Appendix 3: Model code

This appendix contains the model code, written and implemented with the software `nimble` and implemented in parallel with the `parallel` package. See Appendix 4 for more generic, modular model code that closely follows the model description in the main text without the extra data structures required to account for the heterogeneous data used to fit the model.

First data will be read in:

```
# read in time series data
n_C <- readRDS("data/model_data/counts.rds")
n_R <- readRDS("data/model_data/n_cap.rds")

# read in time series constants
index <- readRDS("data/model_data/index.rds")
D <- readRDS("data/model_data/index_frac.rds")
totalt <- readRDS("data/model_data/totalt.rds")
totalo <- readRDS("data/model_data/totalo.rds")
soak_days <- readRDS("data/model_data/soak_days.rds")
m_index <- readRDS("data/model_data/m_index.rds")
f_index <- readRDS("data/model_data/f_index.rds")
s_index <- readRDS("data/model_data/s_index.rds")
recruit_intro <- readRDS("data/model_data/recruit_intro.rds")
additionalt <- readRDS("data/model_data/additionalt.rds")

# read in mark-recapture data
n1_mc <- readRDS("data/model_data/n1_mc.rds")
m2_mc <- readRDS("data/model_data/m2_mc.rds")

# read in mark-recapture constants
D_mc <- readRDS("data/model_data/mc_index.rds")
totalo_mc <- readRDS("data/model_data/totalo_mc.rds")
soak_days_mc <- readRDS("data/model_data/soak_days_mc.rds")
f_index_mc <- readRDS("data/model_data/f_index_mc.rds")
s_index_mc <- readRDS("data/model_data/s_index_mc.rds")
m_index_mc <- readRDS("data/model_data/m_index_mc.rds")

# read in IPM constants
b <- readRDS("data/model_data/b.rds")
y <- readRDS("data/model_data/y.rds")
```

Then the `nimble` model code:

```
# Write model code
model_code <- nimbleCode({

  ##################
  ##################
```

```
# Process model #
#################
#################

############################
# Integral projection model #
############################

# project to first observed time period - year 2-4
for (i in 2:n_year) {
  N[1, i, 1:n_size] <- get_kernel(yinf, gk, sigma_G, C,
                                  ts, 0, D[1, i], n_size, pi,
                                  y[1:n_size], lower[1:n_size],
                                  upper[1:n_size], S[1, i, 1:n_size]) %*%
    N_overwinter[i - 1, 1:n_size]

  ## natural survival
  S[1, i, 1:n_size] <- survival(alpha, beta[i], y[1:n_size], 0, D[1, i])
}

# intra-annual change
for (i in 1:n_year) {
  for (t in 1:(totalt[i]-1)) {

    # Equations 1 - 2
    ## project
    N[t + 1, i, 1:n_size] <- get_kernel(yinf, gk, sigma_G,
                                        C, ts, D[t, i],
                                        D[t + 1, i], n_size, pi,
                                        y[1:n_size], lower[1:n_size],
                                        upper[1:n_size],
                                        S[t + 1, i, 1:n_size])  %*%
      (N[t, i, 1:n_size] - n_R[t, i, 1:n_size]) +
      recruit_intro[t, i] * R[i, 1:n_size] # introduce recruits, t = 6

    # Equation 7
    ## natural survival
    S[t + 1, i, 1:n_size] <- survival(alpha, beta[i], y[1:n_size], D[t, i],
                                      D[t + 1, i])

  }
}

# Equation 8
## year-specific natural mortality (process error, includes mark-recapture)
for (i in 1:(n_year + 1)) {
  beta[i] ~ dgamma(beta_alpha, beta_theta)
}

# project all years to the same intra-annual end point by
# applying the growth kernel and natural mortality
for (i in 1:n_year_short) {
  for (t in additionalt[year_short[i], 1]:additionalt[year_short[i], 2]) {
```

```
    # Equations 1 - 2
    ## project
