Appendix B: Individual heterogeneity in capture-recapture models - Bayesian approach using Jags

Introduction

In this appendix, we introduce three methods to cope with individual heterogeneity in capture-recapture models, which we implement in a Bayesian framework using MCMC methods. First, we present multistate models in which heterogeneity is measured on individuals using states. Then, we illustrate models with individual random effects and finite mixtures that can help in dealing with hidden heterogeneity. We refer to the paper for a formal presentation of these models and a list of references using them. Throughout this appendix, we use R to simulate data and program Jags is called from R using package Rjags to fit models. We do our best to ensure reproducibility. Note that the frequentist approach can be used instead, and implemented either with program E-SURGE (appendix C) or program Mark (appendix A).

Multistate models

In this section, we aim at illustrating how not accounting for individual heterogeneity may obscure the detection of life-history tradeoffs. In details, we consider two states for the individuals of our fake population, non-breeding (NB) and breeding (B). To mimic individual heterogeneity, we simulate a bunch of good individuals with survival $\phi_{NB} = 0.7$ and $\phi_B = 0.8$ and a bunch of bad individuals with survival $\phi_{NB} = 0.7$ and $\phi_B = 0.6$. Overall, the cost of breeding on survival should be detected only in bad individuals after accounting for individual heterogeneity through quality. For each group of bad vs. good individuals, we consider the same detection probability $p = 0.9$, the same transition probabilities between breeding states $\psi_{NB,B} = 0.8$ and $\psi_{B,NB} = 0.3$, and 100 newly marked individuals for each group in each year of the 6-year study.

Data simulation

Using R code from Kéry and Schaub (2012) book (chapter 9), we first define a function to simulate multistate capture-recapture data:

```r
# Define function to simulate multistate capture-recapture data
simul.ms <- function(PSI.STATE, PSI.OBS, marked, unobservable = NA){
  # Unobservable: number of state that is unobservable
  n.occasions <- dim(PSI.STATE)[4] + 1
  CH <- CH.TRUE <- matrix(NA, ncol = n.occasions, nrow = sum(marked))
  # Define a vector with the occasion of marking
  mark.occ <- matrix(0, ncol = dim(PSI.STATE)[1], nrow = sum(marked))
  g <- colSums(marked)
  for (s in 1:dim(PSI.STATE)[1]){
    if (g[s]==0) next # To avoid error message if nothing to replace
    mark.occ[(cumsum(g[1:s])-g[s]+1)[s]:cumsum(g[1:s])[s],s] <-
    rep(1:n.occasions, marked[1:n.occasions,s])
  } #s
  for (i in 1:sum(marked)){
    for (s in 1:dim(PSI.STATE)[1]){
      if (mark.occ[i,s]==0) next
      first <- mark.occ[i,s]
      CH[i,first] <- s
      CH.TRUE[i,first] <- s
    } #s
    for (t in (first+1):n.occasions){
      # Multinomial trials for state transitions
      if (first==n.occasions) next
      state <- which(rmultinom(1, 1, PSI.STATE[CH.TRUE[i,t-1],,i,t-1])==1)
      CH.TRUE[i,t] <- state
      # Multinomial trials for observation process
      event <- which(rmultinom(1, 1, PSI.OBS[CH.TRUE[i,t],,i,t-1])==1)
      CH[i,t] <- event
    } #t
  } #i
  # Replace the NA and the highest state number (dead) in the file by 0
  CH[is.na(CH)] <- 0
  CH[CH==dim(PSI.STATE)[1]] <- 0
  CH[CH==unobservable] <- 0
  id <- numeric(0)
  for (i in 1:dim(CH)[1]){
    z <- min(which(CH[i,]!=0))
    ifelse(z==dim(CH)[2], id <- c(id,i), id <- c(id))
  }
  return(list(CH=CH[-id,], CH.TRUE=CH.TRUE[-id,]))
# CH: capture histories to be used
# CH.TRUE: capture histories with perfect observation
}
```

Second, we use this function to simulate the two datasets of good and bad individuals:

```
set.seed(1) # for reproducibility
p = 0.9
R = 100
#-------------------------------
#---- good quality individuals
#-------------------------------
# Define mean survival, transitions, recapture, as well as number of
occasions, states, observations and released individuals
phiA <- 0.7
phiB <- 0.8
psiAB <- 0.8
psiBA <- 0.3
pA <- p
pB <- p
n.occasions <- 6
```

```r
n.states <- 3
n.obs <- 3
marked <- matrix(NA, ncol = n.states, nrow = n.occasions)
marked[,1] <- rep(R, n.occasions)
marked[,2] <- rep(R, n.occasions)
marked[,3] <- rep(0, n.occasions)
# Define matrices with survival, transition and recapture probabilities
# 1. State process matrix
totrel <- sum(marked)*(n.occasions-1)
PSI.STATE <- array(NA, dim=c(n.states, n.states, totrel, n.occasions-1))
for (i in 1:totrel){
  for (t in 1:(n.occasions-1)){
    PSI.STATE[,,i,t] <- matrix(c(
      phiA*(1-psiAB), phiA*psiAB, 1-phiA,
      phiB*psiBA, phiB*(1-psiBA), 1-phiB,
      0, 0, 1 ), nrow = n.states, byrow = TRUE)
  } #t
} #i
# 2.Observation process matrix
PSI.OBS <- array(NA, dim=c(n.states, n.obs, totrel, n.occasions-1))
for (i in 1:totrel){
  for (t in 1:(n.occasions-1)){
    PSI.OBS[,,i,t] <- matrix(c(
      pA, 0, 1-pA,
      0, pB, 1-pB,
      0, 0, 1 ), nrow = n.states, byrow = TRUE)
  } #t
} #i

# Execute function
sim <- simul.ms(PSI.STATE, PSI.OBS, marked)
CH <- sim$CH
his1 = CH[!apply(CH,1,sum)==0,] # remove lines of 0s

#------------------------------
#---- bad quality individuals
#------------------------------
# Define mean survival, transitions, recapture, as well as number of
occasions, states, observations and released individuals
phiA <- 0.7
phiB <- 0.6
psiAB <- 0.8
psiBA <- 0.3
pA <- p
pB <- p
n.occasions <- 6
n.states <- 3
n.obs <- 3
marked <- matrix(NA, ncol = n.states, nrow = n.occasions)
marked[,1] <- rep(R, n.occasions)
```

```
marked[,2] <- rep(R, n.occasions)
marked[,3] <- rep(0, n.occasions)
# Define matrices with survival, transition and recapture probabilities
# 1. State process matrix
totrel <- sum(marked)*(n.occasions-1)
PSI.STATE <- array(NA, dim=c(n.states, n.states, totrel, n.occasions-1))
for (i in 1:totrel){
  for (t in 1:(n.occasions-1)){
    PSI.STATE[,,i,t] <- matrix(c(
    phiA*(1-psiAB), phiA*psiAB, 1-phiA,
    phiB*psiBA, phiB*(1-psiBA), 1-phiB,
    0, 0, 1 ), nrow = n.states, byrow = TRUE)
  } #t
} #i
# 2.Observation process matrix
PSI.OBS <- array(NA, dim=c(n.states, n.obs, totrel, n.occasions-1))
for (i in 1:totrel){
  for (t in 1:(n.occasions-1)){
    PSI.OBS[,,i,t] <- matrix(c(
    pA, 0, 1-pA,
    0, pB, 1-pB,
    0, 0, 1 ), nrow = n.states, byrow = TRUE)
  } #t
} #i

# Execute function
sim <- simul.ms(PSI.STATE, PSI.OBS, marked)
CH <- sim$CH
his2 = CH[!apply(CH,1,sum)==0,] # remove lines of 0s
```

Last, we pool these two datasets together:

```
his = rbind(his1,his2)
head(his) # display first lines

##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    2    1    0    2    2
## [2,]    1    0    0    0    0    0
## [3,]    1    0    0    0    0    0
## [4,]    1    1    0    0    0    0
## [5,]    1    0    0    0    0    0
## [6,]    1    2    2    2    2    0

tail(his) # display last lines

##         [,1] [,2] [,3] [,4] [,5] [,6]
## [1995,]    0    0    0    0    2    2
## [1996,]    0    0    0    0    2    0
## [1997,]    0    0    0    0    2    2
## [1998,]    0    0    0    0    2    2
## [1999,]    0    0    0    0    2    0
## [2000,]    0    0    0    0    2    1
```

Model fitting

We adapted R code from Kéry and Schaub (2012) book (chapter 9) to fit multistate capture-recapture data.

Let us get the occasion of first capture for each individual:

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

```
##    [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##   [35] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##   [69] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2
##  [103] 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 2 2
##  [137] 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 2 2
##  [171] 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 3
##  [205] 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
##  [239] 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
##  [273] 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 4 4 4 4 4 4
##  [307] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
##  [341] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
##  [375] 4 4 4 4 4 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
##  [409] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
##  [443] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
##  [477] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 1 1 1 1 1 1 1 1 1 1
##  [511] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [545] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [579] 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
##  [613] 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 2 2
##  [647] 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 2 2
##  [681] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [715] 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
##  [749] 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
##  [783] 3 3 3 3 3 3 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
##  [817] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
##  [851] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
##  [885] 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 5 5 5 5
##  [919] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
##  [953] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
##  [987] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1021] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1055] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1089] 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
## [1123] 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 2 2
## [1157] 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 2 2
## [1191] 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [1225] 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
## [1259] 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
## [1293] 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 4 4 4
## [1327] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
## [1361] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
## [1395] 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
## [1429] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
## [1463] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
## [1497] 5 5 5 5 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1531] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1565] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1599] 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 2 2 2 2 2 2 2 2 2
## [1633] 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 2 2
## [1667] 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 2 2
## [1701] 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
## [1735] 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
## [1769] 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 4 4
## [1803] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
## [1837] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
## [1871] 4 4 4 4 4 4 4 4 4 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
## [1905] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
## [1939] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
## [1973] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
```

Recode the data such that 1 = seen alive in A, 2 = seen alive in B, 3 = not seen:

```
his_recoded <- his
his_recoded[his_recoded==0] <- 3
```

