alive C t -> detected B t)
  omega[3,4] <- pC         # Pr(alive C t -> detected C t)
  omega[4,1] <- 1          # Pr(dead t -> non-detected t)
  omega[4,2] <- 0          # Pr(dead t -> detected A t)
  omega[4,3] <- 0          # Pr(dead t -> detected B t)
  omega[4,4] <- 0          # Pr(dead t -> detected C t)

  # likelihood
  for (i in 1:N){
    # latent state at first capture
    z[i,first[i]] <- y[i,first[i]] - 1
    for (t in (first[i]+1):K){
      # z(t) given z(t-1)
      z[i,t] ~ dcat(gamma[z[i,t-1],1:4])
      # y(t) given z(t)
      y[i,t] ~ dcat(omega[z[i,t],1:4])
    }
  }
})
```

Data in a list.

```
my.data <- list(y = y + 1)
```

Constants in a list.

```
my.constants <- list(first = first,
                     K = ncol(y),
                     N = nrow(y),
                     alpha = c(1, 1, 1))
```

Initial values.

```
zinits <- y
zinits[zinits==0] <- sample(c(1,2,3), sum(zinits==0), replace = TRUE)
initial.values <- function(){list(phiA = runif(1, 0, 1),
                                  phiB = runif(1, 0, 1),
                                  phiC = runif(1, 0, 1),
                                  psiA = rep(1/3, 3),
                                  psiB = rep(1/3, 3),
                                  psiC = rep(1/3, 3),
                                  pA = runif(1, 0, 1),
                                  pB = runif(1, 0, 1),
                                  pC = runif(1, 0, 1),
                                  z = zinits)}
```

Run nimble.

```
mcmc.multisite <- nimbleMCMC(code = multisite,
                             constants = my.constants,
                             data = my.data,
                             inits = initial.values,
                             monitors = parameters.to.save,
                             niter = n.iter,
                             nburnin = n.burnin,
                             nchains = n.chains)
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

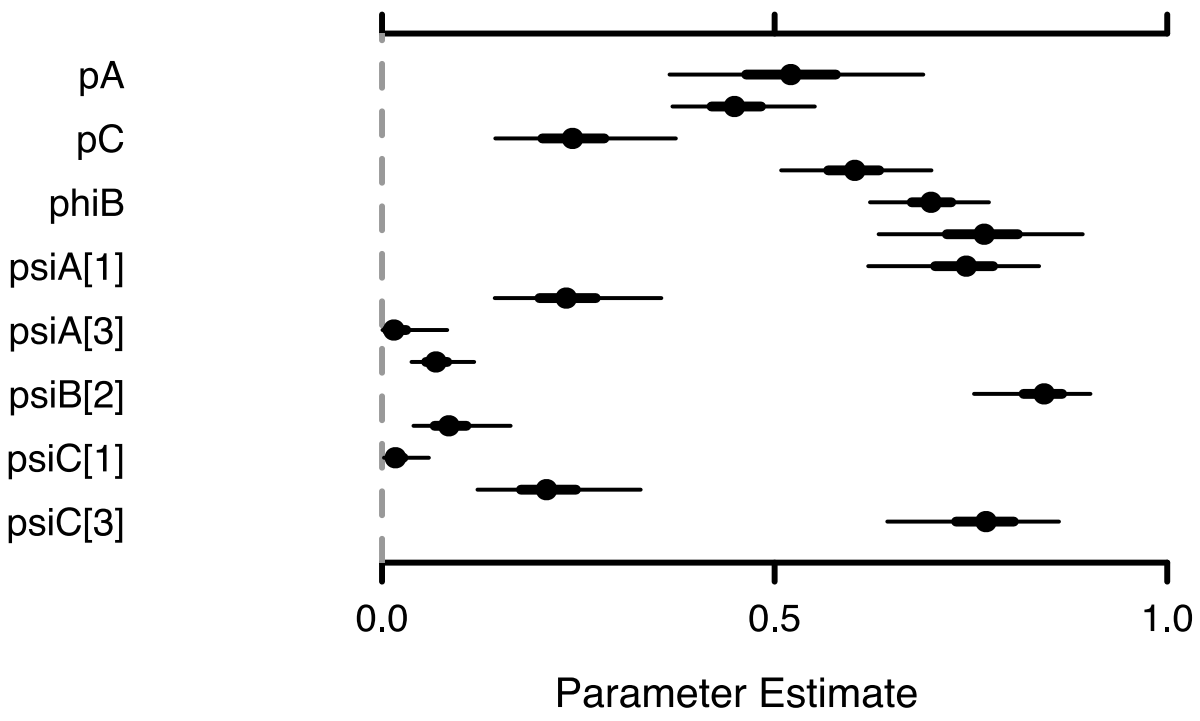
Inspect results. Compare to the estimates we obtained with the multinomial logit link method.

```
MCMCsummary(mcmc.multisite, round = 2)
```

```
        mean   sd 2.5%  50% 97.5% Rhat n.eff
pA      0.52 0.08 0.37 0.52  0.69 1.01   327
pB      0.45 0.05 0.37 0.45  0.55 1.00   250
pC      0.25 0.06 0.14 0.24  0.37 1.06   225
phiA    0.60 0.05 0.51 0.60  0.70 1.00   566
phiB    0.70 0.04 0.62 0.70  0.77 1.01   326
phiC    0.76 0.07 0.63 0.77  0.89 1.13   203
psiA[1] 0.74 0.06 0.62 0.74  0.84 1.00   661
psiA[2] 0.24 0.05 0.14 0.23  0.36 1.00   765
psiA[3] 0.02 0.02 0.00 0.02  0.08 1.01   473
psiB[1] 0.07 0.02 0.04 0.07  0.12 1.01   715
psiB[2] 0.84 0.04 0.75 0.84  0.90 1.03   462
psiB[3] 0.09 0.03 0.04 0.09  0.16 1.02   386
psiC[1] 0.02 0.01 0.00 0.02  0.06 1.00  1072
psiC[2] 0.21 0.05 0.12 0.21  0.33 1.04   736
psiC[3] 0.77 0.05 0.64 0.77  0.86 1.04   553
```

Caterpillar plots.

```
MCMCplot(mcmc.multisite)
```



Multistate model

Now we illustrate how one can replace site by breeding state and address another question in evolution ecology: life-history trade-offs. We use data on Soory shearwaters that were kindly provided by David Fletcher. Birds are seen as breeders or non-breeders.

```
titis <- read_csv2("titis.csv", col_names = FALSE)
y <-  as.matrix(titis)
head(y)
```

```
     X1 X2 X3 X4 X5 X6 X7
[1,]  0  0  0  0  0  0  1
[2,]  0  0  0  0  0  0  1
[3,]  0  0  0  0  0  0  1
[4,]  0  0  0  0  0  0  1
[5,]  0  0  0  0  0  0  1
[6,]  0  0  0  0  0  0  1
```

The code is very similar to that for the model above with two sites, i.e. the structure is the same, but the transition probabilities have different interpretations.

```
multistate <- nimbleCode({

  # ------------------------------------------------
  # Parameters:
  # phiB: survival probability state B
  # phiNB: survival probability state NB
  # psiBNB: transition probability from B to NB
  # psiNBB: transition probability from NB to B
  # pB: recapture probability B
  # pNB: recapture probability NB
  # ------------------------------------------------
  # States (z):
  # 1 alive B
  # 2 alive NB
  # 3 dead
  # Observations (y):
  # 1 not seen
  # 2 seen as B
  # 3 seen as NB
  # ------------------------------------------------

  # priors
  phiB ~ dunif(0, 1)
  phiNB ~ dunif(0, 1)
  psiBNB ~ dunif(0, 1)
  psiNBB ~ dunif(0, 1)
  pB ~ dunif(0, 1)
  pNB ~ dunif(0, 1)

  # probabilities of state z(t+1) given z(t)
  gamma[1,1] <- phiB * (1 - psiBNB)
  gamma[1,2] <- phiB * psiBNB
  gamma[1,3] <- 1 - phiB
  gamma[2,1] <- phiNB * psiNBB
  gamma[2,2] <- phiNB * (1 - psiNBB)
  gamma[2,3] <- 1 - phiNB
  gamma[3,1] <- 0
  gamma[3,2] <- 0
  gamma[3,3] <- 1

  # probabilities of y(t) given z(t)
  omega[1,1] <- 1 - pB    # Pr(alive B t -> non-detected t)
  omega[1,2] <- pB        # Pr(alive B t -> detected B t)
  omega[1,3] <- 0         # Pr(alive B t -> detected NB t)
  omega[2,1] <- 1 - pNB   # Pr(alive NB t -> non-detected t)
  omega[2,2] <- 0         # Pr(alive NB t -> detected B t)
  omega[2,3] <- pNB       # Pr(alive NB t -> detected NB t)
  omega[3,1] <- 1         # Pr(dead t -> non-detected t)
  omega[3,2] <- 0         # Pr(dead t -> detected N t)
  omega[3,3] <- 0         # Pr(dead t -> detected NB t)

  # likelihood
  for (i in 1:N){
```

```
    # latent state at first capture
    z[i,first[i]] <- y[i,first[i]] - 1
    for (t in (first[i]+1):K){
      # z(t) given z(t-1)
      z[i,t] ~ dcat(gamma[z[i,t-1],1:3])
      # y(t) given z(t)
      y[i,t] ~ dcat(omega[z[i,t],1:3])
    }
  }
})
```

Data in a list.

```
my.data <- list(y = y + 1)
```

Get data of first capture.

```
get.first <- function(x) min(which(x != 0))
first <- apply(y, 1, get.first)
```

Constants in a list.

```
my.constants <- list(first = first,
                     K = ncol(y),
                     N = nrow(y))
```

Initial values. Same as before, we initialize the latent states with the detections and non-detections. Then non-detections are replaced by a 1 or a 2 for breeder on non-breeder.

```
zinits <- y
zinits[zinits == 0] <- sample(c(1,2), sum(zinits == 0), replace = TRUE)
initial.values <- function(){list(phiNB = runif(1, 0, 1),
                                  phiB = runif(1, 0, 1),
                                  psiNBB = runif(1, 0, 1),
                                  psiBNB = runif(1, 0, 1),
                                  pNB = runif(1, 0, 1),
                                  pB = runif(1, 0, 1),
                                  z = zinits)}
```

Parameters to be monitored.

```
parameters.to.save <- c("phiNB", "phiB","psiNBB", "psiBNB", "pNB", "pB")
parameters.to.save
```

```
[1] "phiNB"  "phiB"   "psiNBB" "psiBNB" "pNB"    "pB"
```

MCMC details.

```
n.iter <- 5000
n.burnin <- 2500
n.chains <- 2
```

Run nimble.

```
mcmc.multistate <- nimbleMCMC(code = multistate,
                              constants = my.constants,
                              data = my.data,
                              inits = initial.values,
                              monitors = parameters.to.save,
                              niter = n.iter,
                              nburnin = n.burnin,
                              nchains = n.chains)
```

```
|-------------|-------------|-------------|-------------|
|-----------------------------------------------------|
|-------------|-------------|-------------|-------------|
|-----------------------------------------------------|
```

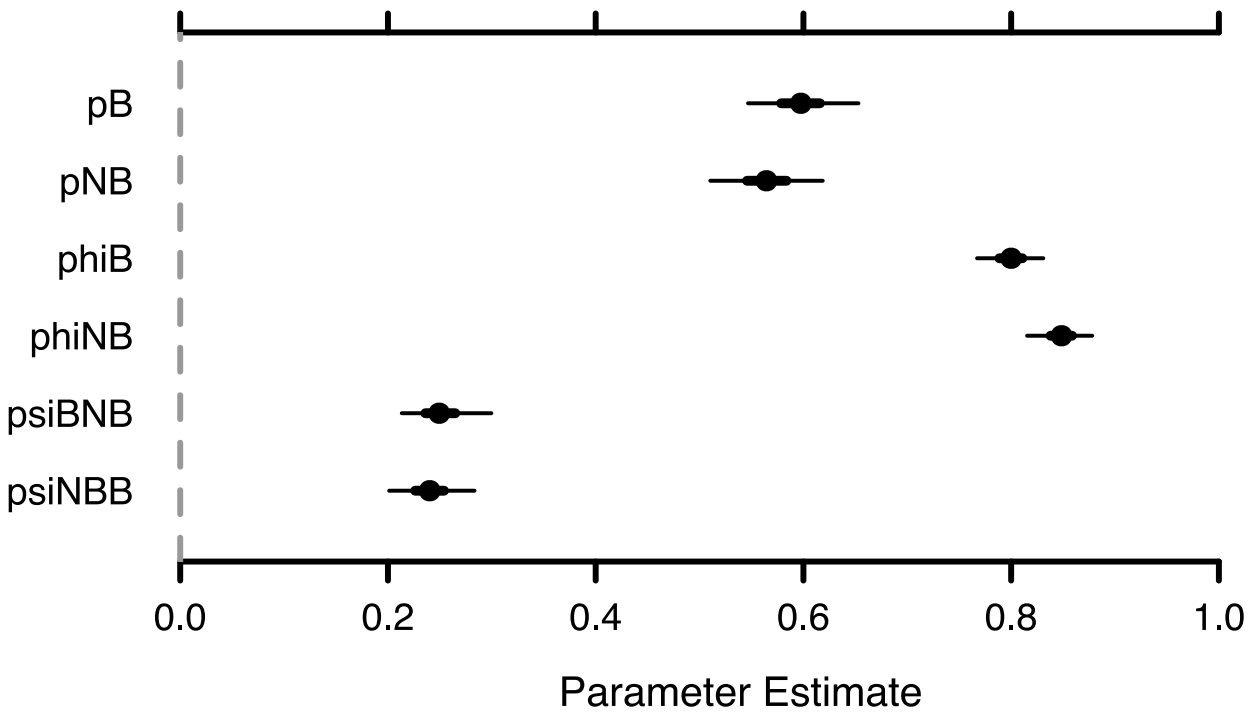
Let's inspect the results.

```
MCMCsummary(mcmc.multistate, round = 2)
```

```
       mean   sd 2.5%  50% 97.5% Rhat n.eff
pB     0.60 0.03 0.55 0.60  0.65 1.00   251
pNB    0.56 0.03 0.51 0.56  0.62 1.01   257
phiB   0.80 0.02 0.77 0.80  0.83 1.01   462
phiNB  0.85 0.02 0.82 0.85  0.88 1.02   468
psiBNB 0.25 0.02 0.21 0.25  0.30 1.00   309
psiNBB 0.24 0.02 0.20 0.24  0.28 1.01   447
```