    N[t + 1, year_short[i], 1:n_size] <- get_kernel(yinf, gk, sigma_G,
                                                    C, ts,
                                                    D[t, year_short[i]],
                                                    D[t + 1, year_short[i]],
                                                    n_size, pi,
                                                    y[1:n_size],
                                                    lower[1:n_size],
                                                    upper[1:n_size],
                                                    S[t + 1, year_short[i],
                                                      1:n_size]) %*%
      N[t, year_short[i], 1:n_size]

    S[t + 1, year_short[i], 1:n_size] <- survival(alpha, beta[i], y[1:n_size],
                                                  D[t, year_short[i]],
                                                  D[t + 1, year_short[i]])
  }
}


## inter-annual change
# project to new year with seasonal growth
for (i in 1:(n_year - 1)) {
  wgrowth_N[i, 1:n_size] <- get_kernel(yinf, gk, sigma_G, C, ts,
                                       max_D, 1,
                                       n_size, pi, y[1:n_size],
                                       lower[1:n_size], upper[1:n_size],
                                       ones[1:n_size]) %*%
    N[additionalt[i, 2] + 1, i, 1:n_size]

  # total density after seasonal growth
  wgrowth_N_sum[i] <- sum(wgrowth_N[i, 1:n_size])

  # size- and density-dependent overwinter mortality
  for (k in 1:n_size) {
    N_overwinter[i, k] ~ dbinom(size = round(wgrowth_N[i, k]),
                                prob = S_o[i, k])
  }

  # Equation 11
  ## size- and density-dependent overwinter survival
  S_o[i, 1:n_size] <- overwinter_survival(alpha_o[i],
                                          wgrowth_N_sum[i],
                                          y[1:n_size])

  # Equation 12
  ## year-specific intensity of overwinter mortality
  alpha_o[i] ~ dgamma(alpha_o_alpha, alpha_o_theta)
}

#########################################################
# Initial population density and annual recruitment #
```

```
#######################################################

# Equation 13
## get initial size-structured abundance of adults - year 1
N_init[1:n_size] <- get_init_adult(mu_A, sigma_A,
                                   lower[1:n_size], upper[1:n_size], lambda_A)

# project to first observed time period - year 1
N[1, 1, 1:n_size] <- get_kernel(yinf, gk, sigma_G, C,
                                ts, 0, D[1, 1], n_size, pi,
                                y[1:n_size], lower[1:n_size],
                                upper[1:n_size], S[1, 1, 1:n_size]) %*%
  N_init[1:n_size]

## natural survival
S[1, 1, 1:n_size] <- survival(alpha, beta[1], y[1:n_size], 0, D[1, 1])

# Equation 14
## annual abundance of recruits
for (m in 1:n_year) {
  lambda_R[m] ~ T(dnorm(mu_lambda, sd = sigma_lambda), 0, Inf)
}

# Equation 15
## annual size distribution of recruits
R[1:n_year, 1:n_size] <- get_init_recruits(mu_R, sigma_R, lower[1:n_size],
                                           upper[1:n_size],
                                           lambda_R[1:n_year], n_year,
                                           n_size)


######################
######################
# Observation model #
######################
######################

for (i in 1:n_year) {
  for (t in 1:totalt[i]) {
    for (k in 1:n_size) {

      # Equation 16
      ## binomial distribution with total crabs removed, n_R
      n_R[t, i, k] ~ dbinom(size = round(N[t, i, k]), prob = p[t, i, k])