Now we fit a multistate model: we assume that survival depends on the breeding states, transition probabilities are constant over time, as well as the detection probability:

```
# sink("state_on_survival.jags")
# cat("
# model {
#
# # -------------------------------------------------
# # Parameters:
# # phiA: survival probability at site A
# # phiB: survival probability at site B
# # psiAB: movement probability from site A to site B
# # psiBA: movement probability from site B to site A
# # p: recapture probability at site A or B
# # -------------------------------------------------
# # States (S):
# # 1 alive at A
# # 2 alive at B
# # 3 dead
# # Observations (O):
# # 1 seen at A
# # 2 seen at B
# # 3 not seen
# # -------------------------------------------------
#
# # Priors
#    phiA ~ dunif(0, 1)
#    phiB ~ dunif(0, 1)
#    psiAB ~ dunif(0, 1)
#    psiBA ~ dunif(0, 1)
#    p ~ dunif(0, 1)
#
# # Define state-transition and observation matrices
# for (i in 1:nind){
#    # Define probabilities of state S(t+1) given S(t)
#    for (t in f[i]:(n.occasions-1)){
#       ps[1,i,t,1] <- phiA * (1-psiAB)
#       ps[1,i,t,2] <- phiA * psiAB
#       ps[1,i,t,3] <- 1-phiA
#       ps[2,i,t,1] <- phiB * psiBA
#       ps[2,i,t,2] <- phiB * (1-psiBA)
#       ps[2,i,t,3] <- 1-phiB
#       ps[3,i,t,1] <- 0
#       ps[3,i,t,2] <- 0
#       ps[3,i,t,3] <- 1
#
#       # Define probabilities of O(t) given S(t)
#       po[1,i,t,1] <- p
#       po[1,i,t,2] <- 0
#       po[1,i,t,3] <- 1-p
#       po[2,i,t,1] <- 0
#       po[2,i,t,2] <- p
#       po[2,i,t,3] <- 1-p
#       po[3,i,t,1] <- 0
#       po[3,i,t,2] <- 0
#       po[3,i,t,3] <- 1
#       } #t
#    } #i
#
# # Likelihood
# for (i in 1:nind){
#    # Define latent state at first capture
#    z[i,f[i]] <- y[i,f[i]]
#    for (t in (f[i]+1):n.occasions){
#       # State process: draw S(t) given S(t-1)
#       z[i,t] ~ dcat(ps[z[i,t-1], i, t-1,])
#       # Observation process: draw O(t) given S(t)
#       y[i,t] ~ dcat(po[z[i,t], i, t-1,])
#       } #t
#    } #i
# }
# ",fill = TRUE)
# sink()
```

```r
# Function to create known latent states z
known.state.ms <- function(ms, notseen){
   # notseen: label for ënot seení
   state <- ms
   state[state==notseen] <- NA
   for (i in 1:dim(ms)[1]){
      m <- min(which(!is.na(state[i,])))
      state[i,m] <- NA
      }
   return(state)
   }

# Function to create initial values for unknown z
ms.init.z <- function(ch, f){
   for (i in 1:dim(ch)[1]){ch[i,1:f[i]] <- NA}
   states <- max(ch, na.rm = TRUE)
   known.states <- 1:(states-1)
   v <- which(ch==states)
   ch[-v] <- NA
   ch[v] <- sample(known.states, length(v), replace = TRUE)
   return(ch)
   }

# Bundle data
jags.data <- list(y = his_recoded, f = f, n.occasions = dim(his_recoded)[2],
nind = dim(his_recoded)[1], z = known.state.ms(his_recoded, 3))

# Initial values
inits <- function(){list(phiA = runif(1, 0, 1), psiAB = runif(1, 0, 1), p =
runif(1, 0, 1), z = ms.init.z(his_recoded, f))}

# Parameters monitored
parameters <- c("phiA", "phiB", "psiAB", "psiBA", "p")

# MCMC settings
ni <- 5000
nb <- 2000
nc <- 2

# Call JAGS from R
library(R2jags)

## Loading required package: rjags

## Loading required package: coda

## Linked to JAGS 4.2.0

## Loaded modules: basemod,bugs
```

```
##
## Attaching package: 'R2jags'

## The following object is masked from 'package:coda':
##
##     traceplot

ms <- jags(jags.data, inits, parameters, "state_on_survival.jags", n.chains =
nc, n.iter = ni, n.burnin = nb)

## module glm loaded

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 8528
##    Unobserved stochastic nodes: 3477
##    Total graph size: 168970
##
## Initializing model

print(ms, digits = 3)

## Inference for Bugs model at "state_on_survival.jags", fit using jags,
##  2 chains, each with 5000 iterations (first 2000 discarded), n.thin = 3
##  n.sims = 2000 iterations saved
##             mu.vect sd.vect     2.5%      25%      50%      75%    97.5%
## p            0.900   0.008    0.884    0.895    0.900    0.905    0.915
## phiA         0.692   0.013    0.668    0.684    0.692    0.701    0.718
## phiB         0.700   0.010    0.679    0.693    0.700    0.707    0.720
## psiAB        0.776   0.014    0.750    0.767    0.777    0.786    0.803
## psiBA        0.308   0.012    0.285    0.301    0.309    0.317    0.332
## deviance 6604.931  81.246 6450.424 6550.027 6600.481 6657.918 6774.438
##             Rhat n.eff
## p        1.001  2000
## phiA     1.003   650
## phiB     1.001  2000
## psiAB    1.001  2000
## psiBA    1.001  2000
## deviance 1.001  2000
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 3301.6 and DIC = 9906.6
## DIC is an estimate of expected predictive error (lower deviance is
better).
```

Run same model without state effect on survival:

```
# sink("no_state_on_survival.jags")
# cat("
# model {
#
# # -------------------------------------------------
# # Parameters:
# # phi: survival probability at site A or B
# # psiAB: movement probability from site A to site B
# # psiBA: movement probability from site B to site A
# # p: recapture probability at site A or B
# # -------------------------------------------------
# # States (S):
# # 1 alive at A
# # 2 alive at B
# # 3 dead
# # Observations (O):
# # 1 seen at A
# # 2 seen at B
# # 3 not seen
# # -------------------------------------------------
#
# # Priors
#    phi ~ dunif(0, 1)
#    psiAB ~ dunif(0, 1)
#    psiBA ~ dunif(0, 1)
#    p ~ dunif(0, 1)
#
# # Define state-transition and observation matrices
# for (i in 1:nind){
#    # Define probabilities of state S(t+1) given S(t)
#    for (t in f[i]:(n.occasions-1)){
#       ps[1,i,t,1] <- phi * (1-psiAB)
#       ps[1,i,t,2] <- phi * psiAB
#       ps[1,i,t,3] <- 1-phi
#       ps[2,i,t,1] <- phi * psiBA
#       ps[2,i,t,2] <- phi * (1-psiBA)
#       ps[2,i,t,3] <- 1-phi
#       ps[3,i,t,1] <- 0
#       ps[3,i,t,2] <- 0
#       ps[3,i,t,3] <- 1
#
#       # Define probabilities of O(t) given S(t)
#       po[1,i,t,1] <- p
#       po[1,i,t,2] <- 0
#       po[1,i,t,3] <- 1-p
#       po[2,i,t,1] <- 0
#       po[2,i,t,2] <- p
#       po[2,i,t,3] <- 1-p
#       po[3,i,t,1] <- 0
#       po[3,i,t,2] <- 0
```

```r
#       po[3,i,t,3] <- 1
#       } #t
#    } #i
#
# # Likelihood
# for (i in 1:nind){
#    # Define latent state at first capture
#    z[i,f[i]] <- y[i,f[i]]
#    for (t in (f[i]+1):n.occasions){
#       # State process: draw S(t) given S(t-1)
#       z[i,t] ~ dcat(ps[z[i,t-1], i, t-1,])
#       # Observation process: draw O(t) given S(t)
#       y[i,t] ~ dcat(po[z[i,t], i, t-1,])
#       } #t
#    } #i
# }
# ",fill = TRUE)
# sink()

# Function to create known latent states z
known.state.ms <- function(ms, notseen){
   # notseen: label for ënot seení
   state <- ms
   state[state==notseen] <- NA
   for (i in 1:dim(ms)[1]){
      m <- min(which(!is.na(state[i,])))
      state[i,m] <- NA
      }
   return(state)
   }

# Function to create initial values for unknown z
ms.init.z <- function(ch, f){
   for (i in 1:dim(ch)[1]){ch[i,1:f[i]] <- NA}
   states <- max(ch, na.rm = TRUE)
   known.states <- 1:(states-1)
   v <- which(ch==states)
   ch[-v] <- NA
   ch[v] <- sample(known.states, length(v), replace = TRUE)
   return(ch)
   }

# Bundle data
jags.data <- list(y = his_recoded, f = f, n.occasions = dim(his_recoded)[2],
nind = dim(his_recoded)[1], z = known.state.ms(his_recoded, 3))

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

```
# Parameters monitored
parameters <- c("phi", "psiAB", "psiBA", "p")

# MCMC settings
ni <- 5000
nb <- 2000
nc <- 2

# Call JAGS from R
library(R2jags)
ms_without <- jags(jags.data, inits, parameters, "no_state_on_survival.jags",
n.chains = nc, n.iter = ni, n.burnin = nb)

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 8528
##    Unobserved stochastic nodes: 3476
##    Total graph size: 168966
##
## Initializing model

print(ms_without, digits = 3)

## Inference for Bugs model at "no_state_on_survival.jags", fit using jags,
##  2 chains, each with 5000 iterations (first 2000 discarded), n.thin = 3
##  n.sims = 2000 iterations saved
##           mu.vect sd.vect    2.5%      25%      50%      75%    97.5%
## p           0.900   0.008    0.884    0.894    0.900    0.905    0.915
## phi         0.697   0.008    0.681    0.692    0.697    0.702    0.712
## psiAB       0.777   0.014    0.750    0.768    0.777    0.786    0.803
## psiBA       0.309   0.012    0.286    0.301    0.308    0.316    0.331
## deviance 6605.816  79.482 6450.834 6551.843 6605.110 6658.311 6759.925
##           Rhat n.eff
## p        1.004  2000
## phi      1.001  2000
## psiAB    1.001  2000
## psiBA    1.004   430
## deviance 1.001  2000
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 3160.0 and DIC = 9765.8
## DIC is an estimate of expected predictive error (lower deviance is
better).
```

If you compare DIC values, it sounds like the difference in survival of breeding vs. non-breeding individuals is not detected.

Let's add individual heterogeneity through an individual covariate for bad vs. good individuals:

```r
quality=c(rep(0,nrow(his1)),rep(1,nrow(his2))) # 0 for good, 1 for bad
quality
```

```
##    [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##   [35] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##   [69] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [103] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [137] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [171] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [205] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [239] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [273] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [307] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [341] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [375] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [409] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [443] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [477] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [511] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [545] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [579] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [613] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [647] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [681] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [715] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [749] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [783] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [817] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [851] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [885] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [919] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [953] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [987] 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
## [1021] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1055] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1089] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1123] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1157] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1191] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1225] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1259] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1293] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1327] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1361] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1395] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1429] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1463] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1497] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1531] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1565] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1599] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1633] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1667] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1701] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1735] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1769] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1803] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1837] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1871] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1905] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1939] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1973] 1 1 1 1 1 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 fit again the two models from above, including the effect of individual heterogeneity.

```
# sink("state_on_survival_withquality.jags")
# cat("
# model {
#
# # -------------------------------------------------
# # Parameters:
# # alpha, beta: regression parameters for survival probability at site A/B
# # psiAB: movement probability from site A to site B
# # psiBA: movement probability from site B to site A
# # p: recapture probability
# # -------------------------------------------------
# # States (S):
# # 1 alive at A
# # 2 alive at B
# # 3 dead
# # Observations (O):
# # 1 seen at A
# # 2 seen at B
# # 3 not seen
# # -------------------------------------------------
#
# # Priors
#    alphaA ~ dnorm(0, 0.01)
#    betaA ~ dnorm(0, 0.01)
#    alphaB ~ dnorm(0, 0.01)
#    betaB ~ dnorm(0, 0.01)
#    for (i in 1:nind){
#    logit(phiA[i]) <- alphaA + betaA * quality[i]
#    logit(phiB[i]) <- alphaB + betaB * quality[i]}
#    psiAB ~ dunif(0, 1)
#    psiBA ~ dunif(0, 1)
#    p ~ dunif(0, 1)
#
# # Define state-transition and observation matrices
# for (i in 1:nind){
#    # Define probabilities of state S(t+1) given S(t)
#    for (t in f[i]:(n.occasions-1)){
#       ps[1,i,t,1] <- phiA[i] * (1-psiAB)
#       ps[1,i,t,2] <- phiA[i] * psiAB
#       ps[1,i,t,3] <- 1-phiA[i]
#       ps[2,i,t,1] <- phiB[i] * psiBA
#       ps[2,i,t,2] <- phiB[i] * (1-psiBA)
#       ps[2,i,t,3] <- 1-phiB[i]
#       ps[3,i,t,1] <- 0
#       ps[3,i,t,2] <- 0
#       ps[3,i,t,3] <- 1
#
#       # Define probabilities of O(t) given S(t)
#       po[1,i,t,1] <- p
#       po[1,i,t,2] <- 0
#       po[1,i,t,3] <- 1-p
#       po[2,i,t,1] <- 0
#       po[2,i,t,2] <- p
#       po[2,i,t,3] <- 1-p
#       po[3,i,t,1] <- 0
#       po[3,i,t,2] <- 0
#       po[3,i,t,3] <- 1
#       } #t
#    } #i
#
# # Likelihood
# for (i in 1:nind){
#    # Define latent state at first capture
#    z[i,f[i]] <- y[i,f[i]]
#    for (t in (f[i]+1):n.occasions){
#       # State process: draw S(t) given S(t-1)
#       z[i,t] ~ dcat(ps[z[i,t-1], i, t-1,])
#       # Observation process: draw O(t) given S(t)
#       y[i,t] ~ dcat(po[z[i,t], i, t-1,])
#       } #t
#    } #i
# }
# ",fill = TRUE)
# sink()

# Function to create known latent states z
known.state.ms <- function(ms, notseen){
   # notseen: label for ënot seení
   state <- ms
   state[state==notseen] <- NA
   for (i in 1:dim(ms)[1]){
      m <- min(which(!is.na(state[i,])))
      state[i,m] <- NA
      }
   return(state)
   }

# Function to create initial values for unknown z
ms.init.z <- function(ch, f){
   for (i in 1:dim(ch)[1]){ch[i,1:f[i]] <- NA}
   states <- max(ch, na.rm = TRUE)
   known.states <- 1:(states-1)
   v <- which(ch==states)
   ch[-v] <- NA
   ch[v] <- sample(known.states, length(v), replace = TRUE)
   return(ch)
   }

# Bundle data
jags.data <- list(y = his_recoded, f = f, quality = quality, n.occasions =
dim(his_recoded)[2], nind = dim(his_recoded)[1], z =
known.state.ms(his_recoded, 3))

# Initial values
inits <- function(){list(alphaA = rnorm(1, 0, 1), betaA = rnorm(1, 0, 1),
psiAB = runif(1, 0, 1), p = runif(1, 0, 1), z = ms.init.z(his_recoded, f))}

# Parameters monitored
parameters <- c("alphaA", "alphaB","betaA", "betaB", "psiAB", "psiBA", "p")

# MCMC settings
ni <- 5000
nb <- 2000
nc <- 2

# Call JAGS from R
library(R2jags)
ms_quality <- jags(jags.data, inits, parameters,
"state_on_survival_withquality.jags", n.chains = nc, n.iter = ni, n.burnin =
nb)

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 8528
##    Unobserved stochastic nodes: 3479
##    Total graph size: 194994
##
## Initializing model
```

```
print(ms_quality, digits = 3)

## Inference for Bugs model at "state_on_survival_withquality.jags", fit
using jags,
##  2 chains, each with 5000 iterations (first 2000 discarded), n.thin = 3
##  n.sims = 2000 iterations saved
##           mu.vect sd.vect     2.5%      25%      50%      75%    97.5%
## alphaA      0.787   0.083    0.627    0.732    0.789    0.842    0.952
## alphaB      1.393   0.078    1.245    1.341    1.390    1.444    1.551
## betaA       0.057   0.121   -0.175   -0.025    0.055    0.137    0.295
## betaB      -1.079   0.099   -1.270   -1.147   -1.080   -1.013   -0.883
## p           0.900   0.008    0.884    0.894    0.900    0.905    0.915
## psiAB       0.777   0.013    0.749    0.768    0.778    0.786    0.801
## psiBA       0.309   0.012    0.285    0.301    0.308    0.317    0.332
## deviance 6572.069  78.824 6417.383 6518.582 6572.417 6625.296 6726.832
##           Rhat n.eff
## alphaA   1.009   180
## alphaB   1.003   540
## betaA    1.005   380
## betaB    1.001  2000
## p        1.001  2000
## psiAB    1.000  2000
## psiBA    1.004   450
## deviance 1.001  2000
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 3107.7 and DIC = 9679.8
## DIC is an estimate of expected predictive error (lower deviance is
better).
```

Same model without state effect on survival:

```
# sink("no_state_on_survival_withquality.jags")
# cat("
# model {
#
# # -------------------------------------------------
# # Parameters:
# # alpha, beta: regression parameters for survival probability at site A/B
# # psiAB: movement probability from site A to site B
# # psiBA: movement probability from site B to site A
# # pA: recapture probability
# # -------------------------------------------------
# # States (S):
# # 1 alive at A
# # 2 alive at B
# # 3 dead
# # Observations (O):
```

```
# # 1 seen at A
# # 2 seen at B
# # 3 not seen
# # -------------------------------------------------
#
# # Priors
#    alpha ~ dnorm(0, 0.01)
#    beta ~ dnorm(0, 0.01)
#    for (i in 1:nind){logit(phi[i]) <- alpha + beta * quality[i]}
#    psiAB ~ dunif(0, 1)
#    psiBA ~ dunif(0, 1)
#    p ~ dunif(0, 1)
#
# # Define state-transition and observation matrices
# for (i in 1:nind){
#    # Define probabilities of state S(t+1) given S(t)
#    for (t in f[i]:(n.occasions-1)){
#       ps[1,i,t,1] <- phi[i] * (1-psiAB)
#       ps[1,i,t,2] <- phi[i] * psiAB
#       ps[1,i,t,3] <- 1-phi[i]
#       ps[2,i,t,1] <- phi[i] * psiBA
#       ps[2,i,t,2] <- phi[i] * (1-psiBA)
#       ps[2,i,t,3] <- 1-phi[i]
#       ps[3,i,t,1] <- 0
#       ps[3,i,t,2] <- 0
#       ps[3,i,t,3] <- 1
#
#       # Define probabilities of O(t) given S(t)
#       po[1,i,t,1] <- p
#       po[1,i,t,2] <- 0
#       po[1,i,t,3] <- 1-p
#       po[2,i,t,1] <- 0
#       po[2,i,t,2] <- p
#       po[2,i,t,3] <- 1-p
#       po[3,i,t,1] <- 0
#       po[3,i,t,2] <- 0
#       po[3,i,t,3] <- 1
#       } #t
#    } #i
#
# # Likelihood
# for (i in 1:nind){
#    # Define latent state at first capture
#    z[i,f[i]] <- y[i,f[i]]
#    for (t in (f[i]+1):n.occasions){
#       # State process: draw S(t) given S(t-1)
#       z[i,t] ~ dcat(ps[z[i,t-1], i, t-1,])
#       # Observation process: draw O(t) given S(t)
#       y[i,t] ~ dcat(po[z[i,t], i, t-1,])
#       } #t
```

```r
#     } #i
# }
# ",fill = TRUE)
# sink()

# Function to create known latent states z
known.state.ms <- function(ms, notseen){
   # notseen: label for ënot seení
   state <- ms
   state[state==notseen] <- NA
   for (i in 1:dim(ms)[1]){
      m <- min(which(!is.na(state[i,])))
      state[i,m] <- NA
      }
   return(state)
   }

# Function to create initial values for unknown z
ms.init.z <- function(ch, f){
   for (i in 1:dim(ch)[1]){ch[i,1:f[i]] <- NA}
   states <- max(ch, na.rm = TRUE)
   known.states <- 1:(states-1)
   v <- which(ch==states)
   ch[-v] <- NA
   ch[v] <- sample(known.states, length(v), replace = TRUE)
   return(ch)
   }

# Bundle data
jags.data <- list(y = his_recoded, f = f, quality = quality, n.occasions =
dim(his_recoded)[2], nind = dim(his_recoded)[1], z =
known.state.ms(his_recoded, 3))

# Initial values
inits <- function(){list(alpha = rnorm(1, 0, 1), beta = rnorm(1, 0, 1), psiAB
= runif(1, 0, 1), p = runif(1, 0, 1), z = ms.init.z(his_recoded, f))}

# Parameters monitored
parameters <- c("alpha","beta", "psiAB", "psiBA", "p")

# MCMC settings
ni <- 5000
nb <- 2000
nc <- 2

# Call JAGS from R
library(R2jags)
ms_quality_without <- jags(jags.data, inits, parameters,
"no_state_on_survival_withquality.jags", n.chains = nc, n.iter = ni, n.burnin
= nb)

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 8528
##    Unobserved stochastic nodes: 3477
##    Total graph size: 194980
##
## Initializing model

print(ms_quality_without, digits = 3)

## Inference for Bugs model at "no_state_on_survival_withquality.jags", fit
using jags,
##  2 chains, each with 5000 iterations (first 2000 discarded), n.thin = 3
##  n.sims = 2000 iterations saved
##          mu.vect sd.vect     2.5%      25%      50%      75%    97.5%
## alpha      1.138   0.055    1.033    1.100    1.139    1.176    1.244
## beta      -0.620   0.074   -0.757   -0.670   -0.620   -0.572   -0.472
## p          0.899   0.008    0.883    0.894    0.899    0.904    0.914
## psiAB      0.778   0.013    0.751    0.768    0.778    0.787    0.803
## psiBA      0.309   0.012    0.286    0.301    0.309    0.316    0.332
## deviance 6589.290  79.332 6433.388 6534.157 6586.661 6646.266 6741.670
##          Rhat n.eff
## alpha    1.002  1200
## beta     1.001  2000
## p        1.002  1200
## psiAB    1.004   480
## psiBA    1.001  2000
## deviance 1.001  1800
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 3146.6 and DIC = 9735.9
## DIC is an estimate of expected predictive error (lower deviance is
better).
```