Caterpillar plot of the parameter estimates.

```
MCMCplot(mcmc.multistate)
```



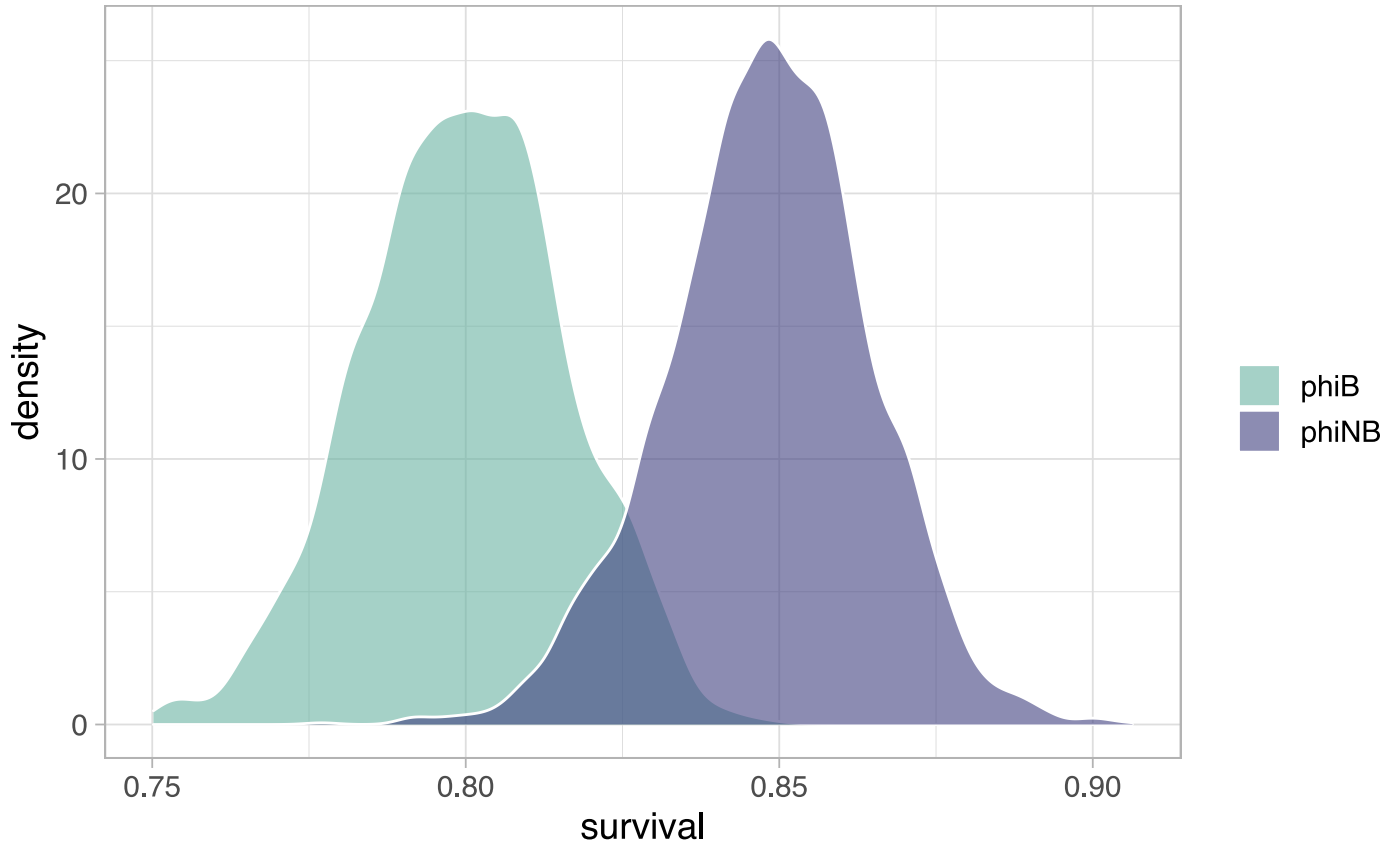
Non-breeder individuals seem to have a survival higher than breeder individuals, suggesting a trade-off between reproduction and survival. Let's compare graphically the survival of breeder and non-breeder individuals. First we gather the values generated for ϕ_B and ϕ_{NB} for the two chains.

```
phiB <- c(mcmc.multistate$chain1[,"phiB"], mcmc.multistate$chain2[,"phiB"])
phiNB <- c(mcmc.multistate$chain1[,"phiNB"], mcmc.multistate$chain2[,"phiNB"])
df <- data.frame(param = c(rep("phiB", length(phiB)), rep("phiNB", length(phiB))),
                 value = c(phiB, phiNB))
head(df)
```

```
  param     value
1  phiB 0.8347852
2  phiB 0.7826723
3  phiB 0.7762608
4  phiB 0.7731614
5  phiB 0.7731614
6  phiB 0.7731614
```

Then, we plot a histogram of the two posterior distributions.

```
df %>%
  ggplot(aes(x = value, fill = param)) +
  geom_density(color = "white", alpha = 0.6, position = 'identity') +
  scale_fill_manual(values = c("#69b3a2", "#404080")) +
  labs(fill = "", x = "survival")
```



To formally test this difference, one could fit a model with no state effect on survival and compare the two models with WAIC.

Class 7 live demo: Uncertainty in state assignment

The team

last updated: 2021-04-24

Introduction

In this demo, we illustrate how to fit models with several states in which the assignment of individuals to states is difficult to make with certainty. We start by breeding states then continue with disease states and end by illustrating how to incorporate individual heterogeneity in capture-recapture models.

```
library(tidyverse)
library(nimble)
library(MCMCvis)
```

Breeding state uncertainty

Let's get back to the analysis of the Sooty shearwater data. For some individuals, we have some uncertainty in assigning individuals in the breeding or non-breeding state (code 3 in the data).

```
titis <- read_csv2("titis_with_uncertainty.csv", col_names = FALSE)
```

```
ℹ Using ',' as decimal and '.' as grouping mark. Use `read_delim()` for more control.
```

```
── Column specification ────────────────────────────────────────────────────────
cols(
  X1 = col_double(),
  X2 = col_double(),
  X3 = col_double(),
  X4 = col_double(),
  X5 = col_double(),
  X6 = col_double(),
  X7 = col_double()
)
```

```
y <- as.matrix(titis)
head(y)
```

```
     X1 X2 X3 X4 X5 X6 X7
[1,]  0  0  0  0  0  0  3
[2,]  0  0  0  0  0  0  3
[3,]  0  0  0  0  0  0  3
[4,]  0  0  0  0  0  0  3
[5,]  0  0  0  0  0  0  3
[6,]  0  0  0  0  0  0  3
```

Let’s write down the model code.

```
multievent <- nimbleCode({

  # ------------------------------------------------
  # Parameters:
  # phiB: survival probability state B
  # phiNB: survival probability state NB
  # psiBNB: transition probability from B to NB
  # psiNBB: transition probability from NB to B
  # pB: recapture probability B
  # pNB: recapture probability NB
  # piB prob. of being in initial state breeder
  # betaNB prob to ascertain the breeding status of an individual encountered as non-bre
eder
  # betaB prob to ascertain the breeding status of an individual encountered as breeder
  # ------------------------------------------------
  # States (z):
  # 1 alive B
  # 2 alive NB
  # 3 dead
  # Observations (y):
  # 1 = non-detected
  # 2 = seen and ascertained as breeder
  # 3 = seen and ascertained as non-breeder
  # 4 = not ascertained
  # ------------------------------------------------

  # priors
  phiB ~ dunif(0, 1)
  phiNB ~ dunif(0, 1)
  psiBNB ~ dunif(0, 1)
  psiNBB ~ dunif(0, 1)
  pB ~ dunif(0, 1)
  pNB ~ dunif(0, 1)
  piB ~ dunif(0, 1)
  betaNB ~ dunif(0, 1)
  betaB ~ dunif(0, 1)

  # vector of initial stats probs
  delta[1] <- piB      # prob. of being in initial state B
  delta[2] <- 1 - piB  # prob. of being in initial state NB
  delta[3] <- 0        # prob. of being in initial state dead

  # probabilities of state z(t+1) given z(t)
  gamma[1,1] <- phiB * (1 - psiBNB)
  gamma[1,2] <- phiB * psiBNB
  gamma[1,3] <- 1 - phiB
  gamma[2,1] <- phiNB * psiNBB
  gamma[2,2] <- phiNB * (1 - psiNBB)
  gamma[2,3] <- 1 - phiNB
  gamma[3,1] <- 0
  gamma[3,2] <- 0
  gamma[3,3] <- 1
```

```
  # probabilities of y(t) given z(t)
  omega[1,1] <- 1 - pB             # Pr(alive B t -> non-detected t)
  omega[1,2] <- pB * betaB         # Pr(alive B t -> detected B t)
  omega[1,3] <- 0                  # Pr(alive B t -> detected NB t)
  omega[1,4] <- pB * (1 - betaB)   # Pr(alive B t -> detected U t)
  omega[2,1] <- 1 - pNB            # Pr(alive NB t -> non-detected t)
  omega[2,2] <- 0                  # Pr(alive NB t -> detected B t)
  omega[2,3] <- pNB * betaNB       # Pr(alive NB t -> detected NB t)
  omega[2,4] <- pNB * (1 - betaNB) # Pr(alive NB t -> detected U t)
  omega[3,1] <- 1                  # Pr(dead t -> non-detected t)
  omega[3,2] <- 0                  # Pr(dead t -> detected N t)
  omega[3,3] <- 0                  # Pr(dead t -> detected NB t)
  omega[3,4] <- 0                  # Pr(dead t -> detected U t)

  omega.init[1,1] <- 0                  # Pr(alive B t = 1 -> non-detected t = 1)
  omega.init[1,2] <- betaB              # Pr(alive B t = 1 -> detected B t = 1)
  omega.init[1,3] <- 0                  # Pr(alive B t = 1 -> detected NB t = 1)
  omega.init[1,4] <- 1 - betaB          # Pr(alive B t = 1 -> detected U t = 1)
  omega.init[2,1] <- 0                  # Pr(alive NB t = 1 -> non-detected t = 1)
  omega.init[2,2] <- 0                  # Pr(alive NB t = 1 -> detected B t = 1)
  omega.init[2,3] <- betaNB             # Pr(alive NB t = 1 -> detected NB t = 1)
  omega.init[2,4] <- 1 - betaNB         # Pr(alive NB t = 1 -> detected U t = 1)
  omega.init[3,1] <- 1                  # Pr(dead t = 1 -> non-detected t = 1)
  omega.init[3,2] <- 0                  # Pr(dead t = 1 -> detected N t = 1)
  omega.init[3,3] <- 0                  # Pr(dead t = 1 -> detected NB t = 1)
  omega.init[3,4] <- 0                  # Pr(dead t = 1 -> detected U t = 1)

  # likelihood
  for (i in 1:N){
    # latent state at first capture
    z[i,first[i]] ~ dcat(delta[1:3])
    y[i,first[i]] ~ dcat(omega.init[z[i,first[i]],1:4])
    for (t in (first[i]+1):K){
      # z(t) given z(t-1)
      z[i,t] ~ dcat(gamma[z[i,t-1],1:3])
      # y(t) given z(t)
      y[i,t] ~ dcat(omega[z[i,t],1:4])
    }
  }
})
```

Get the data of first capture.

```
get.first <- function(x) min(which(x != 0))
first <- apply(y, 1, get.first)
```

Constants in a list.

```
my.constants <- list(first = first,
                     K = ncol(y),
                     N = nrow(y))
```

Data in a list.

```
my.data <- list(y = y + 1)
```

Initial values.

```
zinit <- y
zinit[zinit==3] <- sample(c(1,2), sum(zinit==3), replace = TRUE)
for (i in 1:nrow(y)) {
  for (j in 1:ncol(y)) {
    if (j > first[i] & y[i,j]==0) {zinit[i,j] <- which(rmultinom(1, 1, c(1/2,1/2))==1)}
    if (j < first[i]) {zinit[i,j] <- 0}
  }
}
zinit <- as.matrix(zinit)
initial.values <- function(){list(phiNB = runif(1, 0, 1),
                                  phiB = runif(1, 0, 1),
                                  psiNBB = runif(1, 0, 1),
                                  psiBNB = runif(1, 0, 1),
                                  pNB = runif(1, 0, 1),
                                  pB = runif(1, 0, 1),
                                  piB = runif(1, 0, 1),
                                  betaB = runif(1, 0, 1),
                                  betaNB = runif(1, 0, 1),
                                  z = zinit)}
```

Parameters to be monitored.

```
parameters.to.save <- c("phiB",
                        "phiNB",
                        "psiNBB",
                        "psiBNB",
                        "piB",
                        "pB",
                        "pNB",
                        "betaNB",
                        "betaB")
```

MCMC details.

```
n.iter <- 5000
n.burnin <- 2500
n.chains <- 2
```

Run nimble.

```
mcmc.multievent <- nimbleMCMC(code = multievent,
                              constants = my.constants,
                              data = my.data,
                              inits = initial.values,
                              monitors = parameters.to.save,
                              niter = n.iter,
                              nburnin = n.burnin,
                              nchains = n.chains)
```

```
defining model...
```

```
building model...
```

```
setting data and initial values...
```

```
running calculate on model (any error reports that follow may simply reflect missing val
ues in model variables) ...
checking model sizes and dimensions...
checking model calculations...
model building finished.
compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilat
ion details.
compilation finished.
running chain 1...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