      # Equation 17
      ## dirichlet-multinomial mixture, conditional probability of capture
      alpha_D[t, 1:totalo[t, i], i, k] <- p_C[t, 1:totalo[t, i],
                                              i, k] * n_p_dir
      n_C[t, 1:totalo[t, i],
          i, k] ~ ddirchmulti(alpha = alpha_D[t, 1:totalo[t, i], i, k],
                              size = n_R[t, i, k])
    }
```

```
    # Equations 18 - 20
    ## calculate hazard rate
    hazard[t, 1:totalo[t, i], i, 1:n_size] <- calc_hazard(
      totalo[t, i], n_size, h_F_max, h_F_k, h_F_0, h_S_max, h_S_k, h_S_0,
      h_M_max, h_M_A, h_M_sigma, f_index[t, 1:totalo[t, i], i],
      s_index[t, 1:totalo[t, i], i], m_index[t, 1:totalo[t, i], i],
      soak_days[t, 1:totalo[t, i], i], y[1:n_size]
    )

    # Equation 21
    ## total capture probability
    p[t, i, 1:n_size] <- calc_prob(totalo[t, i], n_size,
                                   hazard[t, 1:totalo[t, i], i, 1:n_size])

    # mean conditional probability of capture
    p_C[t, 1:totalo[t, i],
        i, 1:n_size] <- calc_cond_prob(totalo[t, i], n_size,
                                       hazard[t, 1:totalo[t, i],
                                              i, 1:n_size])

  }
}

################################
################################
# Integrated population model #
################################
################################

########################
# Mark-recapture data #
########################

# Equation 24
## draw recaptured samples, m2, from projected marked samples, n1_project
for (k in 1:n_size) {
  m2[k] ~ dbinom(size = round(n1_project[k]), prob = p_mc[k])
}

# Equation 25
## total capture probability with mark-recapture data
p_mc[1:n_size] <- calc_prob(totalo_mc, n_size,
                            hazard_mc[1:totalo_mc, 1:n_size])

# size-dependent hazard rates with mark-recapture data
hazard_mc[1:totalo_mc, 1:n_size] <- calc_hazard(
  totalo_mc, n_size, h_F_max, h_F_k, h_F_0, h_S_max, h_S_k, h_S_0, h_M_max,
  h_M_A, h_M_sigma, f_index_mc[1:totalo_mc], s_index_mc[1:totalo_mc],
  m_index_mc[1:totalo_mc], soak_days_mc[1:totalo_mc], y[1:n_size]
)

# Equation 26
## apply kernel from time of marking to time of recapture
```

```
n1_project[1:n_size] <- get_kernel(yinf, gk, sigma_G, C, ts, D_mc[1], D_mc[2],
                                   n_size, pi, y[1:n_size], lower[1:n_size],
                                   upper[1:n_size], S_mc[1:n_size]) %*%
  n1[1:n_size]

S_mc[1:n_size] <- survival(alpha, beta[5], y[1:n_size], D_mc[1], D_mc[2])

#################
# Growth model #
#################

# asymptotic size -- (from seasonal growth posterior)
yinf ~ dnorm(95.4, sd = 5.52)
# growth rate -- (from seasonal growth posterior)
gk ~ dnorm(0.67, sd = 0.1)
# amplitude of growth oscillations -- (from seasonal growth posterior)
C ~ dnorm(0.79, sd = 0.11)
# inflection point of growth oscillations -- (from seasonal growth posterior)
ts ~ dnorm(-0.642, sd = 0.027)


#################
#################
# Vague priors #
#################
#################

########################
# Prior distributions #
########################

##
# size selectivity parameters
##

# minnow max. hazard rate
h_M_max ~ dunif(0, 0.1)
# minnow max. size of capture
h_M_A ~ dunif(35, 60)
# minnow sigma of gaussian size selectivity curve
h_M_sigma ~ dunif(3, 10)
# fukui max. hazard rate
h_F_max ~ dunif(0, 0.1)
# fukui k of logistic size selectivity curve
h_F_k ~ dunif(0.1, 1.5)
# fukui midpoint of logistic size selectivity curve
h_F_0 ~ dunif(30, 100)
# shrimp max. hazard rate
h_S_max ~ dunif(0, 0.1)
# shrimp k of logistic size selectivity curve
h_S_k ~ dunif(0.1, 1.5)