Clearly, the inclusion of quality improves the DIC. Also, the model with a difference in survival between breeders and non-breeders is better supported by the data when individual heterogeneity is accounted for.

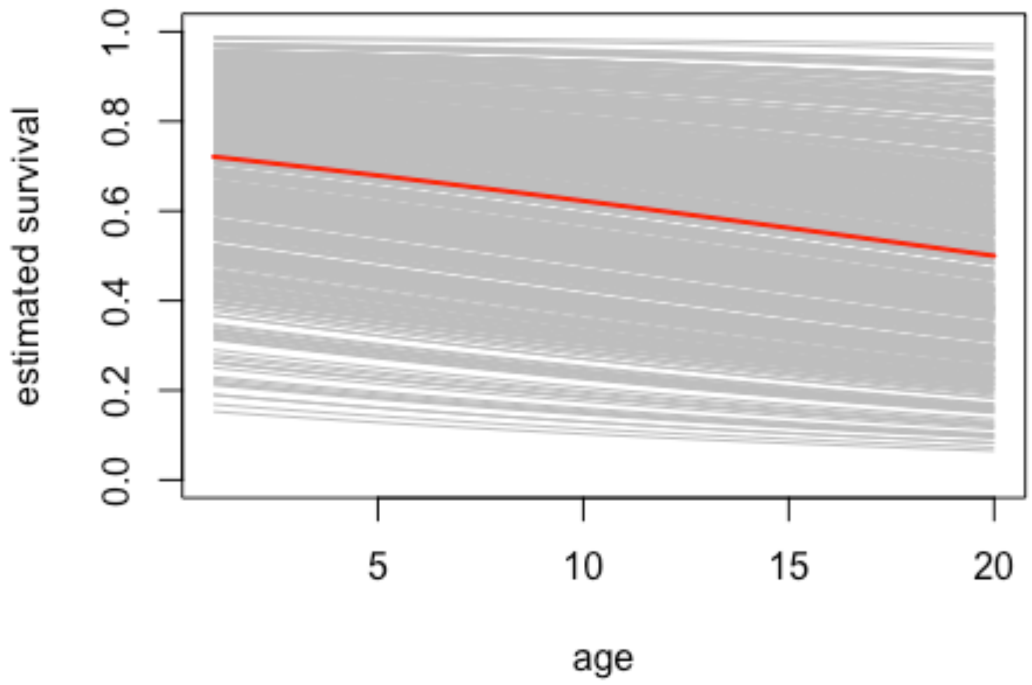
Models with individual random effects

Here, we aim at illustrating how not accounting for individual heterogeneity may obscure the detection of senescence in survival. More specifically, we consider a single cohort of 500 individuals with survival decreasing as they age over a 20-year study. We also add a frailty for each individual under the form of a normal distribution. Specifically, we specify $logit(\phi_i(a)) = \beta_0 + \beta_1 a + \varepsilon_i$ where $\varepsilon_i \sim N(0, \sigma^2)$. We use $\beta_0 = 1, \beta_1 = -0.05$ and $\sigma = 1$. If we condition upon the random effect, survival is decreasing as age increases. Note that we consider the same detection probability $p = 0.5$ for all individuals.

Data simulation

First, we simulate survival for each individual then plot the individual trajectories (in grey) as well as survival conditional on the random effect (in red):

```r
rm(list=ls())
r = set.seed(3) # for reproducibility
p = 0.5 # detection
intercept_phi = 1
slope_phi = -0.05
sigmaphi = 1
nind = 500 # nb of individuals
nyear = 20 # duration of the study
expit<-function(x){exp(x)/(1+exp(x))} # reciprocal logit function
z<-data<-x<-matrix(NA,nrow=nind,ncol=nyear)
first<-rep(1,nind)
age = matrix(NA,nind,nyear)
phi = matrix(NA,nind,nyear)
# simulate age-varying survival for each individual
for (i in 1:nind){
  mask <- first[i]:nyear
  age[i,mask] <- mask - first[i] + 1
  phi[i,mask] <- expit(intercept_phi + slope_phi * age[i,mask] +
rnorm(1,0,sigmaphi))
}
plot(age[1,],phi[1,],type='l',col='grey',ylim=c(0,1),xlab='age',ylab='estimat
ed survival')
for (i in 2:nind){
  lines(age[i,],phi[i,],type='l',col='grey')
}
lines(1:nyear,expit(intercept_phi + slope_phi * 1:nyear),col='red',lwd=2)
```



Now simulate the encounter histories:

```
for(i in 1:nind){
  z[i,first[i]] <- x[i,first[i]] <- 1
  for(j in (first[i]+1):nyear){
    z[i,j]<-rbinom(1,1,phi[i,j-1]*z[i,j-1])
    x[i,j]<-rbinom(1,1,z[i,j]*p)
  }
}
his = x
his[is.na(his)]=0 # remove lines with 0's
```

Model fitting

We fit the model with an age effect but no individual heterogeneity to the simulated dataset:

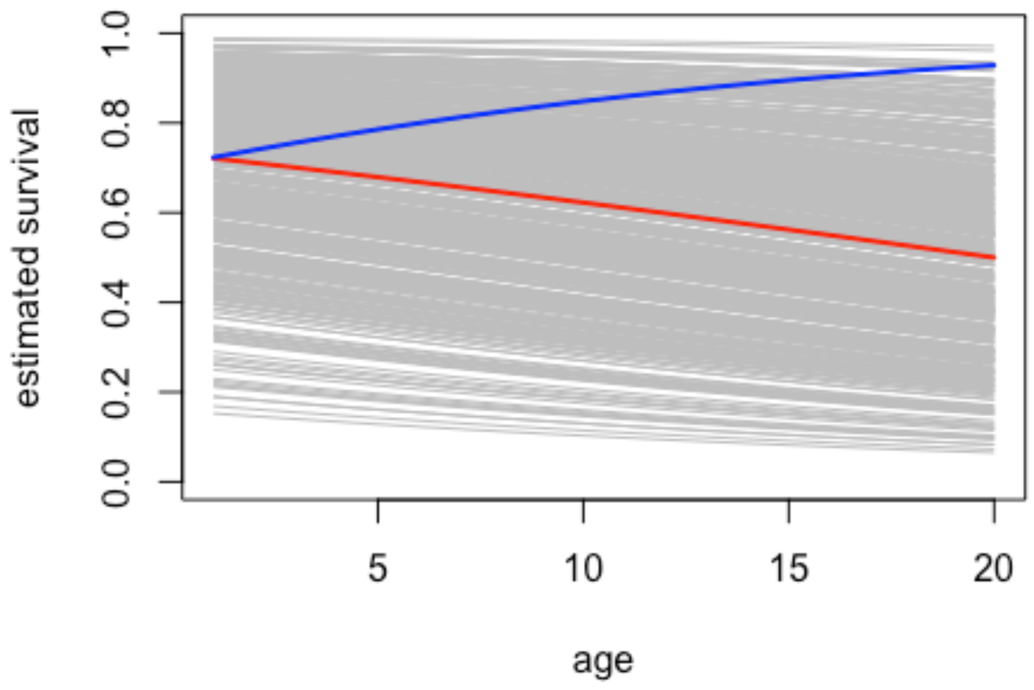
```
# sink("cjs_age.jags")
# cat("
# model {
#
# # Priors and constraints
# for (i in 1:nind){
#    for (t in f[i]:(n.occasions-1)){
#       logit(phi[i,t]) <- alpha + beta * t # a single cohort is used here,
so that age = time elapsed since first capture = time
#       p[i,t] <- mean.p
#       } #t
#    } #i
# alpha ~ dnorm(0, 0.01)                  # Prior for intercept of age
effect
# beta ~ dnorm(0, 0.01)                  # Prior for slope of age effect
# mean.p ~ dunif(0, 1)                    # Prior for mean recapture
#
# # Likelihood
# for (i in 1:nind){
#    # Define latent state at first capture
#    z[i,f[i]] <- 1
#    for (t in (f[i]+1):n.occasions){
#       # State process
#       z[i,t] ~ dbern(mu1[i,t])
#       mu1[i,t] <- phi[i,t-1] * z[i,t-1]
#       # Observation process
#       y[i,t] ~ dbern(mu2[i,t])
#       mu2[i,t] <- p[i,t-1] * z[i,t]
#       } #t
#    } #i
# }
# ",fill = TRUE)
# sink()

# Bundle data

known.state.cjs <- function(ch){
   state <- ch
   for (i in 1:dim(ch)[1]){
      n1 <- min(which(ch[i,]==1))
      n2 <- max(which(ch[i,]==1))
      state[i,n1:n2] <- 1
      state[i,n1] <- NA
      }
   state[state==0] <- NA
   return(state)
   }

jags.data <- list(y = his, f = first, nind = dim(his)[1], n.occasions =
dim(his)[2], z = known.state.cjs(his))

# Initial values

cjs.init.z <- function(ch,f){
   for (i in 1:dim(ch)[1]){
      if (sum(ch[i,])==1) next
      n2 <- max(which(ch[i,]==1))
      ch[i,f[i]:n2] <- NA
      }
   for (i in 1:dim(ch)[1]){
   ch[i,1:f[i]] <- NA
   }
   return(ch)
   }

inits <- function(){list(z = cjs.init.z(his, first), alpha = rnorm(1, 0, 1),
mean.p = runif(1, 0, 1), beta = rnorm(1, 0, 1))}

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

# MCMC settings
ni <- 5000
nb <- 2000
nc <- 2

# Call JAGS from R
library(R2jags)
cjs_age <- jags(jags.data, inits, parameters, "cjs_age.jags", n.chains = nc,
n.iter = ni, n.burnin = nb)

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 10903
##    Unobserved stochastic nodes: 8100
##    Total graph size: 45786
##
## Initializing model

# Summarize posteriors
print(cjs_age, digits = 3)

## Inference for Bugs model at "cjs_age.jags", fit using jags,
##  2 chains, each with 5000 iterations (first 2000 discarded), n.thin = 3
##  n.sims = 2000 iterations saved
##           mu.vect sd.vect     2.5%      25%      50%      75%    97.5%
## alpha       0.876   0.093    0.693    0.816    0.876    0.941    1.063
## beta        0.085   0.018    0.051    0.072    0.084    0.096    0.121
## mean.p      0.497   0.015    0.467    0.488    0.497    0.507    0.526
## deviance 3051.846  43.122 2968.525 3023.216 3051.532 3080.882 3136.516
##           Rhat n.eff
## alpha    1.004  1400
## beta     1.003  2000
## mean.p   1.001  2000
## deviance 1.003   580
##
```

```
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 928.6 and DIC = 3980.5
## DIC is an estimate of expected predictive error (lower deviance is
better).
```