```
running chain 2...
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

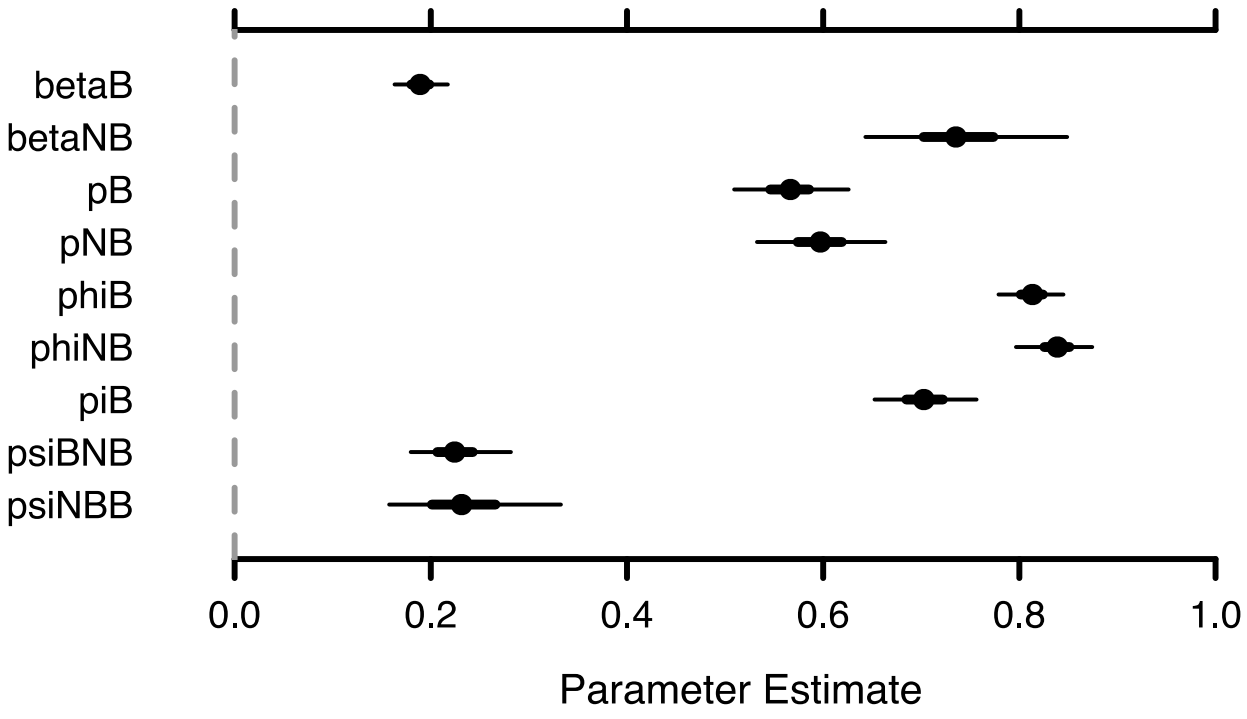
Let's have a look to the results. Breeders were mostly assigned as uncertain, while non-breeders were positively assigned.

```
MCMCsummary(mcmc.multievent, round = 2)
```

```
       mean   sd 2.5%  50% 97.5% Rhat n.eff
betaB  0.19 0.01 0.16 0.19  0.22 1.04   224
betaNB 0.74 0.05 0.64 0.74  0.85 1.10    63
pB     0.57 0.03 0.51 0.57  0.63 1.00   203
pNB    0.60 0.03 0.53 0.60  0.66 1.00   160
phiB   0.81 0.02 0.78 0.81  0.84 1.01   308
phiNB  0.84 0.02 0.80 0.84  0.87 1.01   301
piB    0.70 0.03 0.65 0.70  0.76 1.05    90
psiBNB 0.23 0.03 0.18 0.22  0.28 1.01   206
psiNBB 0.24 0.05 0.16 0.23  0.33 1.11    69
```

Caterpillar plot of the parameter estimates.

```
MCMCplot(mcmc.multievent)
```



Disease state uncertainty

We now turn to the analysis of data from a study of the dynamics of conjunctivitis in the house finch (Carpodacus mexicanus Müller). A challenge in modeling disease dynamics involves the estimation of recovery and infection rates, which can be complicated when disease state is uncertain (seen at distance in the house finch case study). In the data we have 0 for a non-detection, 1 for a detection of a healthy individual, 2 for the detection of a sick individual and 3 for an individual for which we cannot say whether it is sane or ill.

```
y <- as.matrix(read.table("hofi_2000.txt"))
head(y)
```

```
     V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13
[1,]  0  0  0  0  0  0  0  0  3   0   0   0   0
[2,]  0  0  0  0  0  0  0  0  0   0   0   0   1
[3,]  0  0  0  0  0  0  3  0  0   0   0   0   1
[4,]  0  0  0  0  0  0  0  0  0   0   1   0   0
[5,]  0  0  0  0  0  0  3  3  1   0   0   0   1
[6,]  0  1  0  0  0  0  0  0  0   0   0   0   0
```

Write the model. The model is similar to the model of the previous section.

```
hmm.disease <- nimbleCode({

  # priors
  phiH ~ dunif(0, 1)    # prior survival healthy
  phiI ~ dunif(0, 1)    # prior survival ill
  psiIH ~ dunif(0, 1)   # prior transition ill -> healthy
  psiHI ~ dunif(0, 1)   # prior transition healthy -> ill
  pH ~ dunif(0, 1)      # prior detection healthy
  pI ~ dunif(0, 1)      # prior detection ill
  pi ~ dunif(0, 1)      # prob init state healthy
  betaH ~ dunif(0,1)
  betaI ~ dunif(0,1)

  # HMM ingredients
  delta[1] <- pi          # Pr(healthy t = 1) = pi
  delta[2] <- 1 - pi      # Pr(ill t = 1) = 1 - pi
  delta[3] <- 0           # Pr(dead t = 1) = 0

  gamma[1,1] <- phiH * (1 - psiHI)      # Pr(H t -> H t+1)
  gamma[1,2] <- phiH * psiHI            # Pr(H t -> I t+1)
  gamma[1,3] <- 1 - phiH                # Pr(alive t -> dead t+1)
  gamma[2,1] <- phiI * psiIH            # Pr(I t -> H t+1)
  gamma[2,2] <- phiI * (1 - psiIH)      # Pr(I t -> I t+1)
  gamma[2,3] <- 1 - phiI                # Pr(alive t -> dead t+1)
  gamma[3,1] <- 0                       # Pr(dead t -> alive t+1)
  gamma[3,2] <- 0                       # Pr(dead t -> alive t+1)
  gamma[3,3] <- 1                       # Pr(dead t -> dead t+1)

  omega[1,1] <- 1 - pH           # Pr(H t -> non-detected t)
  omega[1,2] <- pH * betaH       # Pr(H t -> detected H t)
  omega[1,3] <- 0                # Pr(H t -> detected I t)
  omega[1,4] <- pH * (1 - betaH) # Pr(H t -> detected U t)
  omega[2,1] <- 1 - pI           # Pr(I t -> non-detected t)
  omega[2,2] <- 0                # Pr(I t -> detected H t)
  omega[2,3] <- pI * betaI       # Pr(I t -> detected I t)
  omega[2,4] <- pI * (1 - betaI) # Pr(I t -> detected U t)
  omega[3,1] <- 1                # Pr(dead t -> non-detected t)
  omega[3,2] <- 0                # Pr(dead t -> detected H t)
  omega[3,3] <- 0                # Pr(dead t -> detected I t)
  omega[3,4] <- 0                # Pr(dead t -> detected U t)

  omega.init[1,1] <- 0               # Pr(H t = 1 -> non-detected t = 1)
  omega.init[1,2] <- betaH           # Pr(H t = 1 -> detected H t = 1)
  omega.init[1,3] <- 0               # Pr(H t = 1 -> detected I t = 1)
  omega.init[1,4] <- 1 - betaH       # Pr(H t = 1 -> detected U t = 1)
  omega.init[2,1] <- 0               # Pr(I t = 1 -> non-detected t = 1)
  omega.init[2,2] <- 0               # Pr(I t = 1 -> detected H t = 1)
  omega.init[2,3] <- betaI           # Pr(I t = 1 -> detected I t = 1)
  omega.init[2,4] <- 1 - betaI       # Pr(I t = 1 -> detected U t = 1)
  omega.init[3,1] <- 1               # Pr(dead t = 1 -> non-detected t = 1)
  omega.init[3,2] <- 0               # Pr(dead t = 1 -> detected H t = 1)
  omega.init[3,3] <- 0               # Pr(dead t = 1 -> detected I t = 1)
  omega.init[3,4] <- 0               # Pr(dead t = 1 -> detected U t = 1)
```

```
  # likelihood
  for (i in 1:N){
    z[i,first[i]] ~ dcat(delta[1:3])
    y[i,first[i]] ~ dcat(omega.init[z[i,first[i]], 1:4])
    for (j in (first[i]+1):K){
      z[i,j] ~ dcat(gamma[z[i,j-1], 1:3])
      y[i,j] ~ dcat(omega[z[i,j], 1:4])
    }
  }
})
```

Get the date of first capture.

```
first <- apply(y, 1, function(x) min(which(x !=0)))
```

Constants in a list.

```
my.constants <- list(N = nrow(y), K = ncol(y), first = first)
```

Data in a list.

```
my.data <- list(y = y + 1)
```

Initial values.

```
zinit <- y
zinit[zinit==3] <- sample(c(1,2), sum(zinit==3), replace = TRUE)
for (i in 1:nrow(y)) {
  for (j in 1:ncol(y)) {
    if (j > first[i] & y[i,j]==0) {zinit[i,j] <- which(rmultinom(1, 1, c(1/2,1/2))==1)}
    if (j < first[i]) {zinit[i,j] <- 0}
  }
}
zinit <- as.matrix(zinit)
initial.values <- function() list(phiH = runif(1, 0, 1),
                                  phiI = runif(1, 0, 1),
                                  pH = runif(1, 0, 1),
                                  pI = runif(1, 0, 1),
                                  pi = runif(1, 0, 1),
                                  betaH = runif(1, 0, 1),
                                  betaI = runif(1, 0, 1),
                                  psiHI = runif(1, 0, 1),
                                  psiIH = runif(1, 0, 1),
                                  z = zinit)
```

Parameters to be monitored.

```
parameters.to.save <- c("phiH",
                        "phiI",
                        "pH",
                        "pI",
                        "pi",
                        "betaH",
                        "betaI",
                        "psiHI",
                        "psiIH")
```

MCMC details.

```
n.iter <- 40000
n.burnin <- 25000
n.chains <- 2
```

Run nimble.

```
out <- nimbleMCMC(code = hmm.disease,
                            constants = my.constants,
                            data = my.data,
                            inits = initial.values,
                            monitors = parameters.to.save,
                            niter = n.iter,
                            nburnin = n.burnin,
                            nchains = n.chains)
save(out, file = "house_finch.RData")
```

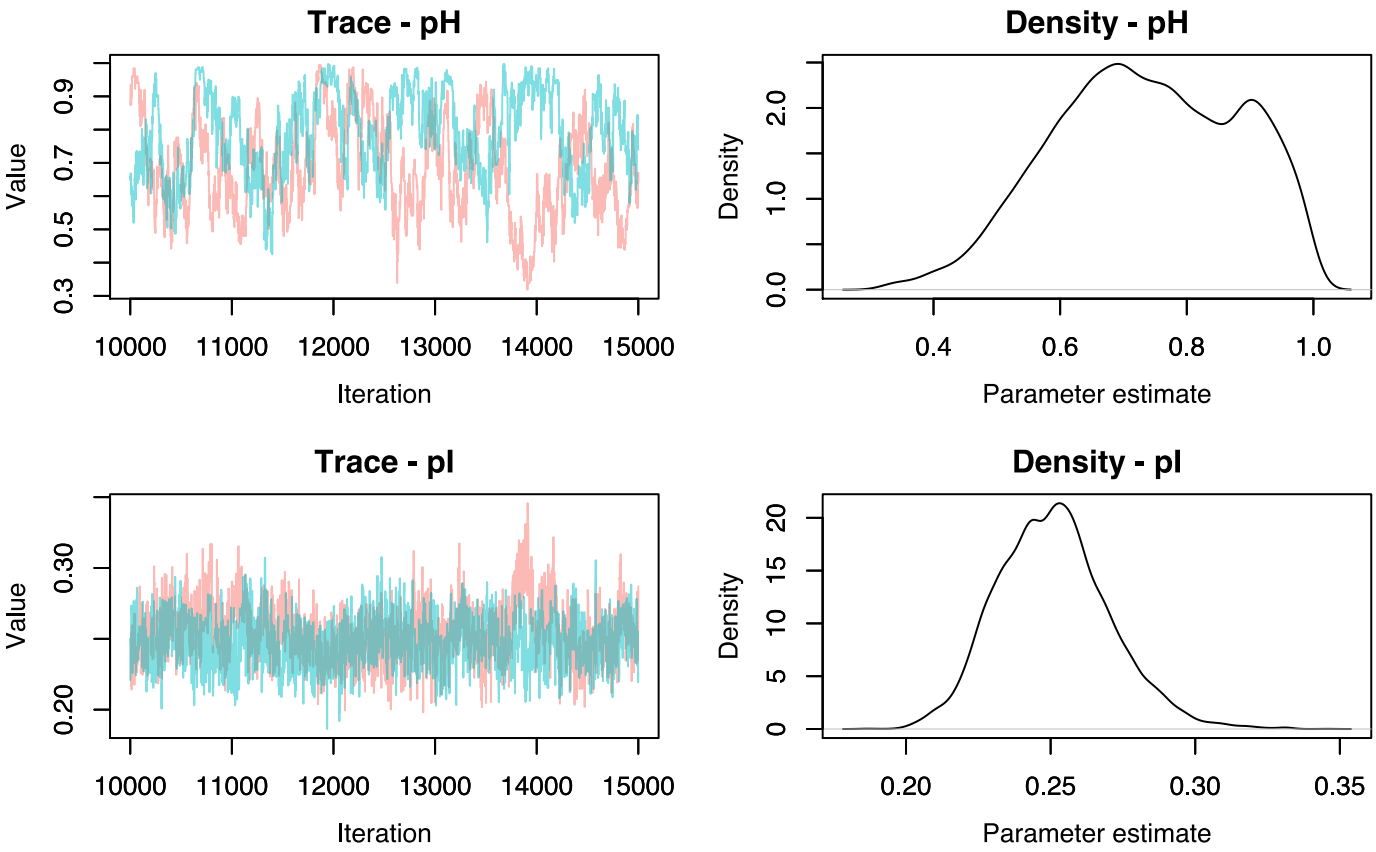
Inspect the results.

```
MCMCsummary(out, round = 2)
```

```
      mean   sd 2.5%  50% 97.5% Rhat n.eff
betaH 0.99 0.01 0.97 0.99  1.00 1.00  1301
betaI 0.05 0.01 0.03 0.05  0.08 1.00  6458
pH    0.75 0.15 0.45 0.77  0.98 1.27   156
pI    0.25 0.02 0.21 0.25  0.29 1.11   452
phiH  0.69 0.04 0.62 0.69  0.77 1.03   454
phiI  0.99 0.01 0.96 0.99  1.00 1.06   525
pi    0.96 0.01 0.93 0.96  0.98 1.00  3769
psiHI 0.80 0.06 0.66 0.81  0.88 1.20   298
psiIH 0.17 0.04 0.12 0.16  0.27 1.26   196
```

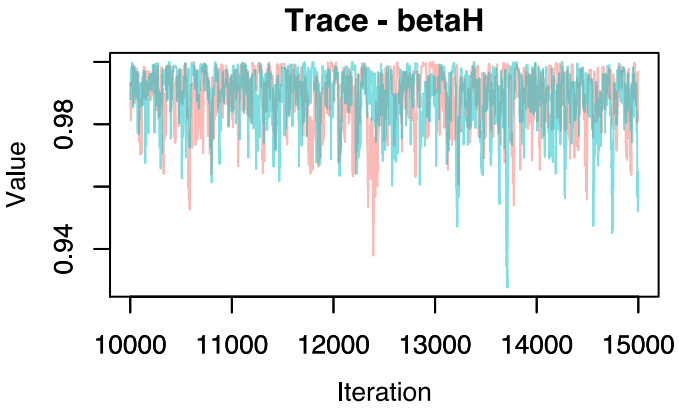
Let's have a look to the trace plots, and the estimated posterior distribution of the detection probabilities in healthy and ill states.