# shrimp midpoint of logistic size selectivity curve
h_S_0 ~ dunif(30, 100)
```

```
  ##
  # IPM - natural mortality
  ##

  # gamma distributions shape for instantaneous intensity of mortality
  beta_alpha ~ dunif(0, 50)
  # gamma distributions rate for instantaneous intensity of mortality
  beta_theta ~ dunif(0, 150)
  # size-dependent overwinter natural mortality, shared across all sites
  alpha ~ dunif(0, 10000)
  # gamma distribution shape - instantaneous probability of overwinter mortality
  alpha_o_alpha ~ dunif(0, 50)
  # gamma distribution rate - instantaneous probability of overwinter mortality
  alpha_o_theta ~ dunif(0, 150)

  ##
  # IPM - growth
  ##

  # growth error
  sigma_G ~ dunif(0.01, 4)

  ##
  # observation process
  ##
  # dirichlet multinomial (overdispersion in count data)
  ro_dir ~ dbeta(1, 1)
  n_p_dir <- (1 - ro_dir) / ro_dir

  ##
  # initial population density and annual recruitment
  ##

  # initial adult size (mean and sd)
  mu_A ~ dunif(3.25, 4.5)
  sigma_A ~ dunif(0.1, 1)

  # initial recruit size (mean and sd)
  mu_R ~ dunif(1, 25)
  sigma_R ~ dunif(0.01, 20)

  # abundance of recruits (mean and sd)
  mu_lambda ~ dunif(1, 1000000)
  sigma_lambda ~ dunif(0, 10000)

  # abundance of adults in year 1
  lambda_A ~ dunif(1, 1000000)


})
```

Then declare the model data and constants, as well as generate initial values for the MCMC:

```
# bundle up data and constants
constants <- list(
  # number of years
  n_year = dim(n_C)[3],
  # number of time periods in each year
  totalt = totalt,
  # number of trap obs in year i, time t
  totalo = totalo,
  # time periods to project all years to the same intra-annual end point
  additionalt = additionalt,
  # year indices where intra-annual end point > t of last observation
  year_short = which(apply(index, 2, max, na.rm = TRUE) <
                       max(index, na.rm = TRUE)),
  # number of years where intra-annual end point > t of last observation
  n_year_short = length(which(apply(index, 2, max, na.rm = TRUE) <
                                max(index, na.rm = TRUE))),
  # number of sizes in IPM mesh
  n_size = length(y),
  # upper bounds in IPM mesh
  upper = b[2:length(b)],
  # lower bounds in IPM mesh
  lower = b[1:length(y)],
  # binary indicator of fukui traps in time t, obs j, year i
  f_index = f_index,
  # binary indicator of minnow traps in time t, obs j, year i
  m_index = m_index,
  # binary indicator of shrimp traps in time t, obs j, year i
  s_index = s_index,
  # midpoints in IPM mesh
  y = y,
  # calendar date indices (fraction of year) in time t, year i
  D = D,
  # maximum calendar date index
  max_D = max(D, na.rm = TRUE),
  # number of soak days for each trap at time t, trap j, year i
  soak_days = soak_days,
  # data structure to introduce recruits into the model at t = 6
  recruit_intro = recruit_intro,
  # total number of trap observations in mark-recapture dataset
  totalo_mc = totalo_mc,
  # binary indicator of fukui traps in trap j of mark-recapture dataset
  f_index_mc = f_index_mc,
  # binary indicator of shrimp traps in trap j of mark-recapture dataset
  s_index_mc = s_index_mc,
  # binary indicator of minnow traps in trap j of mark-recapture dataset
  m_index_mc = m_index_mc,
  # number of soak days in trap j of mark-recapture dataset
  soak_days_mc = soak_days_mc,
  # calendar date indices (fraction of year) of mark-recapture dataset
  D_mc = D_mc,
  pi = pi,
  ones = rep(1, length(y))
)
```

```
data <- list(
  n_R = n_R, # total captured within at time t, year i, size y
  n_C = n_C, # n_C at time t, trap j, year i, size y