Having a look to the parameter estimates, it sounds like the slope of the age effect on survival is estimated positive...

Which means that at the population level, whenever individual heterogeneity is ignored, then senescence (in red) is completely masked. Even worse, survival is increasing with increasing age (in blue).

```
plot(age[1,],phi[1,],type='l',col='grey',ylim=c(0,1),xlab='age',ylab='estimat
ed survival')
for (i in 2:nind){
  lines(age[i,],phi[i,],type='l',col='grey')
}
lines(1:nyear,expit(intercept_phi + slope_phi * 1:nyear),col='red',lwd=2)
lines(1:nyear,expit(mean(cjs_age$BUGSoutput$sims.matrix[,'alpha']) +
mean(cjs_age$BUGSoutput$sims.matrix[,'beta']) * 1:nyear),col='blue',lwd=2)
```



Now we fit the model with a random effect in the survival process.

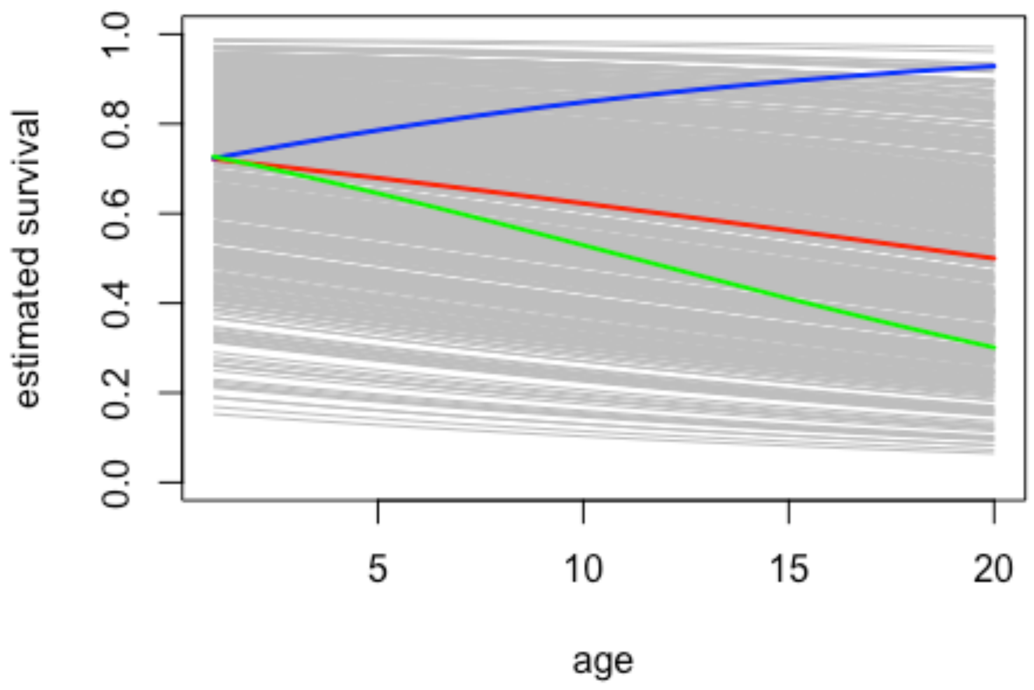
```
# sink("cjs_age_re.jags")
# cat("
# model {
#
# # Priors and constraints
# for (i in 1:nind){
#    for (t in f[i]:(n.occasions-1)){
#       logit(phi[i,t]) <- alpha + beta * t + epsilon[i] # a single cohort is
used here, so that age = time elapsed since first capture = time
#       p[i,t] <- mean.p
#       } #t
#    } #i
# for (i in 1:nind){
#    epsilon[i] ~ dnorm(0, tau)
#    }
# alpha ~ dnorm(0, 0.01)                    # Prior for intercept of age
effect
# beta ~ dnorm(0, 0.01)                    # Prior for slope of age effect
# sigma ~ dunif(0, 5)                       # Prior for standard deviation
# tau <- pow(sigma, -2)
# sigma2 <- pow(sigma, 2)
# mean.p ~ dunif(0, 1)                      # Prior for mean recapture
#
# # Likelihood
# for (i in 1:nind){
#    # Define latent state at first capture
#    z[i,f[i]] <- 1
#    for (t in (f[i]+1):n.occasions){
#       # State process
#       z[i,t] ~ dbern(mu1[i,t])
#       mu1[i,t] <- phi[i,t-1] * z[i,t-1]
#       # Observation process
#       y[i,t] ~ dbern(mu2[i,t])
#       mu2[i,t] <- p[i,t-1] * z[i,t]
#       } #t
#    } #i
# }
# ",fill = TRUE)
# sink()

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

# Initial values
inits <- function(){list(z = cjs.init.z(his, first), alpha = rnorm(1, 0, 1),
beta = rnorm(1, 0, 1),mean.p = runif(1, 0, 1), sigma = runif(1, 0, 2))}
```

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

# MCMC settings
ni <- 5000
nb <- 2000
nc <- 2

# Call JAGS from R
cjs_age_re <- jags(jags.data, inits, parameters, "cjs_age_re.jags", n.chains
= nc, n.iter = ni, n.burnin = nb)

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 10903
##    Unobserved stochastic nodes: 8601
##    Total graph size: 77638
##
## Initializing model

# Summarize posteriors
print(cjs_age_re, digits = 3)

## Inference for Bugs model at "cjs_age_re.jags", fit using jags,
##  2 chains, each with 5000 iterations (first 2000 discarded), n.thin = 3
##  n.sims = 2000 iterations saved
##           mu.vect sd.vect     2.5%      25%      50%      75%    97.5%
## alpha       1.076   0.136    0.823    0.983    1.073    1.168    1.363
## beta       -0.096   0.052   -0.208   -0.132   -0.089   -0.055   -0.014
## mean.p      0.505   0.015    0.476    0.496    0.505    0.515    0.535
## sigma2      2.024   0.872    0.843    1.351    1.847    2.524    4.163
## deviance 2882.854  54.349 2776.637 2846.857 2882.106 2919.539 2991.459
##           Rhat n.eff
## alpha    1.043    43
## beta     1.336     8
## mean.p   1.003   670
## sigma2   1.417     7
## deviance 1.155    14
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 1366.9 and DIC = 4249.8
## DIC is an estimate of expected predictive error (lower deviance is
better).
```

The intercept and slope of the age-survival relationship are quite close to the values we used to simulate the data.

Now we add to our previous plot the survival as estimated when individual heterogeneity is explicitely accounted for using individual random effects (in green):