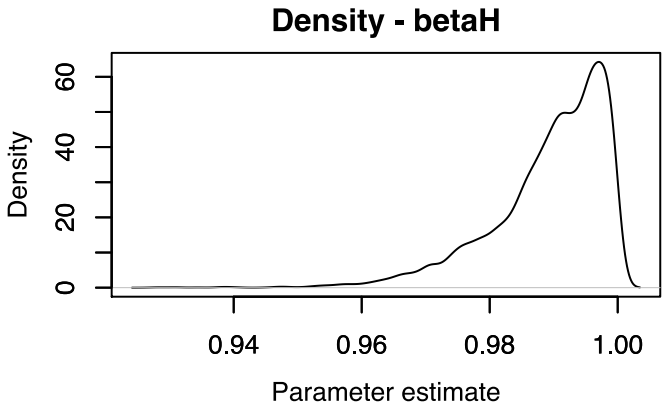
```
MCMCtrace(out, pdf = FALSE, params = c("pH", "pI"))
```

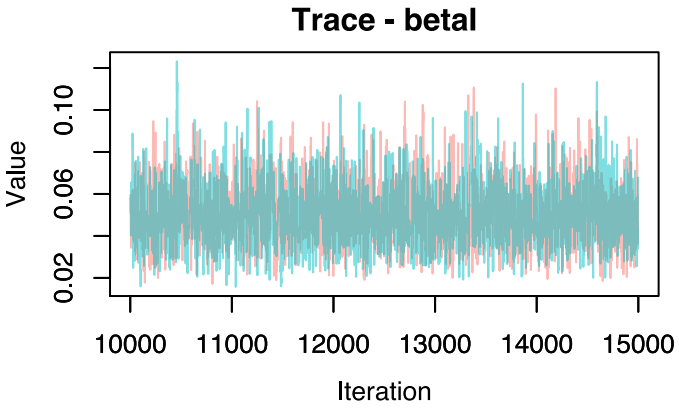


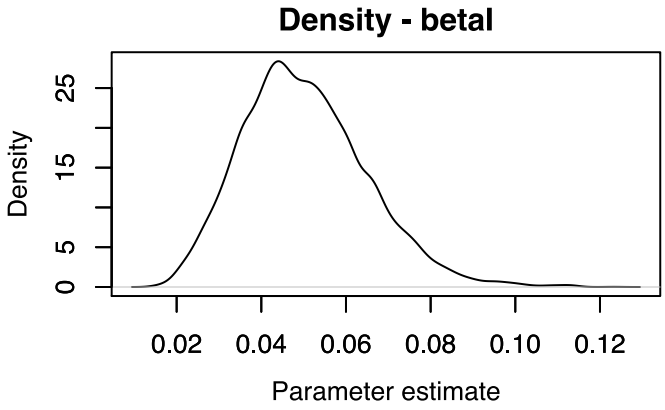
What about the assignment probabilities? The non-infected birds were positively identified, while most infected encounters were classified as unknown.

```
MCMCtrace(out, pdf = FALSE, params = c("betaH", "betaI"))
```



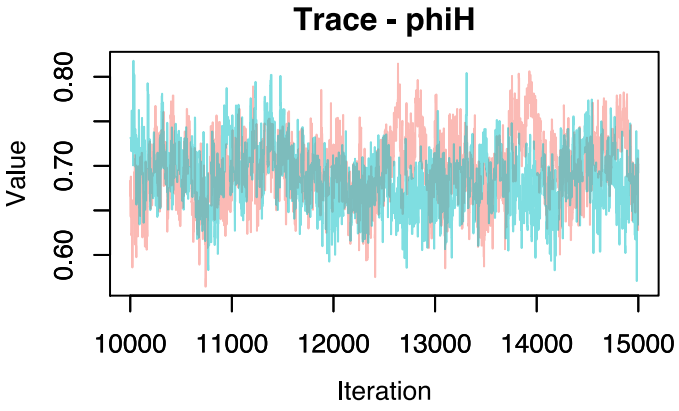


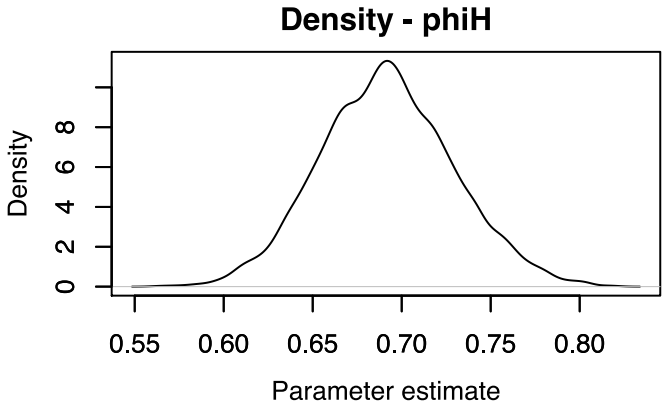


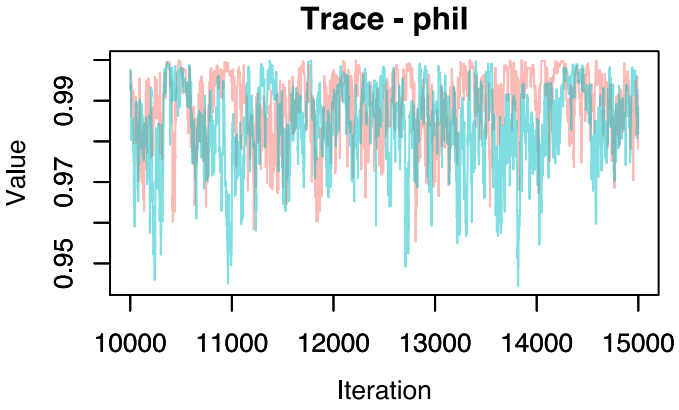


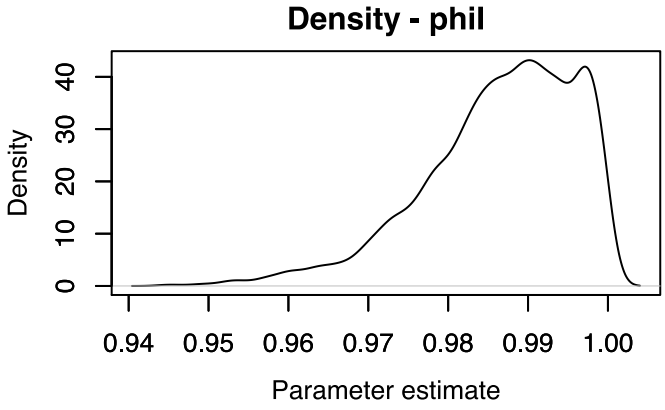
And survival?

```
MCMCtrace(out, pdf = FALSE, params = c("phiH", "phiI"))
```



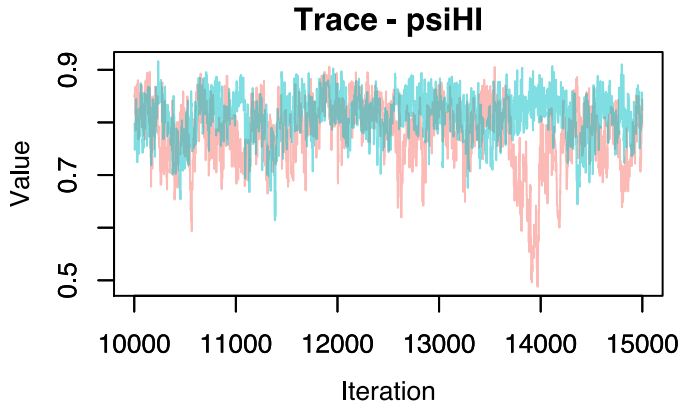


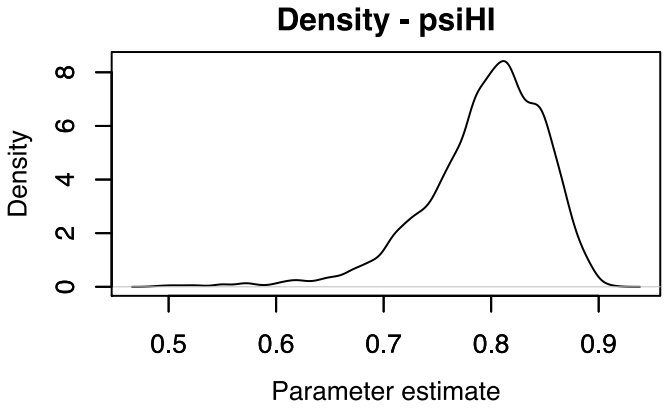


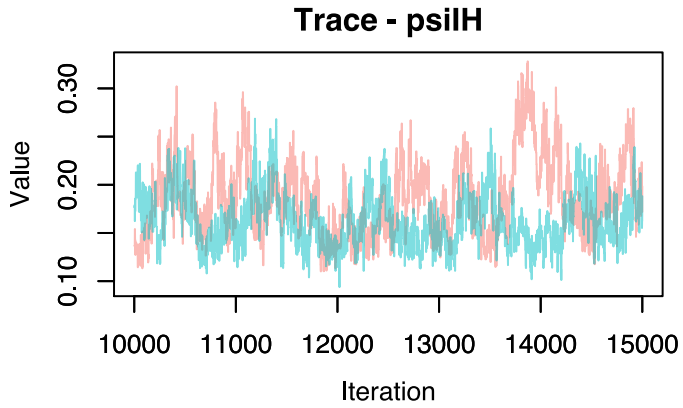


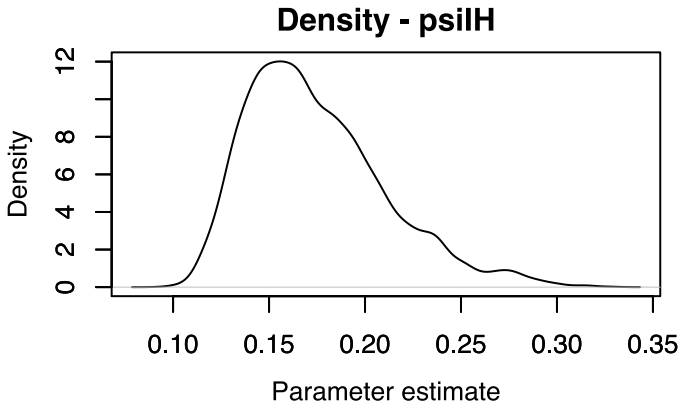
What about the infection and recovery rates?

```
MCMCtrace(out, pdf = FALSE, params = c("psiHI", "psiIH"))
```



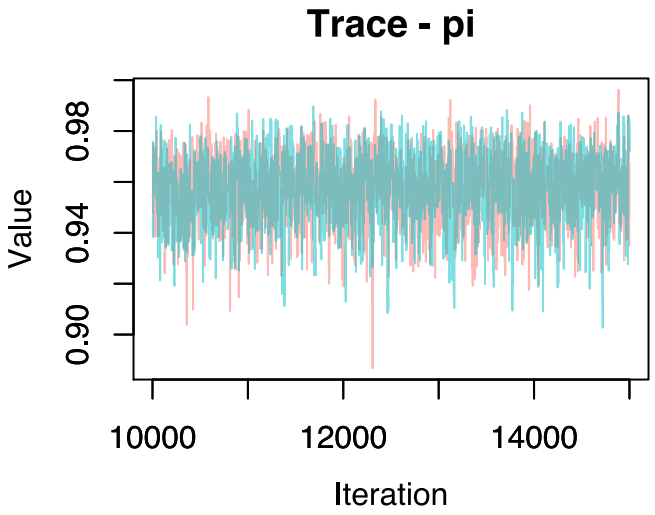


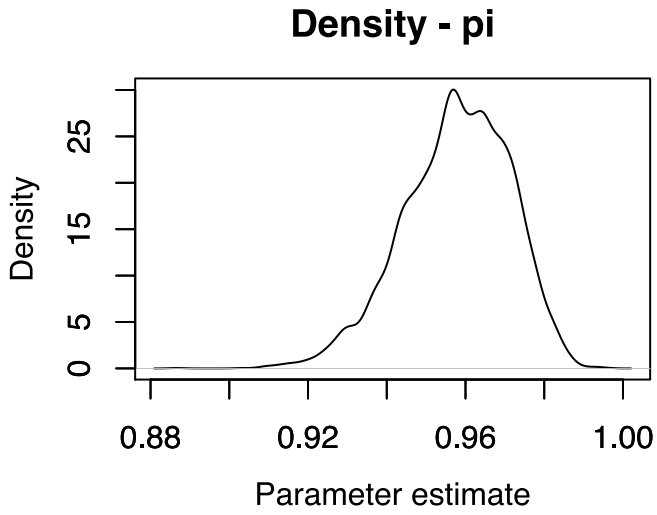




Last, the proportion of new marked individuals that are in the healthy state.

```
MCMCtrace(out, pdf = FALSE, params = c("pi"))
```





Individual heterogeneity

For our last example, we will see how to incorporate individual heterogeneity in capture-recapture models using finite mixtures. We will use a case study on gray wolves. The data were kindly provided by the French Office for Biodiversity.

```
y <- as.matrix(read.table("wolf.txt"))
# 1 seen
# 0 not seen
```