  n1 = n1_mc, # count of marked crabs of size y in mark-recapture dataset
  m2 = m2_mc # count of recaptured crabs of size y in mark-recapture dataset
)

# create N_overwinter initial values
N_overwinter <- matrix(NA, nrow = dim(n_C)[3] - 1, ncol = length(y))
adult_mean <- c(log(75), log(80), log(82))
mean_recruit <- c(log(55), log(52), log(52))
lambda_A <- c(500, 400, 100)
lambda_R <- c(300, 75, 400)
adult_sd <- 0.08
recruit_sd <- 0.1
for (i in 1:(dim(n_C)[3] - 1)) {
  prob_r <- dlnorm(y, mean_recruit[i], recruit_sd) /
    sum(dlnorm(y, mean_recruit[i], recruit_sd))
  prob_a <- dlnorm(y, adult_mean[i], adult_sd) /
    sum(dlnorm(y, adult_mean[i], adult_sd))
  N_overwinter[i, ] <- round(prob_a * lambda_A[i] + prob_r * lambda_R[i])
}

# initial values
inits <- function() {
  list(
    h_M_max = runif(1, 0.0001, 0.0008), h_M_A = 44, h_M_sigma = 6.67,
    h_F_max = runif(1, 0.0001, 0.0008), h_F_k = 0.2, h_F_0 = 45,
    h_S_max = runif(1, 0.001, 0.005), h_S_k = 0.2, h_S_0 = 45,
    ro_dir = 0.01, beta_alpha = 2, beta_theta = 1, alpha = 0.1,
    alpha_o_alpha = 2, alpha_o_theta = 1, gk = 1, yinf = 85,
    C = 0.79, ts = -0.64, sigma_G = 2.5, sigma_R = 1,
    mu_R = 20, mu_A = 4, sigma_A = 0.2,
    lambda_A = 1800, lambda_R = c(1000, 100, 1000, 100), mu_lambda = 500,
    sigma_lambda = 100, beta = rep(0.001, 5),
    alpha_o = c(0.0308, 0.00669, 0.0346), N_overwinter = N_overwinter
  )
}
```

Run the MCMC in parallel, including assigning the relevant nimble functions to the parallel nodes:

```
########################
# run MCMC in parallel #
########################

cl <- makeCluster(4)

set.seed(10120)

clusterExport(cl, c("model_code", "inits", "data", "constants", "N_overwinter"))

# Create a function with all the needed code
out <- clusterEvalQ(cl, {
```

```r
library(nimble)
library(coda)
library(expm)

# define dirichlet multinomial mixture
# pdf
ddirchmulti <- nimbleFunction (
  run = function(x = double(1), alpha = double(1), size = double(0),
                 log = integer(0, default = 0)) {
    returnType(double(0))
    logProb <- lgamma(size + 1) - sum(lgamma(x + 1)) + lgamma(sum(alpha)) -
      sum(lgamma(alpha)) + sum(lgamma(alpha + x)) -
      lgamma(sum(alpha) + size)
    if (log) return(logProb)
    else return(exp(logProb))
  })
# distribution number generator
rdirchmulti <- nimbleFunction (
  run = function(n = integer(0), alpha = double(1), size = double(0)) {
    returnType(double(1))
    if (n != 1) print("rdirchmulti only allows n = 1; using n = 1.")
    p <- rdirch(1, alpha)
    return(rmulti(1, size = size, prob = p))
  })

assign("ddirchmulti", ddirchmulti, .GlobalEnv)
assign("rdirchmulti", rdirchmulti, .GlobalEnv)

# size selective hazard rate of trap type minnow - bell-shaped curve
size_sel_norm <- nimbleFunction (
  # input and output types
  run = function(pmax = double(0), xmax = double(0), sigma = double(0),
                 y = double(1))
  {
    returnType(double(1))

    vector <- pmax * exp(-(y - xmax) ^ 2 / (2 * sigma ^ 2))

    return(vector)
  }
)
assign("size_sel_norm", size_sel_norm, envir = .GlobalEnv)

# size selective hazard rate of trap type fukui and shrimp - logistic
size_sel_log <- nimbleFunction (
  #input and output types
  run = function(pmax = double(0), k = double(0), midpoint = double(0),
                 y = double(1))
  {
    returnType(double(1))

    vector <- pmax / (1 + exp(-k * (y - midpoint)))
```

```
    return(vector)
  }
)
assign("size_sel_log", size_sel_log, envir = .GlobalEnv)


# calculate trap hazard rate of obs j, based on trap type and soak days
calc_hazard <- nimbleFunction (
  #input and output types