```
plot(age[1,],phi[1,],type='l',col='grey',ylim=c(0,1),xlab='age',ylab='estimat
ed survival')
for (i in 2:nind){
  lines(age[i,],phi[i,],type='l',col='grey')
}
lines(1:nyear,expit(intercept_phi + slope_phi * 1:nyear),col='red',lwd=2)
lines(1:nyear,expit(mean(cjs_age$BUGSoutput$sims.matrix[,'alpha']) +
mean(cjs_age$BUGSoutput$sims.matrix[,'beta']) * 1:nyear),col='blue',lwd=2)
lines(1:nyear,expit(mean(cjs_age_re$BUGSoutput$sims.matrix[,'alpha']) +
mean(cjs_age_re$BUGSoutput$sims.matrix[,'beta']) *
1:nyear),col='green',lwd=2)
```



Models with finite mixtures

Here, we again aim at illustrating how not accounting for individual heterogeneity may obscure the detection of senescence in survival. In contrast with the previous section, we now use finite mixtures to deal with heterogeneity. More specifically, we consider a cohort of 1000 individuals that are split into a group of robust individuals in proportion π with constant high survival ϕ_R and a group of frail individuals with survival ϕ_F that senesce over the 20 years of the study according to the relationship $logit(\phi_F(a)) = \beta_0 + \beta_1 a$. We use

$\pi = 0.3$, $\phi_R = 0.85$, $\beta_0 = 0$ and $\beta_1 = -0.07$. Note that we consider the same detection probability $p = 0.5$ for all individuals.

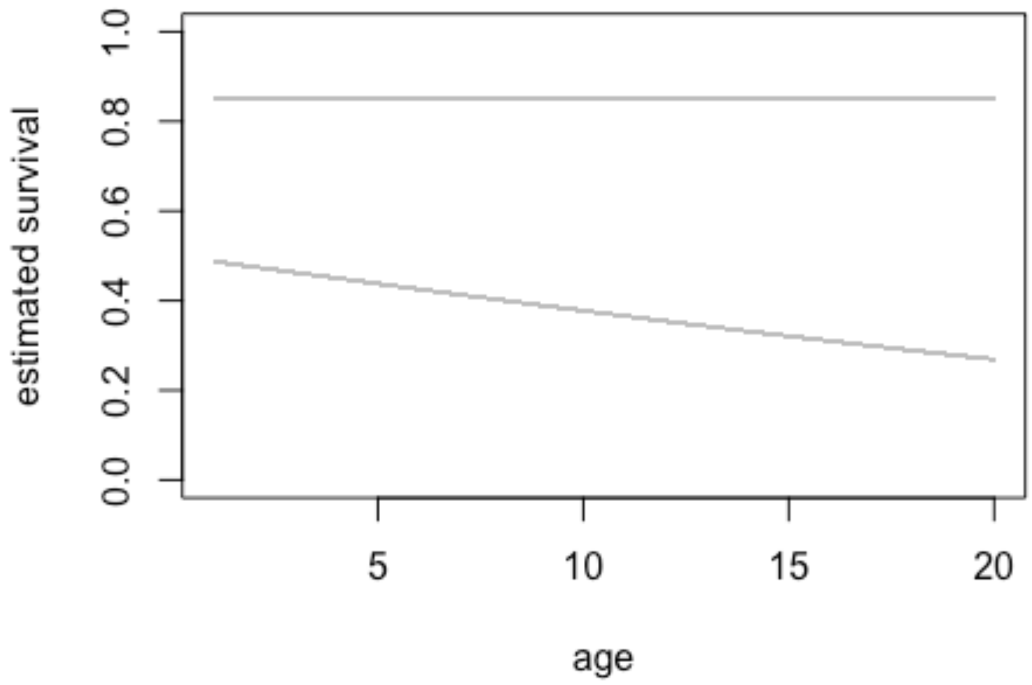
Data simulation

First simulate data

```
rm(list=ls())
r = set.seed(3) # for reproducibility
p = 0.5 # detection
prop_class1 = 0.3 # pi
phi_class1 = 0.85 # survival or robust ind
intercept_phi_class2 = 0 #beta_0
slope_phi_class2 = -0.05 # beta_1
nind = 1000 # nb of ind
nyear = 20 # duration of the study
expit<-function(x){exp(x)/(1+exp(x))} # reciprocal of the logit function
z<-data<-x<-matrix(NA,nrow=nind,ncol=nyear)
first<-rep(1,nind)
age = matrix(NA,nind,nyear)
phi = matrix(NA,nind,nyear)
which_mixture = rep(NA,nind)
# simulate age-varying survival for each individual,
# by first assigning them to the robust or frail class, then using the
corresponding
# survival
for (i in 1:nind){
  mask <- first[i]:nyear
  age[i,mask] <- mask - first[i] + 1
  which_mixture[i] <- rbinom(1,1,prop_class1) # assign ind i to a class with
prob pi
  if (which_mixture[i] == 1){
    phi[i,mask] <- phi_class1 # robust
  } else {
  phi[i,mask] <- expit(intercept_phi_class2 + slope_phi_class2 *
age[i,mask])} # frail
}
```

Represent graphically survival over time in the two classes:

```
plot(age[1,],phi[1,],type='l',col='grey',ylim=c(0,1),xlab='age',ylab='estimat
ed survival')
for (i in 2:nind){
  lines(age[i,],phi[i,],type='l',col='grey')
}
```



Now simulate the encounter histories:

```
for(i in 1:nind){
  z[i,first[i]] <- x[i,first[i]] <- 1
  for(j in (first[i]+1):nyear){
    z[i,j]<-rbinom(1,1,phi[i,j-1]*z[i,j-1])
    x[i,j]<-rbinom(1,1,z[i,j]*p)
  }
}
his = x
his[is.na(his)]=0
```

Model fitting

Let's fit two models assuming homogeneity, first one with constant survival probability, second one with an age effect:

```
# sink("Mct.jags")
# cat("
# model
# {
#
# # notation used
# # STATES
# # V for alive
# # D for dead
#
# # OBSERVATIONS
# # 0 = non-observed (coded 1)
# # 1 = observed (coded 2)
#
# # prior on survival for V1 and V2
# phi ~ dunif(0,1)
#
# # prior on detection
# p ~ dunif(0,1)
#
# # probabilities for each initial state
# px0[1] <- 1 # prob. of being in initial state V
# px0[2] <- 0 # prob. of being in initial state D
#
# # define probabilities of observations at t given states at t
# po[1,1] <- 1-p
# po[1,2] <- p
# po[2,1] <- 1
# po[2,2] <- 0
#
# po.init[1,1] <- 0
# po.init[1,2] <- 1
# po.init[2,1] <- 1
# po.init[2,2] <- 0
#
# # define probabilities of states at t given states at t-1
# px[1,1] <- phi
# px[1,2] <- 1-phi
# px[2,1] <- 0
# px[2,2] <- 1
#
# for (i in 1:nind){  # for each indiv
#   # initial states
#   z[i,first[i]] ~ dcat(px0[1:2])
#   x[i,first[i]] ~ dcat(po.init[z[i,first[i]],1:2])
#   for (j in (first[i]+1):nyear){  # loop over time
#
#     # state equations
#     z[i,j] ~ dcat(px[z[i,j-1],1:2])
#
#     # observation equations
#     x[i,j] ~ dcat(po[z[i,j],1:2])
#     }
# }
#
# }
#
```

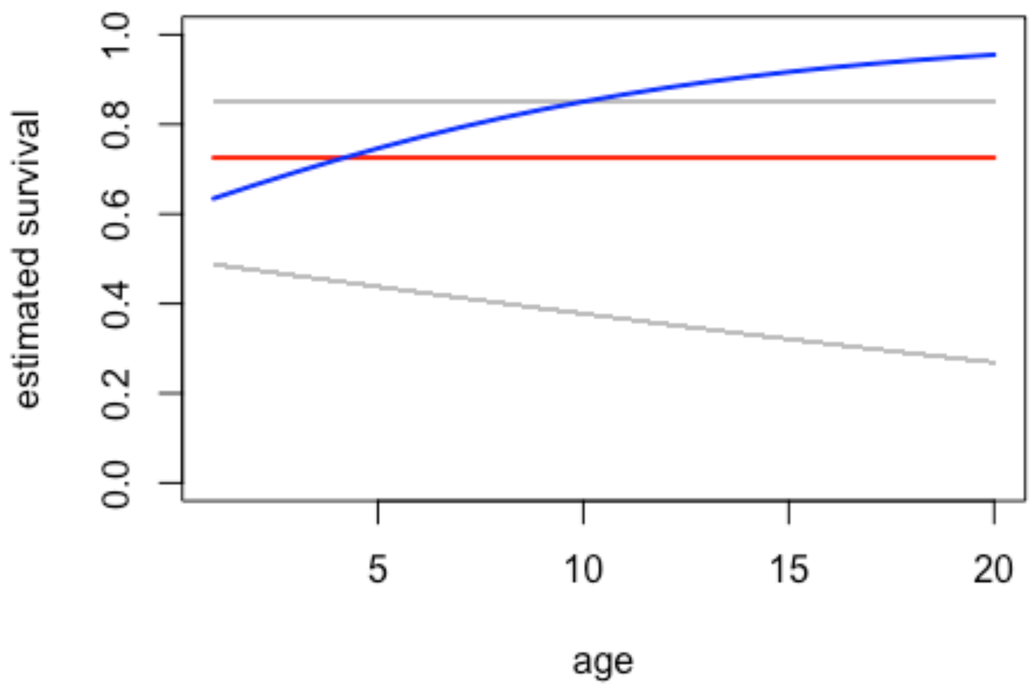
```
# ",fill=TRUE)
# sink()

mydatax <- list(x=his+1,first=first,nind=nind,nyear=nyear)

# initial values
x.init <- his
for (i in 1:nind){x.init[i,1:first[i]] <- NA}
x.init[x.init==0] <- 1
z = x.init
init1 <- list(phi=0.3,p=.2,z=z)
init2 <- list(phi=0.7,p=.8,z=z)
inits <- list(init1,init2)


# parameters to be monitored
parameters <- c("phi","p")

# MCMC settings
ni <- 5000
nb <- 2000
nc <- 2

# Call JAGS from R
library(R2jags)
start<-as.POSIXlt(Sys.time())
no_het_no_age <- jags(mydatax,inits,parameters,"Mct.jags",n.chains = nc,
n.iter = ni,n.burnin = nb)

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 20000
##    Unobserved stochastic nodes: 20002
##    Total graph size: 80031
##
## Initializing model

end <-as.POSIXlt(Sys.time())
duration = end-start

# Summarize posteriors
print(no_het_no_age, digits = 3)

## Inference for Bugs model at "Mct.jags", fit using jags,
##  2 chains, each with 5000 iterations (first 2000 discarded), n.thin = 3
##  n.sims = 2000 iterations saved
##          mu.vect sd.vect    2.5%     25%     50%     75%   97.5%
## p          0.476   0.012   0.451   0.468   0.476   0.484   0.502
## phi          0.726   0.008    0.709    0.720    0.726    0.731    0.742
## deviance 3608.389  52.593 3506.053 3572.926 3608.170 3643.858 3714.467
##           Rhat n.eff
## p        1.001  2000
## phi      1.001  2000
## deviance 1.001  2000
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 1383.6 and DIC = 4992.0
## DIC is an estimate of expected predictive error (lower deviance is
better).
```