Let's write the model. Only thing to notice is that we have two alive states to represent two classes of individuals. We choose to have the detection probabilities dependent on the states and constant survival.

```
hmm.phipmix <- nimbleCode({

  # priors
  phi ~ dunif(0, 1) # prior survival
  p1 ~ dunif(0, 1) # prior detection
  p2 ~ dunif(0, 1) # prior detection
  pi ~ dunif(0, 1) # prob init state 1

  # HMM ingredients
  gamma[1,1] <- phi      # Pr(alive 1 t -> alive 1 t+1)
  gamma[1,2] <- 0        # Pr(alive 1 t -> alive 2 t+1) // no transition
  gamma[1,3] <- 1 - phi  # Pr(alive t -> dead t+1)
  gamma[2,1] <- 0        # Pr(alive 1 t -> alive 1 t+1) // no transition
  gamma[2,2] <- phi      # Pr(alive 1 t -> alive 2 t+1)
  gamma[2,3] <- 1 - phi  # Pr(alive t -> dead t+1)
  gamma[3,1] <- 0        # Pr(dead t -> alive 1 t+1)
  gamma[3,2] <- 0        # Pr(dead t -> alive 1 t+1)
  gamma[3,3] <- 1        # Pr(dead t -> dead t+1)
  delta[1] <- pi         # Pr(alive t = 1) = pi
  delta[2] <- 1 - pi     # Pr(alive t = 1) = 1 - pi
  delta[3] <- 0          # Pr(dead t = 1) = 0
  omega[1,1] <- 1 - p1   # Pr(alive state 1 t -> non-detected t)
  omega[1,2] <- p1       # Pr(alive state 1 t -> detected t)
  omega[2,1] <- 1 - p2   # Pr(alive state 2 t -> non-detected t)
  omega[2,2] <- p2       # Pr(alive state 2 t -> detected t)
  omega[3,1] <- 1        # Pr(dead t -> non-detected t)
  omega[3,2] <- 0        # Pr(dead t -> detected t)
  omega.init[1,1] <- 0   # Pr(alive state 1 t -> non-detected t)
  omega.init[1,2] <- 1   # Pr(alive state 1 t -> detected t)
  omega.init[2,1] <- 0   # Pr(alive state 2 t -> non-detected t)
  omega.init[2,2] <- 1   # Pr(alive state 2 t -> detected t)
  omega.init[3,1] <- 1   # Pr(dead t -> non-detected t)
  omega.init[3,2] <- 0   # Pr(dead t -> detected t)

  # likelihood
  for (i in 1:N){
    z[i,first[i]] ~ dcat(delta[1:3])
    y[i,first[i]] ~ dcat(omega.init[z[i,first[i]], 1:2])
    for (j in (first[i]+1):K){
      z[i,j] ~ dcat(gamma[z[i,j-1], 1:3])
      y[i,j] ~ dcat(omega[z[i,j], 1:2])
    }
  }
})
```

Get the date of first capture.

```
first <- apply(y, 1, function(x) min(which(x != 0)))
```

Constants in a list.

```
my.constants <- list(N = nrow(y),
                     K = ncol(y),
                     first = first)
```

Data in a list.

```
my.data <- list(y = y + 1)
```

Initial values.

```
zinit <- y
for (i in 1:nrow(y)) {
  for (j in first[i]:ncol(y)) {
    if (j == first[i]) zinit[i,j] <- which(rmultinom(1, 1, c(1/2,1/2))==1) # pick alive
 state
    if (j > first[i]) zinit[i,j] <- zinit[i,j-1] # because no transition, state remains
 the same
  }
}
zinit <- as.matrix(zinit)
initial.values <- function() list(phi = runif(1,0,1),
                                  p1 = runif(1,0,1),
                                  p2 = runif(1,0,1),
                                  pi = runif(1,0,1),
                                  z = zinit)
```

Parameters to be monitored.

```
parameters.to.save <- c("phi", "p1", "p2", "pi")
```

MCMC details.

```
n.iter <- 10000
n.burnin <- 5000
n.chains <- 2
```

Run nimble.

```
mcmc.phipmix <- nimbleMCMC(code = hmm.phipmix,
                           constants = my.constants,
                           data = my.data,
                           inits = initial.values,
                           monitors = parameters.to.save,
                           niter = n.iter,
                           nburnin = n.burnin,
                           nchains = n.chains)
```

```
defining model...
```

```
building model...
```

```
setting data and initial values...
```

```
running calculate on model (any error reports that follow may simply reflect missing val
ues in model variables) ...
checking model sizes and dimensions...
checking model calculations...
model building finished.
compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilat
ion details.
compilation finished.
running chain 1...
```

```
|-------------|-------------|-------------|-------------|
|-----------------------------------------------------|
```

```
running chain 2...
```

```
|-------------|-------------|-------------|-------------|
|-----------------------------------------------------|
```

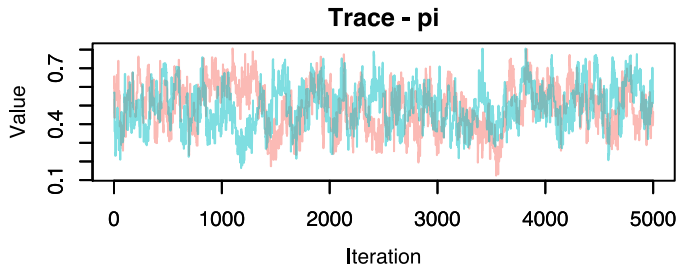
Numerical summaries.

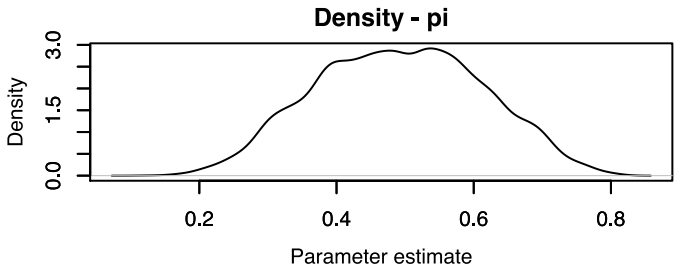
```
MCMCsummary(mcmc.phipmix, round = 2)
```

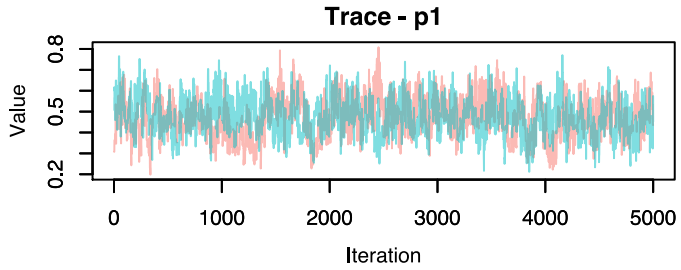
```
    mean   sd 2.5%  50% 97.5% Rhat n.eff
p1  0.47 0.09 0.30 0.47  0.65 1.01   316
p2  0.38 0.10 0.20 0.38  0.57 1.00   178
phi 0.81 0.05 0.71 0.81  0.92 1.00   303
pi  0.49 0.12 0.27 0.49  0.72 1.02   181
```

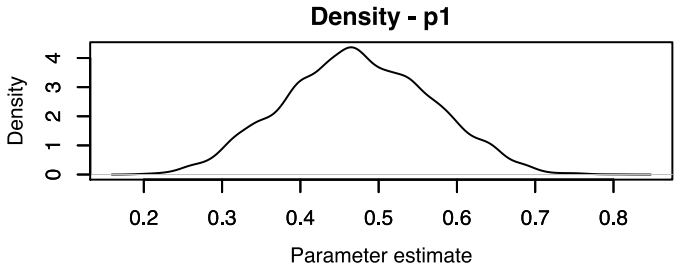
Let's have a look to the trace and posterior distribution of the proportion of individuals in each class, as well as the detection probabilities.

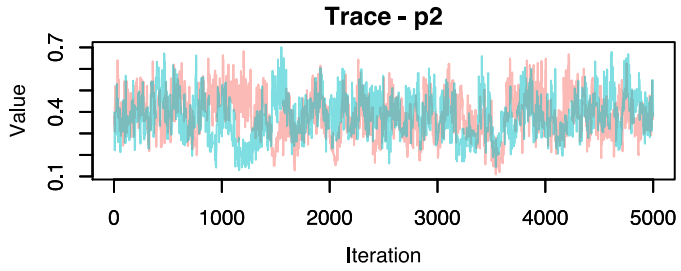
```
MCMCtrace(mcmc.phipmix, pdf = FALSE, params = c("pi", "p1", "p2"))
```

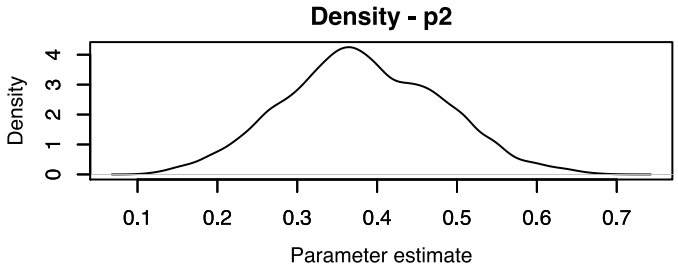






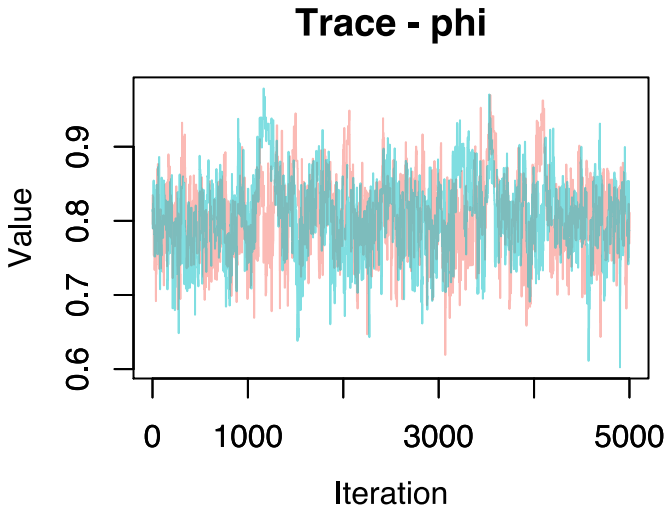


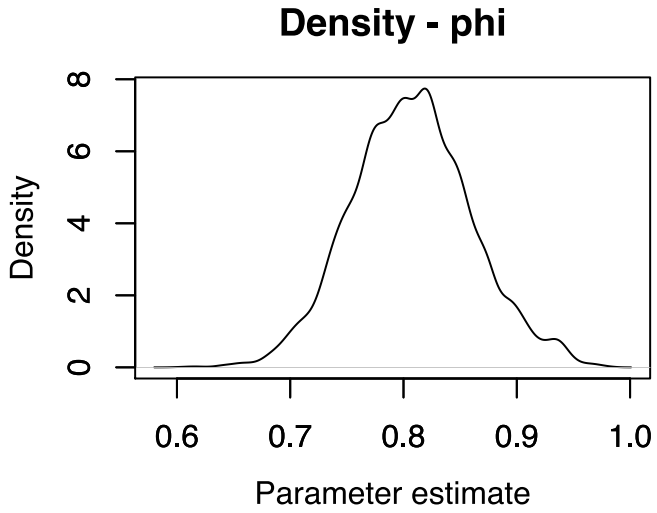




Last, survival.

```
MCMCtrace(mcmc.phipmix, pdf = FALSE, params = c("phi"))
```





Class 8 live demo: Skip your coffee break: Speed up MCMC convergence

The team

last updated: 2021-05-12

Introduction

In this demo, we illustrate how nimble can be used to speed up MCMC computations. First, we demonstrate how to change the default samplers. Then we use the nimbleEcology package (https://cran.r-project.org/web/packages/nimbleEcology/vignettes/Introduction_to_nimbleEcology.html) which implements marginalization (by integrating or sum the likelihood over latent states). Marginalization eliminates the need for MCMC sampling these latent variables, which often provide efficiency gains. We also illustrate how to use nimble functions to write a likelihood that works with a dataset in which we pool individuals with the same encounter histories.

```
library(tidyverse)
library(nimbleEcology)
```

```
Loading nimbleEcology. Registering the following distributions:
 dOcc, dDynOcc, dCJS, dHMM, dDHMM, dNmixture.
```

```
library(MCMCvis)
```

Work with samplers: Use slice sampling

We're going back to the Dipper example. Let's consider a model with wing length and individual random effect on survival.

Read in data.

```
dipper <- read_csv(here::here("slides", "dat", "dipper.csv"))
y <- dipper %>%
  select(year_1981:year_1987) %>%
  as.matrix()
```

Get occasion of first capture.

```
first <- apply(y, 1, function(x) min(which(x !=0)))
```

Constants in a list.

```
wing.length.st <- as.vector(scale(dipper$wing_length))
my.constants <- list(N = nrow(y),
                     T = ncol(y),
                     first = first,
                     winglength = wing.length.st)
```

Data in a list.

```
my.data <- list(y = y + 1)
```

Initial values.

```
zinits <- y + 1 # non-detection -> alive
zinits[zinits == 2] <- 1 # dead -> alive
initial.values <- function() list(beta = rnorm(2,0,1.5),
                                  sdeps = runif(1,0,3),
                                  p = runif(1,0,1),
                                  z = zinits)
```

MCMC details.

```
parameters.to.save <- c("beta", "sdeps", "p")
n.iter <- 10000
n.burnin <- 2500
n.chains <- 2
```

Write the model.

```
hmm.phiwlrep <- nimbleCode({
    p ~ dunif(0, 1) # prior detection
    omega[1,1] <- 1 - p    # Pr(alive t -> non-detected t)
    omega[1,2] <- p        # Pr(alive t -> detected t)
    omega[2,1] <- 1        # Pr(dead t -> non-detected t)
    omega[2,2] <- 0        # Pr(dead t -> detected t)
  for (i in 1:N){
    logit(phi[i]) <- beta[1] + beta[2] * winglength[i] + eps[i]
    eps[i] ~ dnorm(mean = 0, sd = sdeps)
    gamma[1,1,i] <- phi[i]      # Pr(alive t -> alive t+1)
    gamma[1,2,i] <- 1 - phi[i]  # Pr(alive t -> dead t+1)
    gamma[2,1,i] <- 0           # Pr(dead t -> alive t+1)
    gamma[2,2,i] <- 1           # Pr(dead t -> dead t+1)
  }
  beta[1] ~ dnorm(mean = 0, sd = 1.5)
  beta[2] ~ dnorm(mean = 0, sd = 1.5)
  sdeps ~ dunif(0, 10)
  delta[1] <- 1          # Pr(alive t = 1) = 1
  delta[2] <- 0          # Pr(dead t = 1) = 0
  # likelihood
  for (i in 1:N){
    z[i,first[i]] ~ dcat(delta[1:2])
    for (j in (first[i]+1):T){
      z[i,j] ~ dcat(gamma[z[i,j-1], 1:2, i])
      y[i,j] ~ dcat(omega[z[i,j], 1:2])
    }
  }
})
```