  run = function(nobs = double(0), n_size = double(0), h_F_max = double(0),
                 h_F_k = double(0), h_F_0 = double(0),
                 h_S_max = double(0), h_S_k = double(0),
                 h_S_0 = double(0), h_M_max = double(0),
                 h_M_A = double(0), h_M_sigma = double(0),
                 obs_ref_f = double(1), obs_ref_s = double(1),
                 obs_ref_m = double(1), soak_days = double(1),
                 y = double(1))
  {
    returnType(double(2))

    # create empty array
    array <- array(init = FALSE, dim = c(nobs, n_size))
    # loop through observations
    for (j in 1:nobs) {
      # P(capture)
      array[j, 1:n_size] <- size_sel_log(pmax = h_F_max, k = h_F_k,
                                         midpoint = h_F_0, y = y) *
        obs_ref_f[j] * soak_days[j] +
        size_sel_log(pmax = h_S_max, k = h_S_k, midpoint = h_S_0, y = y) *
        obs_ref_s[j] * soak_days[j] +
        size_sel_norm(pmax = h_M_max, xmax = h_M_A,
                      sigma = h_M_sigma, y = y) *
        obs_ref_m[j] * soak_days[j]
    }

    return(array)
  }
)
assign("calc_hazard", calc_hazard, envir = .GlobalEnv)

# calculate conditional probability of capture
calc_cond_prob <- nimbleFunction (
  #input and output types
  run = function(nobs = double(0), n_size = double(0), hazard = double(2))
  {
    returnType(double(2))

    # create empty array
    array <- array(init = FALSE, dim = c(nobs, n_size))

    # loop through sizes
    for (k in 1:n_size) {
      #P(capture in trap j | captured at all)
```

```
      array[1:nobs, k] <- hazard[1:nobs, k] / sum(hazard[1:nobs, k])
    }
    return(array)
  }
)
assign("calc_cond_prob", calc_cond_prob, envir = .GlobalEnv)


# calculate total capture probability
calc_prob <- nimbleFunction (
  #input and output types
  run = function(nobs = double(0), n_size = double(0), hazard = double(2))
  {
    returnType(double(1))

    # create empty array
    p <- rep(NA, n_size)

    # loop through sizes
    for (k in 1:n_size) {
      # 1 - exp(-P(captured at all))
      p[k] <- 1 - exp(-sum(hazard[1:nobs, k]))
    }
    return(p)
  }
)
assign("calc_prob", calc_prob, envir = .GlobalEnv)

# function for seasonal growth kernel -- biweekly time step
get_kernel <- nimbleFunction (
  # input and output types
  run = function(yinf = double(0), k = double(0),
                 sigma_G = double(0), C = double(0),
                 ts = double(0), t1 = double(0), t2 = double(0),
                 n_size = double(0), pi = double(0), y = double(1),
                 lower = double(1), upper = double(1), S = double(1))
  {
    returnType(double(2))

    # create empty array
    array <- matrix(NA, ncol = n_size, nrow = n_size)

    if(!is.na(t2) & t2 == 0) {
      array <- diag(n_size)
    } else {
      # season adjusted params
      S_t <- (C * k / (2 * pi)) * sin(2 * pi * (t2 - (1 + ts)))
      S_t0 <- (C * k / (2 * pi)) * sin(2 * pi * (t1 - (1 + ts)))

      # p(y"|y)
      for (i in 1:n_size) {
        increment <- (yinf - y[i]) *
          (1 - exp(-k * (t2 - t1) - S_t + S_t0))
```

```
        mean <- y[i] + increment
        array[1:n_size, i] <- (pnorm(upper, mean, sd = sigma_G) -
                               pnorm(lower, mean, sd = sigma_G))
      }

      # normalize and apply natural mortality
      for (i in 1:n_size) {
        array[, i] <- array[, i] / sum(array[, i]) * S
      }
    }
    return(array)
  }
)
assign("get_kernel", get_kernel, envir = .GlobalEnv)

# initial size distribution of adults - year 1
get_init_adult <- nimbleFunction (

  run = function(mu_A = double(0), sigma_A = double(0),
                 lower = double(1), upper = double(1),