```
# sink("Mage.jags")
# cat("
# model
# {
#
# # notation used
# # STATES
# # V for alive
# # D for dead
#
# # OBSERVATIONS
# # 0 = non-observed (coded 1)
# # 1 = observed (coded 2)
#
# # prior on survival
# alpha ~ dnorm(0,0.01)
# beta ~ dnorm(0,0.01)
# for (i in 1:nind){  # for each indiv
#   for (j in 1:(nyear-1)){  # loop over time
#     logit(phi[i,j]) <- alpha + beta * j # single cohort, hence age = time
#   }
# }
# # prior on detection
# p ~ dunif(0,1)
#
# # probabilities for each initial state
# px0[1] <- 1 # prob. of being in initial state V
# px0[2] <- 0 # prob. of being in initial state D
#
# # define probabilities of observations at t given states at t
# po[1,1] <- 1-p
# po[1,2] <- p
# po[2,1] <- 1
# po[2,2] <- 0
#
# po.init[1,1] <- 0
# po.init[1,2] <- 1
# po.init[2,1] <- 1
# po.init[2,2] <- 0
#
# # define probabilities of states at t given states at t-1
# for (i in 1:nind){  # for each indiv
#   for (j in 1:(nyear-1)){  # loop over time
#     px[1,i,j,1] <- phi[i,j]
#     px[1,i,j,2] <- 1-phi[i,j]
#     px[2,i,j,1] <- 0
#     px[2,i,j,2] <- 1
#   }
# }
# for (i in 1:nind){  # for each indiv
#   # initial states
#   z[i,first[i]] ~ dcat(px0[1:2])
#   x[i,first[i]] ~ dcat(po.init[z[i,first[i]],1:2])
#   for (j in (first[i]+1):nyear){  # loop over time
#
#     # state equations
#     z[i,j] ~ dcat(px[z[i,j-1],i,j-1,1:2])
#
#     # observation equations
#     x[i,j] ~ dcat(po[z[i,j],1:2])
#     }
# }
#
# }
#
# ",fill=TRUE)
# sink()

mydatax <- list(x=his+1,first=first,nind=nind,nyear=nyear)

# initial values
x.init <- his
for (i in 1:nind){x.init[i,1:first[i]] <- NA}
x.init[x.init==0] <- 1
z = x.init
init1 <- list(alpha=0.3,p=.2,z=z)
init2 <- list(alpha=0.7,p=.8,z=z)
inits <- list(init1,init2)


# parameters to be monitored
parameters <- c("alpha","beta","p")

# MCMC settings
ni <- 5000
```

```
nb <- 2000
nc <- 2

# Call JAGS from R
library(R2jags)
start<-as.POSIXlt(Sys.time())
no_het_with_age <- jags(mydatax,inits,parameters,"Mage.jags",n.chains = nc,
n.iter = ni,n.burnin = nb)

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 20000
##    Unobserved stochastic nodes: 20003
##    Total graph size: 251103
##
## Initializing model

end <-as.POSIXlt(Sys.time())
duration = end-start

# Summarize posteriors
print(no_het_with_age, digits = 3)

## Inference for Bugs model at "Mage.jags", fit using jags,
##  2 chains, each with 5000 iterations (first 2000 discarded), n.thin = 3
##  n.sims = 2000 iterations saved
##          mu.vect sd.vect    2.5%      25%      50%      75%    97.5%
## alpha      0.420   0.068    0.291    0.374    0.422    0.465    0.548
## beta       0.132   0.014    0.106    0.122    0.132    0.141    0.162
## p          0.499   0.013    0.475    0.490    0.499    0.508    0.524
## deviance 3449.533  51.375 3344.638 3414.307 3450.552 3483.450 3549.621
##          Rhat n.eff
## alpha    1.001  2000
## beta     1.001  1500
## p        1.006  2000
## deviance 1.001  2000
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 1320.3 and DIC = 4769.9
## DIC is an estimate of expected predictive error (lower deviance is
better).
```

Graphically, we have the estimate from the model with constant survival (in red) vs. age-varying survival (in blue):

```
plot(age[1,],phi[1,],type='l',col='grey',ylim=c(0,1),xlab='age',ylab='estimat
ed survival')
for (i in 2:nind){
  lines(age[i,],phi[i,],type='l',col='grey')
}
lines(1:nyear,rep(mean(no_het_no_age$BUGSoutput$sims.matrix[,'phi']),nyear),l
wd=2,col='red') # add survival from constant model
lines(1:nyear,expit(mean(no_het_with_age$BUGSoutput$sims.matrix[,'alpha'])+me
an(no_het_with_age$BUGSoutput$sims.matrix[,'beta'])*(1:nyear)),lwd=2,col='blu
e') # add survival from age model
```



Again, as in the previous section, it's striking to see that survival is increasing when age increases if individual heterogeneity is ignored. In other words, senescence is masked.

Now let's fit a model with heterogeneity in the survival probability, with constant parameters over time. Constant survival, two-finite mixture on the survival probability and a constant proportion of individual in each class:

```
# sink("M2class.jags")
# cat("
# model
# {
#
# # notation used
# # STATES
# # V1 for alive in class 1
# # V2 for alive in class 2
# # D for dead
#
# # OBSERVATIONS
# # 0 = non-observed (coded 1)
# # 1 = observed (coded 2)
#
# # prior on survival for V1 and V2
# phi1 ~ dunif(0,1)
# phi2 ~ dunif(0,1)
#
# # prior on detection
# p ~ dunif(0,1)
#
# # prior on initial state prob
# pi ~ dunif(0,1)
#
# # probabilities for each initial state
# px0[1] <- pi # prob. of being in initial state V1
# px0[2] <- 1-pi # prob. of being in initial state V2
# px0[3] <- 0 # prob. of being in initial state dead
#
# # define probabilities of observations at t given states at t
# po[1,1] <- 1-p
# po[1,2] <- p
# po[2,1] <- 1-p
# po[2,2] <- p
# po[3,1] <- 1
# po[3,2] <- 0
#
# po.init[1,1] <- 0
# po.init[1,2] <- 1
# po.init[2,1] <- 0
# po.init[2,2] <- 1
# po.init[3,1] <- 1
# po.init[3,2] <- 0
#
# # define probabilities of states at t given states at t-1
# px[1,1] <- phi1
# px[1,2] <- 0
# px[1,3] <- 1-phi1
# px[2,1] <- 0
# px[2,2] <- phi2
# px[2,3] <- 1-phi2
# px[3,1] <- 0
# px[3,2] <- 0
# px[3,3] <- 1
#
```

```
# for (i in 1:nind){  # for each indiv
#   # initial states
#   z[i,first[i]] ~ dcat(px0[1:3])
#   x[i,first[i]] ~ dcat(po.init[z[i,first[i]],1:2])
#   for (j in (first[i]+1):nyear){  # loop over time
#
#     # state equations
#     z[i,j] ~ dcat(px[z[i,j-1],1:3])
#
#     # observation equations
#     x[i,j] ~ dcat(po[z[i,j],1:2])
#     }
# }
#
# }
#
# ",fill=TRUE)
# sink()

mydatax <- list(x=his+1,first=first,nind=nind,nyear=nyear)

# initial values
x.init <- his
for (i in 1:nind){x.init[i,1:first[i]] <- NA}
x.init[x.init==0] <- 1
z = x.init
init1 <- list(phi1=0.3,phi2=.2,p=.2,z=z)
init2 <- list(phi1=0.7,phi2=.8,p=.8,z=z)
inits <- list(init1,init2)

# load package R2jags
library(R2jags)

# parameters to be monitored
parameters <- c("phi1","phi2","p")

# MCMC settings
ni <- 5000
nb <- 2000
nc <- 2

# Call JAGS from R
start<-as.POSIXlt(Sys.time())
with_het_no_age <- jags(mydatax,inits,parameters,"M2class.jags",n.chains =
nc, n.iter = ni,n.burnin = nb)

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
```

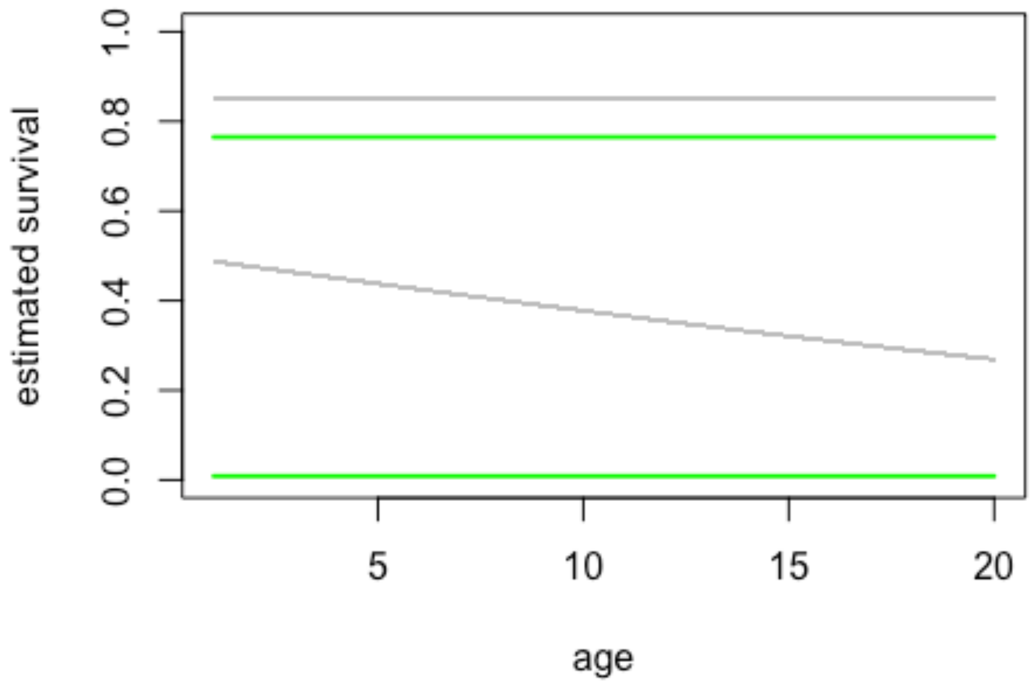
```
##    Observed stochastic nodes: 20000
##    Unobserved stochastic nodes: 20004
##    Total graph size: 80052
##
## Initializing model
```