Run nimble.

```
mcmc.phiwlrep <- nimbleMCMC(code = hmm.phiwlrep,
                            constants = my.constants,
                            data = my.data,
                            inits = initial.values,
                            monitors = parameters.to.save,
                            niter = n.iter,
                            nburnin = n.burnin,
                            nchains = n.chains)
```

```
defining model...
```

```
building model...
```

```
setting data and initial values...
```

```
running calculate on model (any error reports that follow may simply reflect missing val
ues in model variables) ...
checking model sizes and dimensions... This model is not fully initialized. This is not
an error. To see which variables are not initialized, use model$initializeInfo(). For mo
re information on model initialization, see help(modelInitialization).
checking model calculations...
```

```
NAs were detected in model variables: eps, logProb_eps, phi, gamma, logProb_z.
```

```
model building finished.
compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilat
ion details.
compilation finished.
running chain 1...
```

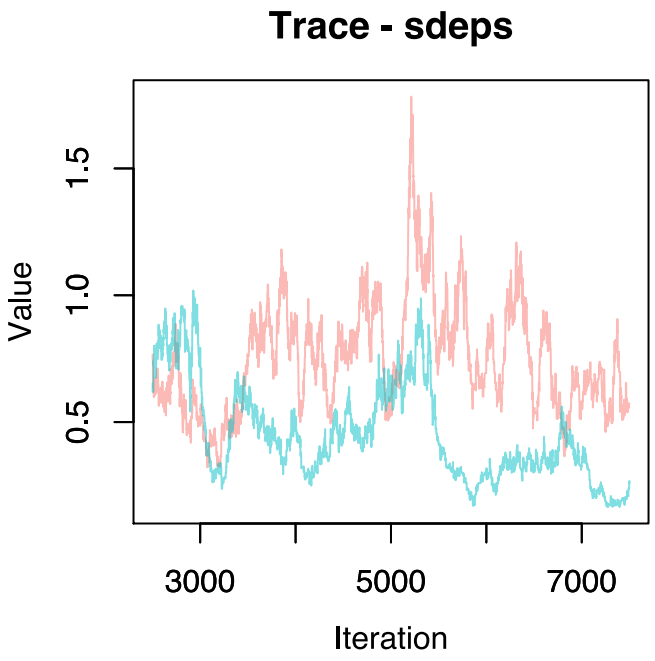
```
|-------------|-------------|-------------|-------------|
|-----------------------------------------------------|
```

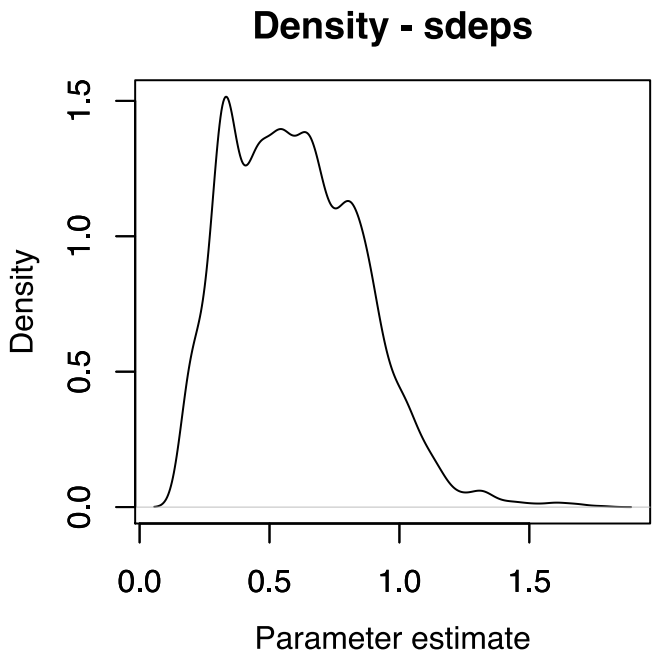
```
running chain 2...
```

```
|-------------|-------------|-------------|-------------|
|-----------------------------------------------------|
```

Let's have a look to the trace of the standard deviation of the random effect. Hmmm.

```
MCMCtrace(mcmc.phiwlrep, params = "sdeps", pdf = FALSE)
```





Let's try and change the default sampler for this parameter. What are the samplers used by default?

```
hmm.phiwlrep <- nimbleModel(code = hmm.phiwlrep,
                            constants = my.constants,
                            data = my.data,
                            inits = initial.values())
mcmcConf <- configureMCMC(hmm.phiwlrep)
```

```
===== Monitors =====
thin = 1: p, beta, sdeps, z
===== Samplers =====
RW sampler (259)
  - p
  - beta[]  (2 elements)
  - sdeps
  - eps[]  (255 elements)
posterior_predictive sampler (78)
  - eps[]  (39 elements)
  - z[]  (39 elements)
categorical sampler (1103)
  - z[]  (1103 elements)
```

We remove the default sampler, and use slice sampler instead.

```
mcmcConf$removeSamplers('sdeps')
mcmcConf$addSampler(target = 'sdeps',
                    type = "slice")
mcmcConf
```

```
===== Monitors =====
thin = 1: p, beta, sdeps, z
===== Samplers =====
slice sampler (1)
  - sdeps
RW sampler (258)
  - p
  - beta[]  (2 elements)
  - eps[]  (255 elements)
posterior_predictive sampler (78)
  - eps[]  (39 elements)
  - z[]  (39 elements)
categorical sampler (1103)
  - z[]  (1103 elements)
```

Compile model and MCMC.

```
Rmcmc <- buildMCMC(mcmcConf)
Cmodel <- compileNimble(hmm.phiwlrep)
```

```
compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilat
ion details.
```

```
compilation finished.
```

```
Cmcmc <- compileNimble(Rmcmc, project = hmm.phiwlrep)
```

```
compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilat
ion details.
compilation finished.
```

Run nimble.

```
Cmcmc$run(10000)
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

```
NULL
```

```
samples1 <- as.matrix(Cmcmc$mvSamples)
Cmcmc$run(10000)
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

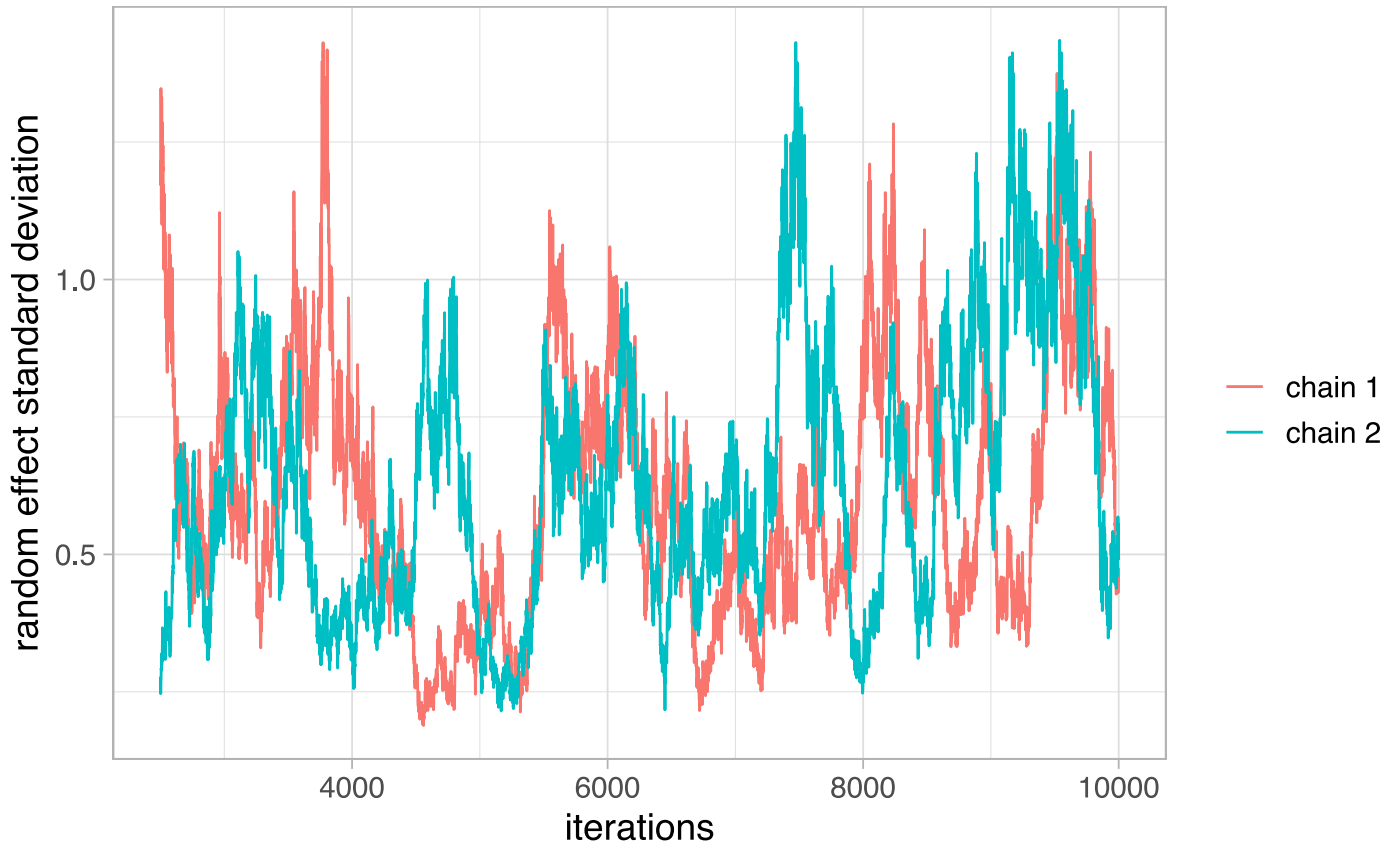
```
NULL
```

```
samples2 <- as.matrix(Cmcmc$mvSamples)
```

Format results in data.frames.

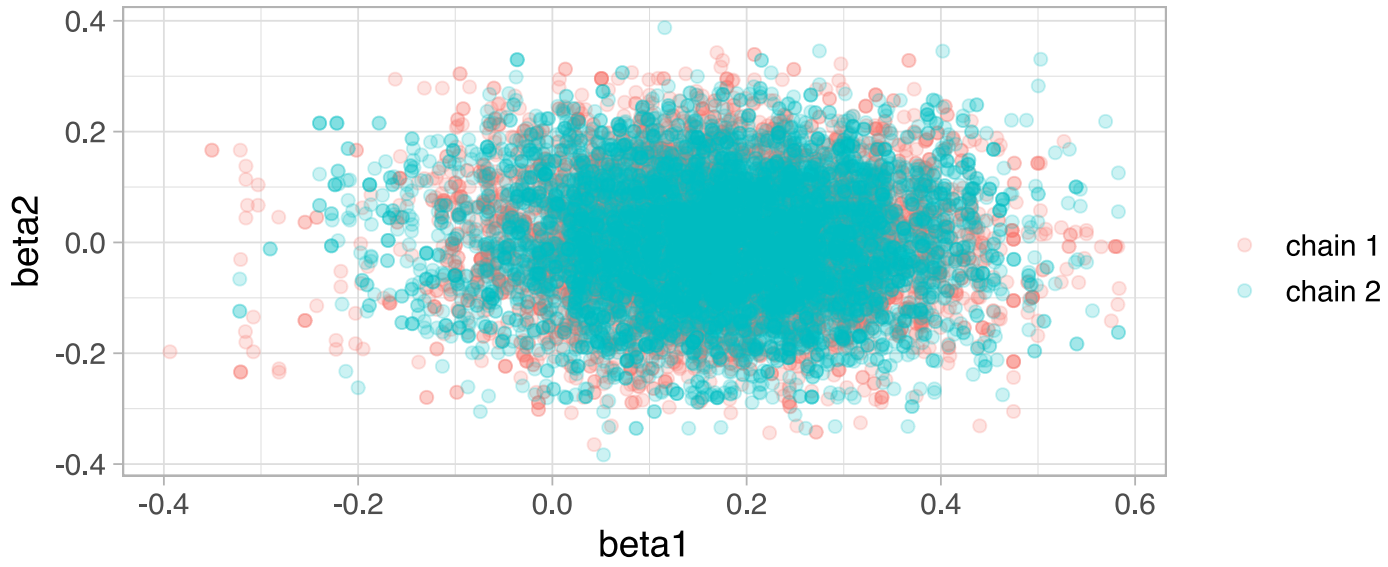
```
df.sdeps <- data.frame(iter = c(2501:10000, 2501:10000),
                  samples = c(samples1[2501:10000,"sdeps"], samples2[2501:10000,"sdeps"
]),
                  chain = c(rep("chain 1", length(samples1[2501:10000,"sdeps"])),
                            rep("chain 2", length(samples2[2501:10000,"sdeps"]))))
df.beta <- data.frame(iter = c(2501:10000, 2501:10000),
                  beta1 = c(samples1[2501:10000,"beta[1]"], samples2[2501:10000,"beta[1]"
]),
                  beta2 = c(samples1[2501:10000,"beta[2]"], samples2[2501:10000,"beta[2]"
]),
                  chain = c(rep("chain 1", length(samples1[2501:10000,"sdeps"])),
                            rep("chain 2", length(samples2[2501:10000,"sdeps"]))))
```

Trace plot for the standard deviation of the random effect.



Work with samplers: Use block sampling

High correlation in (regression) parameters may make independent samplers inefficient. Let's have a look to the correlation between the intercept and the slope of the relationship between survival and wing length in the European dipper.



There seems to be no correlation. Let's act for a minute as if we had a strong correlation. We're gonna see how block sampling might help. In block sampling, we propose candidate values from a multivariate distribution.

First, we remove and replace independent RW samples by block sampling. Then we proceed as usual.

```
mcmcConf$removeSamplers(c('beta[1]','beta[2]'))
mcmcConf$addSampler(target = c('beta[1]','beta[2]'),
                    type = "RW_block")
```

```
Note: Assigning an RW_block sampler to nodes with very different scales can result in lo
w MCMC efficiency.  If all nodes assigned to RW_block are not on a similar scale, we rec
ommend providing an informed value for the "propCov" control list argument, or using the
AFSS sampler instead.
```

```
mcmcConf
```

```
===== Monitors =====
thin = 1: p, beta, sdeps, z
===== Samplers =====
slice sampler (1)
  - sdeps
RW_block sampler (1)
  - beta[1], beta[2]
RW sampler (256)
  - p
  - eps[]  (255 elements)
posterior_predictive sampler (78)
  - eps[]  (39 elements)
  - z[]  (39 elements)
categorical sampler (1103)
  - z[]  (1103 elements)
```

```
Rmcmc <- buildMCMC(mcmcConf)
Cmodel <- compileNimble(hmm.phiwlrep)
```

```
compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilat
ion details.
```

```
compilation finished.
```

```
Cmcmc <- compileNimble(Rmcmc, project = hmm.phiwlrep)
```

```
compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilat
ion details.
compilation finished.
```

Run nimble and a single chain.

```
Cmcmc$run(10000)
```

```
|-------------|-------------|-------------|-------------|
|-------------------------------------------------------|
```

```
NULL
```

```
samples <- as.matrix(Cmcmc$mvSamples)
```

Summarize.

```
samples %>%
  as_tibble() %>%
  select(!starts_with("z")) %>% # ignore the latent states z
  summarise(across(everything(), list(mean = mean, sd = sd)))
```

```
# A tibble: 1 x 8
  `beta[1]_mean` `beta[1]_sd` `beta[2]_mean` `beta[2]_sd` p_mean   p_sd
           <dbl>        <dbl>          <dbl>        <dbl>  <dbl>  <dbl>
1          0.204        0.132       -0.00842        0.104  0.896 0.0370
# … with 2 more variables: sdeps_mean <dbl>, sdeps_sd <dbl>
```

Marginalization with nimbleEcology

Let's get back to the analysis of the Canada geese data, with 3 sites.

```
geese <- read_csv("geese.csv", col_names = TRUE)
```

```
── Column specification ────────────────────────────────────────────────────────
cols(
  year_1984 = col_double(),
  year_1985 = col_double(),
  year_1986 = col_double(),
  year_1987 = col_double(),
  year_1988 = col_double(),
  year_1989 = col_double()
)
```

```
y <- as.matrix(geese)
```

Get the occasion of first capture for each individual.

```
get.first <- function(x) min(which(x != 0))
first <- apply(y, 1, get.first)
```

We filter out individuals that are first captured at last occasion. These individuals do not contribute to parameter estimation, and also they cause problems with nimbleEcology.

```
mask <- which(first!=ncol(y)) # individuals that are not first encountered at last occas
ion
y <- y[mask, ]                # keep only these
first <- first[mask]
```

Let's write the model. Note that the likelihood is simpler due to the use of the function `dHMM`.

```
multisite.marginalized <- nimbleCode({

  # ------------------------------------------------
  # Parameters:
  # phiA: survival probability site A
  # phiB: survival probability site B
  # phiC: survival probability site B
  # psiAA = psiA[1]: movement probability from site A to site A (reference)
  # psiAB = psiA[2]: movement probability from site A to site B
  # psiAC = psiA[3]: movement probability from site A to site C
  # psiBA = psiB[1]: movement probability from site B to site A
  # psiBB = psiB[2]: movement probability from site B to site B (reference)
  # psiBC = psiB[3]: movement probability from site B to site C
  # psiCA = psiC[1]: movement probability from site C to site A
  # psiCB = psiC[2]: movement probability from site C to site B
  # psiCC = psiC[3]: movement probability from site C to site C (reference)
  # pA: recapture probability site A
  # pB: recapture probability site B
  # pC: recapture probability site C
  # ------------------------------------------------
  # States (z):
  # 1 alive at A
  # 2 alive at B
  # 2 alive at C
  # 3 dead
  # Observations (y):
  # 1 not seen
  # 2 seen at A
  # 3 seen at B
  # 3 seen at C
  # ------------------------------------------------

  # survival priors
  phiA ~ dunif(0, 1)
  phiB ~ dunif(0, 1)
  phiC ~ dunif(0, 1)
  # priors for detection
  pA ~ dunif(0, 1)
  pB ~ dunif(0, 1)
  pC ~ dunif(0, 1)
  # priors for transitions: Dirichlet
  psiA[1:3] ~ ddirch(alpha[1:3])
  psiB[1:3] ~ ddirch(alpha[1:3])
  psiC[1:3] ~ ddirch(alpha[1:3])
  # probabilities of state z(t+1) given z(t)
  gamma[1,1] <- phiA * psiA[1]
  gamma[1,2] <- phiA * psiA[2]
  gamma[1,3] <- phiA * psiA[3]
  gamma[1,4] <- 1 - phiA
  gamma[2,1] <- phiB * psiB[1]
  gamma[2,2] <- phiB * psiB[2]
  gamma[2,3] <- phiB * psiB[3]
  gamma[2,4] <- 1 - phiB
```

```
  gamma[3,1] <- phiC * psiC[1]
  gamma[3,2] <- phiC * psiC[2]
  gamma[3,3] <- phiC * psiC[3]
  gamma[3,4] <- 1 - phiC
  gamma[4,1] <- 0
  gamma[4,2] <- 0
  gamma[4,3] <- 0
  gamma[4,4] <- 1

  # probabilities of y(t) given z(t)
  omega[1,1] <- 1 - pA     # Pr(alive A t -> non-detected t)
  omega[1,2] <- pA         # Pr(alive A t -> detected A t)
  omega[1,3] <- 0          # Pr(alive A t -> detected B t)
  omega[1,4] <- 0          # Pr(alive A t -> detected C t)
  omega[2,1] <- 1 - pB     # Pr(alive B t -> non-detected t)
  omega[2,2] <- 0          # Pr(alive B t -> detected A t)
  omega[2,3] <- pB         # Pr(alive B t -> detected B t)
  omega[2,4] <- 0          # Pr(alive B t -> detected C t)
  omega[3,1] <- 1 - pC     # Pr(alive C t -> non-detected t)
  omega[3,2] <- 0          # Pr(alive C t -> detected A t)
  omega[3,3] <- 0          # Pr(alive C t -> detected B t)
  omega[3,4] <- pC         # Pr(alive C t -> detected C t)
  omega[4,1] <- 1          # Pr(dead t -> non-detected t)
  omega[4,2] <- 0          # Pr(dead t -> detected A t)
  omega[4,3] <- 0          # Pr(dead t -> detected B t)
  omega[4,4] <- 0          # Pr(dead t -> detected C t)
  # initial state probs
  for(i in 1:N) {
    init[i, 1:4] <- gamma[ y[i, first[i] ] - 1, 1:4 ] # First state propagation
  }

  # likelihood
  for (i in 1:N){
      y[i,(first[i]+1):K] ~ dHMM(init = init[i,1:4],  # count data from first[i] + 1
                                 probObs = omega[1:4,1:4],     # observation matrix
                                 probTrans = gamma[1:4,1:4],   # transition matrix
                                 len = K - first[i],           # nb of occasions
                                 checkRowSums = 0)             # do not check whether el
ements in a row sum tp 1
      }
})
```

Data in a list.

```
my.data <- list(y = y + 1,
                alpha = c(1, 1, 1))
```

Constants in a list.

```
my.constants <- list(first = first,
                     K = ncol(y),
                     N = nrow(y))
```

Initial values. Note that we do not need initial values for the latent states anymore.

```
initial.values <- function(){list(phiA = runif(1, 0, 1),
                                  phiB = runif(1, 0, 1),
                                  phiC = runif(1, 0, 1),
                                  psiA = rdirch(1, c(1,1,1)),
                                  psiB = rdirch(1, c(1,1,1)),
                                  psiC = rdirch(1, c(1,1,1)),
                                  pA = runif(1, 0, 1),
                                  pB = runif(1, 0, 1),
                                  pC = runif(1, 0, 1))}
```

Parameters to monitor.

```
parameters.to.save <- c("phiA", "phiB", "phiC", "psiA", "psiB", "psiC","pA", "pB", "pC")
```

MCMC settings.

```
n.iter <- 10000
n.burnin <- 5000
n.chains <- 2
```

Run nimble.

```
system.time(multisite.marginalized.out <- nimbleMCMC(code = multisite.marginalized,
                                                     constants = my.constants,
                                                     data = my.data,
                                                     inits = initial.values(),
                                                     monitors = parameters.to.save,
                                                     niter = n.iter,
                                                     nburnin = n.burnin,
                                                     nchains = n.chains))
```

The run took around 2 minutes. For comparison, the standard formulation (see live demo for class 6 on transition estimation) took around 4 minutes.

Explore outputs.

```
MCMCsummary(multisite.marginalized.out, round = 2)
```

```
         mean   sd 2.5%  50% 97.5% Rhat n.eff
pA       0.51 0.08 0.36 0.51  0.69 1.00   771
pB       0.46 0.05 0.37 0.45  0.56 1.00   741
pC       0.24 0.06 0.14 0.23  0.38 1.00   542
phiA     0.61 0.05 0.51 0.61  0.71 1.00  1304
phiB     0.70 0.04 0.63 0.70  0.77 1.00  1002
phiC     0.77 0.07 0.64 0.77  0.91 1.00   865
psiA[1] 0.74 0.05 0.63 0.74  0.84 1.00  1564
psiA[2] 0.24 0.05 0.14 0.23  0.35 1.00  1460
psiA[3] 0.02 0.02 0.00 0.02  0.08 1.00  2372
psiB[1] 0.07 0.02 0.04 0.07  0.12 1.00  1250
psiB[2] 0.83 0.04 0.73 0.84  0.90 1.01   849
psiB[3] 0.10 0.04 0.04 0.09  0.19 1.02   890
psiC[1] 0.02 0.02 0.00 0.02  0.06 1.01  2321
psiC[2] 0.21 0.05 0.12 0.21  0.33 1.00  1495
psiC[3] 0.77 0.06 0.64 0.77  0.86 1.00  1359
```

Marginalization and weighted likelihood with nimbleEcology

In this section, we're gonna use nimble functions to express the likelihood using pooled encounter histories. We use a vector `mult` that contains the number of individuals with a particular encounter history. We hacked the `dHMM` nimbleEcology function below.

```
dHMMweighted <- nimbleFunction(
  run = function (x = double(1),
                  init = double(1),
                  probObs = double(2),
                  probTrans = double(2),
                  len = double(0),
                  mult = double(0), # NEWLY ADDED: argument stating number of occurrence
s
                                    # of same encounter history in entire dataset
                  checkRowSums = integer(0, default = 0),
                  log = integer(0, default = 0))
  {
    if (length(x) != len)
      nimStop("In dHMM: Argument len must be length of x or 0.")
    if (nimDim(probObs)[1] != nimDim(probTrans)[1])
      nimStop("In dHMM: Length of dimension 1 in probObs must equal length of dimension
 1 in probTrans.")
    if (nimDim(probTrans)[1] != nimDim(probTrans)[2])
      nimStop("In dHMM: probTrans must be a square matrix.")
    ## There was a strict test for sum(init) == 1.  This could be true in R and false in
C++!
    if (abs(sum(init) - 1) > 1e-06)
      nimStop("In dHMM: Initial probabilities must sum to 1.")
    if (checkRowSums) {
      transCheckPasses <- TRUE
      for (i in 1:nimDim(probTrans)[1]) {
        thisCheckSum <- sum(probTrans[i, ])
        if (abs(thisCheckSum - 1) > 1e-06) {
          nimPrint("In dHMM: Problem with sum(probTrans[i,]) with i = ",
                   i, ". The sum should be 1 but is ", thisCheckSum)
          transCheckPasses <- FALSE
        }
      }
      obsCheckPasses <- TRUE
      for (i in 1:nimDim(probObs)[1]) {
        thisCheckSum <- sum(probObs[i, ])
        if (abs(thisCheckSum - 1) > 1e-06) {
          nimPrint("In dHMM: Problem with sum(probObs[i,]) with i = ",
                   i, ". The sum should be 1 but is ", thisCheckSum)
          obsCheckPasses <- FALSE
        }
      }
      if (!(transCheckPasses | obsCheckPasses))
        nimStop("In dHMM: probTrans and probObs were not specified correctly.  Probabili
ties in each row (second dimension) must sum to 1.")
      if (!transCheckPasses)
        nimStop("In dHMM: probTrans was not specified correctly.  Probabilities in each
 row (second dimension) must sum to 1.")
      if (!obsCheckPasses)
        nimStop("In dHMM: probObs was not specified correctly. Probabilities in each row
must sum to 1.")
    }
    pi <- init
```

```
    logL <- 0
    nObsClasses <- nimDim(probObs)[2]
    for (t in 1:len) {
      if (x[t] > nObsClasses | x[t] < 1)
        nimStop("In dHMM: Invalid value of x[t].")
      Zpi <- probObs[, x[t]] * pi
      sumZpi <- sum(Zpi)
      logL <- logL + log(sumZpi) * mult # NEWLY ADDED
      if (t != len)
        pi <- ((Zpi %*% probTrans)/sumZpi)[1, ]
    }
    if (log)
      return(logL)
    return(exp(logL))
    returnType(double())
  })
```