```r
end <-as.POSIXlt(Sys.time())
duration = end-start

# Summarize posteriors
print(with_het_no_age, digits = 3)
```

```
## Inference for Bugs model at "M2class.jags", fit using jags,
##  2 chains, each with 5000 iterations (first 2000 discarded), n.thin = 3
##  n.sims = 2000 iterations saved
##           mu.vect sd.vect     2.5%      25%      50%      75%    97.5%
## p           0.503   0.013    0.477    0.494    0.503    0.511    0.527
## phi1        0.765   0.010    0.746    0.758    0.765    0.771    0.785
## phi2        0.009   0.009    0.000    0.002    0.005    0.011    0.034
## deviance 3425.315  54.950 3322.713 3387.547 3423.276 3461.700 3533.584
##           Rhat n.eff
## p        1.005   360
## phi1     1.001  2000
## phi2     1.002  1300
## deviance 1.004   400
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 1506.8 and DIC = 4932.1
## DIC is an estimate of expected predictive error (lower deviance is
better).
```

Let's have a look graphically:

```r
plot(age[1,],phi[1,],type='l',col='grey',ylim=c(0,1),xlab='age',ylab='estimat
ed survival')
for (i in 2:nind){
  lines(age[i,],phi[i,],type='l',col='grey')
}
lines(1:nyear,rep(mean(with_het_no_age$BUGSoutput$sims.matrix[,'phi1']),nyear
),lwd=2,col='green') # add survival from first class
lines(1:nyear,rep(mean(with_het_no_age$BUGSoutput$sims.matrix[,'phi2']),nyear
),lwd=2,col='green') # add survival from second class
```



Not too bad. Obviously, for frail individuals, we miss the age effect to be able to detect senescence. Now let's add age to this model:

```
# sink("M2classage.jags")
# cat("
# model
# {
#
# # notation used
# # STATES
# # V1 for alive in class 1
# # V2 for alive in class 2
# # D for dead
#
# # OBSERVATIONS
# # 0 = non-observed (coded 1)
# # 1 = observed (coded 2)
#
# # prior on survival for V1 and V2
# alpha1 ~ dnorm(0,0.01)
# beta1 ~ dnorm(0,0.01)
# alpha2 ~ dnorm(0,0.01)
# beta2 ~ dnorm(0,0.01)
# for (i in 1:nind){  # for each indiv
#   for (j in 1:(nyear-1)){  # loop over time
#     logit(phi1[i,j]) <- alpha1 + beta1 * j # single cohort, hence age =
time
#     logit(phi2[i,j]) <- alpha2 + beta2 * j # single cohort, hence age =
time
#   }
# }
#
# # prior on detection
# p ~ dunif(0,1)
#
# # prior on initial state prob
# pi ~ dunif(0,1)
#
# # probabilities for each initial state
# px0[1] <- pi # prob. of being in initial state V1
# px0[2] <- 1-pi # prob. of being in initial state V2
# px0[3] <- 0 # prob. of being in initial state dead
#
# # define probabilities of observations at t given states at t
# po[1,1] <- 1-p
# po[1,2] <- p
# po[2,1] <- 1-p
# po[2,2] <- p
# po[3,1] <- 1
# po[3,2] <- 0
#
# po.init[1,1] <- 0
# po.init[1,2] <- 1
# po.init[2,1] <- 0
# po.init[2,2] <- 1
# po.init[3,1] <- 1
# po.init[3,2] <- 0
#
# # define probabilities of states at t given states at t-1
# for (i in 1:nind){  # for each indiv
#   for (j in 1:(nyear-1)){  # loop over time
#     px[1,i,j,1] <- phi1[i,j]
#     px[1,i,j,2] <- 0
#     px[1,i,j,3] <- 1-phi1[i,j]
#     px[2,i,j,1] <- 0
#     px[2,i,j,2] <- phi2[i,j]
#     px[2,i,j,3] <- 1-phi2[i,j]
#     px[3,i,j,1] <- 0
#     px[3,i,j,2] <- 0
#     px[3,i,j,3] <- 1
#   }
# }
#
```

```
# for (i in 1:nind){  # for each indiv
#   # initial states
#   z[i,first[i]] ~ dcat(px0[1:3])
#   x[i,first[i]] ~ dcat(po.init[z[i,first[i]],1:2])
#   for (j in (first[i]+1):nyear){  # loop over time
#
#     # state equations
#     z[i,j] ~ dcat(px[z[i,j-1],i,j-1,1:3])
#
#     # observation equations
#     x[i,j] ~ dcat(po[z[i,j],1:2])
#     }
# }
#
# }
#
# ",fill=TRUE)
# sink()

mydatax <- list(x=his+1,first=first,nind=nind,nyear=nyear)

# initial values
x.init <- his
for (i in 1:nind){x.init[i,1:first[i]] <- NA}
x.init[x.init==0] <- 1
z = x.init
init1 <- list(alpha1=1,alpha2=-1,p=.2,z=z)
init2 <- list(alpha1=-1,alpha2=1,p=.8,z=z)
inits <- list(init1,init2)

# load package R2jags
library(R2jags)

# parameters to be monitored
parameters <- c("alpha1","alpha2","beta1","beta2","p")

# MCMC settings
ni <- 5000
nb <- 2000
nc <- 2

# Call JAGS from R
start<-as.POSIXlt(Sys.time())
with_het_with_age <- jags(mydatax,inits,parameters,"M2classage.jags",n.chains
= nc, n.iter = ni,n.burnin = nb)

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
```

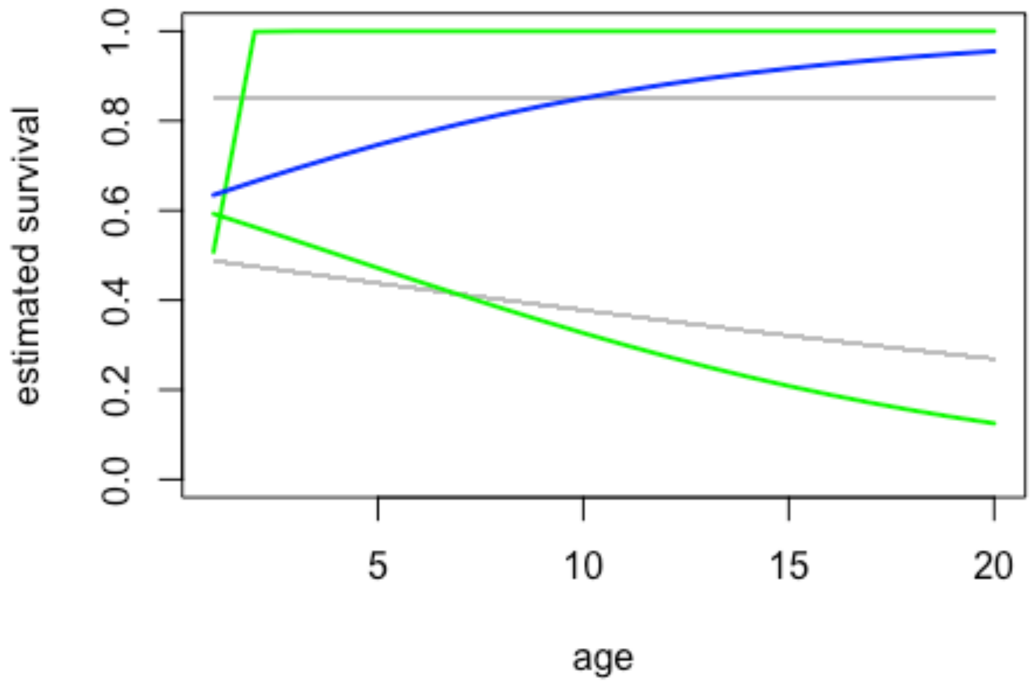
```
##    Observed stochastic nodes: 20000
##    Unobserved stochastic nodes: 20006
##    Total graph size: 422196
##
## Initializing model
```

```
end <-as.POSIXlt(Sys.time())
duration = end-start

# Summarize posteriors
print(with_het_with_age, digits = 3)
```

```
## Inference for Bugs model at "M2classage.jags", fit using jags,
##  2 chains, each with 5000 iterations (first 2000 discarded), n.thin = 3
##  n.sims = 2000 iterations saved
##            mu.vect sd.vect     2.5%      25%      50%      75%    97.5%
## alpha1       0.497   0.086    0.334    0.439    0.495    0.552    0.675
## alpha2      -6.658   8.412  -22.919  -12.234   -6.675   -1.179    9.874
## beta1        0.122   0.016    0.092    0.111    0.121    0.133    0.154
## beta2       -6.689   8.138  -22.960  -12.042   -6.428   -1.168    9.022
## p            0.501   0.012    0.478    0.493    0.501    0.509    0.526
## deviance  3431.579  51.912 3333.012 3396.749 3431.065 3466.650 3536.181
##            Rhat n.eff
## alpha1    1.025    66
## alpha2    1.001  1800
## beta1     1.017    93
## beta2     1.001  2000
## p         1.001  2000
## deviance 1.003  1800
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 1347.3 and DIC = 4778.9
## DIC is an estimate of expected predictive error (lower deviance is
better).
```

Let's have a look graphically:

```
plot(age[1,],phi[1,],type='l',col='grey',ylim=c(0,1),xlab='age',ylab='estimat
ed survival')
for (i in 2:nind){
  lines(age[i,],phi[i,],type='l',col='grey')
}
phi1 = 1/(1+exp(-mean(with_het_with_age$BUGSoutput$sims.matrix[,'alpha1']) +
mean(with_het_with_age$BUGSoutput$sims.matrix[,'beta1']) * 1:nyear))
phi2 = 1/(1+exp(-mean(with_het_with_age$BUGSoutput$sims.matrix[,'alpha2']) +
mean(with_het_with_age$BUGSoutput$sims.matrix[,'beta2']) * 1:nyear))
lines(1:nyear,phi1,lwd=2,col='green') # add survival from first class
lines(1:nyear,phi2,lwd=2,col='green') # add survival from second class
lines(1:nyear,expit(mean(no_het_with_age$BUGSoutput$sims.matrix[,'alpha'])+me
an(no_het_with_age$BUGSoutput$sims.matrix[,'beta'])*(1:nyear)),lwd=2,col='blu
e') # add survival from age model
```



Seems like we've managed to capture the main patterns in the simulated data.