Write the model.

```
multisite.marginalized <- nimbleCode({

  # ------------------------------------------------
  # Parameters:
  # phiA: survival probability site A
  # phiB: survival probability site B
  # phiC: survival probability site B
  # psiAA = psiA[1]: movement probability from site A to site A (reference)
  # psiAB = psiA[2]: movement probability from site A to site B
  # psiAC = psiA[3]: movement probability from site A to site C
  # psiBA = psiB[1]: movement probability from site B to site A
  # psiBB = psiB[2]: movement probability from site B to site B (reference)
  # psiBC = psiB[3]: movement probability from site B to site C
  # psiCA = psiC[1]: movement probability from site C to site A
  # psiCB = psiC[2]: movement probability from site C to site B
  # psiCC = psiC[3]: movement probability from site C to site C (reference)
  # pA: recapture probability site A
  # pB: recapture probability site B
  # pC: recapture probability site C
  # ------------------------------------------------
  # States (z):
  # 1 alive at A
  # 2 alive at B
  # 2 alive at C
  # 3 dead
  # Observations (y):
  # 1 not seen
  # 2 seen at A
  # 3 seen at B
  # 3 seen at C
  # ------------------------------------------------

  # survival priors
  phiA ~ dunif(0, 1)
  phiB ~ dunif(0, 1)
  phiC ~ dunif(0, 1)
  # detection priors
  pA ~ dunif(0, 1)
  pB ~ dunif(0, 1)
  pC ~ dunif(0, 1)
  # transition priors: Dirichlet
  psiA[1:3] ~ ddirch(alpha[1:3])
  psiB[1:3] ~ ddirch(alpha[1:3])
  psiC[1:3] ~ ddirch(alpha[1:3])
  gamma[1,1] <- phiA * psiA[1]
  gamma[1,2] <- phiA * psiA[2]
  gamma[1,3] <- phiA * psiA[3]
  gamma[1,4] <- 1 - phiA
  gamma[2,1] <- phiB * psiB[1]
  gamma[2,2] <- phiB * psiB[2]
  gamma[2,3] <- phiB * psiB[3]
  gamma[2,4] <- 1 - phiB
  gamma[3,1] <- phiC * psiC[1]
```

```
  gamma[3,2] <- phiC * psiC[2]
  gamma[3,3] <- phiC * psiC[3]
  gamma[3,4] <- 1 - phiC
  gamma[4,1] <- 0
  gamma[4,2] <- 0
  gamma[4,3] <- 0
  gamma[4,4] <- 1

  # probabilities of y(t) given z(t)
  omega[1,1] <- 1 - pA     # Pr(alive A t -> non-detected t)
  omega[1,2] <- pA         # Pr(alive A t -> detected A t)
  omega[1,3] <- 0          # Pr(alive A t -> detected B t)
  omega[1,4] <- 0          # Pr(alive A t -> detected C t)
  omega[2,1] <- 1 - pB     # Pr(alive B t -> non-detected t)
  omega[2,2] <- 0          # Pr(alive B t -> detected A t)
  omega[2,3] <- pB         # Pr(alive B t -> detected B t)
  omega[2,4] <- 0          # Pr(alive B t -> detected C t)
  omega[3,1] <- 1 - pC     # Pr(alive C t -> non-detected t)
  omega[3,2] <- 0          # Pr(alive C t -> detected A t)
  omega[3,3] <- 0          # Pr(alive C t -> detected B t)
  omega[3,4] <- pC         # Pr(alive C t -> detected C t)
  omega[4,1] <- 1          # Pr(dead t -> non-detected t)
  omega[4,2] <- 0          # Pr(dead t -> detected A t)
  omega[4,3] <- 0          # Pr(dead t -> detected B t)
  omega[4,4] <- 0          # Pr(dead t -> detected C t)

  for(i in 1:N) {
    init[i, 1:4] <- gamma[ y[i, first[i] ] - 1, 1:4 ] # First state propagation
    }

  # likelihood
  for (i in 1:N){
    y[i,(first[i]+1):K] ~ dHMMweighted(init = init[i,1:4], # count data from first[i] +
 1
                                       mult = mult[i],
                                       probObs = omega[1:4,1:4],
                                       probTrans = gamma[1:4,1:4],
                                       len = K - first[i],
                                       checkRowSums = 0)
  }
})
```

We need to pool the individual encounter histories by unique encounter histories, and to record the number of individuals with a particular unique encounter history.

```
geese <- read_csv("geese.csv", col_names = TRUE)
```

```
── Column specification ────────────────────────────────────────────────────────
cols(
  year_1984 = col_double(),
  year_1985 = col_double(),
  year_1986 = col_double(),
  year_1987 = col_double(),
  year_1988 = col_double(),
  year_1989 = col_double()
)
```

```
y <- as.matrix(geese)
y_weighted <- y %>%
  as_tibble() %>%
  group_by_all() %>%
  summarise(mult = n()) %>%
  relocate(mult) %>%
  as.matrix()
```

```
`summarise()` has grouped output by 'year_1984', 'year_1985', 'year_1986', 'year_1987',
'year_1988'. You can override using the `.groups` argument.
```

```
head(y_weighted)
```

```
     mult year_1984 year_1985 year_1986 year_1987 year_1988 year_1989
[1,]    5         0         0         0         0         0         1
[2,]    8         0         0         0         0         0         2
[3,]    1         0         0         0         0         0         3
[4,]    6         0         0         0         0         1         0
[5,]    2         0         0         0         0         1         1
[6,]   27         0         0         0         0         2         0
```

```
mult <- y_weighted[,1] # nb of individuals w/ a particular encounter history
y <- y_weighted[,-1] # pooled data
```

There are 5 individuals that were detected only once, in site A in year 1989, 8 individuals that were detected only once, in 1989 in site B, and so on.

Get the occasion of first capture for each history.

```
get.first <- function(x) min(which(x != 0))
first <- apply(y, 1, get.first)
```

Filter out individuals that are first captured at last occasion.

```
mask <- which(first!=ncol(y))
y <- y[mask, ]
```

Apply filter on occasion of first capture and sample size.

```
first <- first[mask]
mult <- mult[mask]
```

Data in a list.

```
my.data <- list(y = y + 1,
                alpha = c(1, 1, 1))
```

Constants in a list.

```
my.constants <- list(first = first,
                     K = ncol(y),
                     N = nrow(y),
                     mult = mult)
```

Initial values. Note that we do not need initial values for the latent states anymore.

```
initial.values <- function(){list(phiA = runif(1, 0, 1),
                                  phiB = runif(1, 0, 1),
                                  phiC = runif(1, 0, 1),
                                  psiA = rdirch(1, c(1,1,1)),
                                  psiB = rdirch(1, c(1,1,1)),
                                  psiC = rdirch(1, c(1,1,1)),
                                  pA = runif(1, 0, 1),
                                  pB = runif(1, 0, 1),
                                  pC = runif(1, 0, 1))}
```

Parameters to monitor.

```
parameters.to.save <- c("phiA", "phiB", "phiC", "psiA", "psiB", "psiC","pA", "pB", "pC")
```

MCMC settings.

```
n.iter <- 10000
n.burnin <- 5000
n.chains <- 2
```

Run nimble.

```
system.time(multisite.marginalized.out <- nimbleMCMC(code = multisite.marginalized,
                                                     constants = my.constants,
                                                     data = my.data,
                                                     inits = initial.values(),
                                                     monitors = parameters.to.save,
                                                     niter = n.iter,
                                                     nburnin = n.burnin,
                                                     nchains = n.chains))
```

This run won't work cause we need a `rHMMweighted` as well.

```
rHMMweighted <- nimbleFunction(
  run = function(n = integer(),    ## Observed capture (state) history
                 init = double(1),
                 probObs = double(2),
                 probTrans = double(2),
                 len = double(0, default = 0),
                 mult = double(0),
                 checkRowSums = double(0, default = 1)) {
    returnType(double(1))
    if (dim(probObs)[1] != dim(probTrans)[1]) stop("In rHMM: Number of cols in probObs m
ust equal number of cols in probTrans.")
    if (dim(probTrans)[1] != dim(probTrans)[2]) stop("In rHMM: probTrans must be a squar
e matrix.")
    if (abs(sum(init) - 1) > 1e-06) stop("In rHMM: Initial probabilities must sum to 1."
)
    if (checkRowSums) {
      transCheckPasses <- TRUE
      for (i in 1:dim(probTrans)[1]) {
        thisCheckSum <- sum(probTrans[i,])
        if (abs(thisCheckSum - 1) > 1e-6) {
          ## Compilation doesn't support more than a simple string for stop()
          ## so we provide more detail using a print().
          print("In rHMM: Problem with sum(probTrans[i,]) with i = ", i, ". The sum shou
ld be 1 but is ", thisCheckSum)
          transCheckPasses <- FALSE
        }
      }
      obsCheckPasses <- TRUE
      for (i in 1:dim(probObs)[1]) {
        thisCheckSum <- sum(probObs[i,])
        if (abs(thisCheckSum - 1) > 1e-6) {
          print("In rHMM: Problem with sum(probObs[i,]) with i = ", i, ". The sum should
be 1 but is ", thisCheckSum)
          obsCheckPasses <- FALSE
        }
      }
      if(!(transCheckPasses | obsCheckPasses))
        stop("In rHMM: probTrans and probObs were not specified correctly.  Probabilitie
s in each row (second dimension) must sum to 1.")
      if(!transCheckPasses)
        stop("In rHMM: probTrans was not specified correctly.  Probabilities in each row
(second dimension) must sum to 1.")
      if(!obsCheckPasses)
        stop("In rHMM: probObs was not specified correctly. Probabilities in each row mu
st sum to 1.")
    }
    ans <- numeric(len)
    probInit <- init
    trueInit <- 0
    r <- runif(1, 0, 1)
    j <- 1
    while (r > sum(probInit[1:j])) j <- j + 1
    trueInit <- j
```

```
    trueState <- trueInit
    for (i in 1:len) {
      # Transition to a new true state
      r <- runif(1, 0, 1)
      j <- 1
      while (r > sum(probTrans[trueState, 1:j])) j <- j + 1
      trueState <- j
      # Detect based on the true state
      r <- runif(1, 0, 1)
      j <- 1
      while (r > sum(probObs[trueState, 1:j])) j <- j + 1
      ans[i] <- j
    }
    return(ans)
  })
```

Run nimble again.

```
system.time(multisite.marginalized.out <- nimbleMCMC(code = multisite.marginalized,
                                                     constants = my.constants,
                                                     data = my.data,
                                                     inits = initial.values(),
                                                     monitors = parameters.to.save,
                                                     niter = n.iter,
                                                     nburnin = n.burnin,
                                                     nchains = n.chains))
```

Wow, this run took 1.5 minute, even faster than the marginalized version of the likelihood. The outputs are similar.

```
MCMCsummary(multisite.marginalized.out, round = 2)
```

```
        mean   sd 2.5%  50% 97.5% Rhat n.eff
pA      0.51 0.08 0.36 0.51  0.68 1.00   894
pB      0.45 0.05 0.36 0.45  0.55 1.02   844
pC      0.24 0.06 0.14 0.24  0.37 1.01   654
phiA    0.60 0.05 0.51 0.60  0.71 1.00  1086
phiB    0.70 0.04 0.63 0.70  0.77 1.01  1348
phiC    0.77 0.07 0.64 0.77  0.90 1.01   782
psiA[1] 0.74 0.06 0.62 0.75  0.84 1.01  1727
psiA[2] 0.24 0.05 0.14 0.23  0.35 1.01  1674
psiA[3] 0.02 0.02 0.00 0.02  0.08 1.00  2078
psiB[1] 0.07 0.02 0.04 0.07  0.12 1.00  1665
psiB[2] 0.84 0.04 0.75 0.84  0.90 1.01  1055
psiB[3] 0.09 0.03 0.04 0.09  0.17 1.01  1079
psiC[1] 0.02 0.01 0.00 0.02  0.06 1.00  2545
psiC[2] 0.21 0.05 0.12 0.21  0.33 1.01  1745
psiC[3] 0.77 0.05 0.65 0.77  0.86 1.01  1649
```

Analysis of the whole Canada geese dataset

So far we've worked with a subset of the original dataset. Fitting the same model to the whole data would be difficult, if not impossible. Let's try it with the weighted likelihood. We first read in the data.

```
geese <- read_csv2("allgeese.csv", col_names = TRUE)
```

```
ℹ Using ',' as decimal and '.' as grouping mark. Use `read_delim()` for more control.
```

```
Warning: Duplicated column names deduplicated: '0' => '0_1' [2], '0' =>
'0_2' [3], '0' => '0_3' [4], '0' => '0_4' [5]
```

```
── Column specification ────────────────────────────────────────────────────────
cols(
  `0` = col_double(),
  `0_1` = col_double(),
  `0_2` = col_double(),
  `0_3` = col_double(),
  `0_4` = col_double(),
  `1` = col_double(),
  `158` = col_double()
)
```

```
geese <- as.matrix(geese)
y <- geese[,-7]
mult <- geese[,7]
```

In total, we have 2.1277^{4} banded geese, and only 622 unique capture histories.

Get the occasion of first capture for each individual

```
get.first <- function(x) min(which(x != 0))
first <- apply(y, 1, get.first)
```

Filter out individuals that are first captured at last occasion

```
mask <- which(first!=ncol(y))
y <- y[mask, ]
```

Recalculate occasion of first capture and sample size

```
first <- first[mask]
mult <- mult[mask]
```

Data in list.

```
my.data <- list(y = y + 1,
                alpha = c(1, 1, 1))
```

Constant in a list.

```
my.constants <- list(first = first,
                     K = ncol(y),
                     N = nrow(y),
                     mult = mult)
```

Initial values.

```
initial.values <- function(){list(phiA = runif(1, 0, 1),
                                  phiB = runif(1, 0, 1),
                                  phiC = runif(1, 0, 1),
                                  psiA = rdirch(1, c(1,1,1)),
                                  psiB = rdirch(1, c(1,1,1)),
                                  psiC = rdirch(1, c(1,1,1)),
                                  pA = runif(1, 0, 1),
                                  pB = runif(1, 0, 1),
                                  pC = runif(1, 0, 1))}
```

MCMC settings.

```
parameters.to.save <- c("phiA", "phiB", "phiC", "psiA", "psiB", "psiC","pA", "pB", "pC")
n.iter <- 10000
n.burnin <- 5000
n.chains <- 2
```

Run nimble.

```
system.time(multisite.marginalized.out <- nimbleMCMC(code = multisite.marginalized,
                                                     constants = my.constants,
                                                     data = my.data,
                                                     inits = initial.values(),
                                                     monitors = parameters.to.save,
                                                     niter = n.iter,
                                                     nburnin = n.burnin,
                                                     nchains = n.chains))
```

It would take ages to run the same model on the > 20,000 individual encounter histories. Not even sure it can be run.

Let's inspect the results.

```
MCMCsummary(multisite.marginalized.out, round = 2)
```

```
        mean   sd 2.5%  50% 97.5% Rhat n.eff
pA      0.47 0.01 0.45 0.47  0.50 1.01   766
pB      0.41 0.01 0.39 0.41  0.42 1.00   809
pC      0.34 0.01 0.31 0.34  0.37 1.00   800
phiA    0.65 0.01 0.64 0.65  0.67 1.00  1216
phiB    0.68 0.01 0.67 0.68  0.70 1.00   985
phiC    0.67 0.01 0.65 0.67  0.69 1.00  1041
psiA[1] 0.73 0.01 0.72 0.73  0.75 1.00  1516
psiA[2] 0.26 0.01 0.24 0.26  0.28 1.00  1540
psiA[3] 0.01 0.00 0.00 0.01  0.01 1.00  2150
psiB[1] 0.11 0.00 0.10 0.11  0.12 1.00  1240
psiB[2] 0.87 0.00 0.86 0.87  0.88 1.00  1261
psiB[3] 0.03 0.00 0.02 0.03  0.03 1.00  1793
psiC[1] 0.05 0.00 0.04 0.05  0.06 1.00  2188
psiC[2] 0.26 0.01 0.24 0.26  0.28 1.00  1446
psiC[3] 0.70 0.01 0.67 0.70  0.72 1.00  1471
```