                 lambda_A = double(0))
  {
    returnType(double(1))

    prop_adult <- plnorm(q = upper, meanlog = mu_A, sdlog = sigma_A) -
      plnorm(q = lower, meanlog = mu_A, sdlog = sigma_A)

    # get initial size-structured abundance of adults
    out <- prop_adult * lambda_A

    return(out)
  }
)
assign("get_init_adult", get_init_adult, envir = .GlobalEnv)

# annual size distributions of recruit
get_init_recruits <- nimbleFunction (

  run = function(mu_R = double(0), sigma_R = double(0),
                 lower = double(1), upper = double(1),
                 lambda_R = double(1), n_year = double(0),
                 n_size = double(0))
  {
    returnType(double(2))

    # create empty array
    out <- matrix(NA, ncol = n_size, nrow = n_year)

    # moment match from normal to gamma
    var <- sigma_R ^ 2
    shape <- mu_R ^ 2 / var
    rate <- mu_R / var
    prop_recruit <- pgamma(q = upper, shape = shape, rate = rate) -
```

```
      pgamma(q = lower, shape = shape, rate = rate)

    # get initial size-structured abundance of recruits
    for (i in 1:n_year) {
      out[i, ] <- prop_recruit[1:n_size] * lambda_R[i]
    }

    return(out)
  }
)
assign("get_init_recruits", get_init_recruits, envir = .GlobalEnv)

# natural (non-winter) survival
survival <- nimbleFunction (

  run = function(alpha = double(0), beta = double(0), y = double(1),
                 t1 = double(0), t2 = double(0))
  {
    returnType(double(1))

    # number of biweeks
    deltat <- round((t2 - t1) * (52.1429 / 2))

    # get survival rate
    out <- exp(-deltat * (beta + alpha / y ^ 2))

    return(out)
  }
)
assign("survival", survival, envir = .GlobalEnv)

# density- and size-dependent overwinter survival
overwinter_survival <- nimbleFunction (

  run = function(alpha_o = double(0), N_sum = double(0),
                 y = double(1))
  {
    returnType(double(1))

    # get probability of survival
    out <- exp(-(alpha_o * N_sum / y ^ 2))

    return(out)
  }
)
assign("overwinter_survival", overwinter_survival, envir = .GlobalEnv)


# build model
myModel <- nimbleModel(code = model_code,
                       data = data,
                       constants = constants,
                       inits = inits())
```

```
  # build the MCMC
  mcmcConf_myModel <- configureMCMC(
    myModel,
    monitors = c("h_M_max", "h_M_A", "h_M_sigma", "h_F_max",
                 "h_F_k", "h_F_0", "h_S_max", "h_S_k",
                 "h_S_0", "beta_alpha", "beta_theta", "gk",
                 "yinf", "C", "ts", "sigma_G",
                 "sigma_R", "mu_R", "mu_A",
                 "sigma_A", "mu_lambda", "sigma_lambda",
                 "lambda_R", "lambda_A", "beta", "alpha_o",
                 "alpha", "beta_alpha", "beta_theta",
                 "alpha_o_alpha", "alpha_o_theta", "N_overwinter",
                 "wgrowth_N_sum", "ro_dir"),
    useConjugacy = FALSE, enableWAIC = TRUE)

  # add block sampler for nmort params
  mcmcConf_myModel$removeSamplers(c("beta_alpha", "beta_theta"))
  mcmcConf_myModel$addSampler(c("beta_alpha", "beta_theta"),
                              type = "RW_block")
  mcmcConf_myModel$removeSamplers(c("alpha_o_alpha", "alpha_o_theta"))
  mcmcConf_myModel$addSampler(c("alpha_o_alpha", "alpha_o_theta"),
                            type = "RW_block")

  # build MCMC
  myMCMC <- buildMCMC(mcmcConf_myModel)

  # compile the model and MCMC
  CmyModel <- compileNimble(myModel)

  # compile the MCMC
  cmodel_mcmc <- compileNimble(myMCMC, project = myModel)

  # run MCMC
  cmodel_mcmc$run(100000, thin = 10,
                  reset = FALSE)

  samples <- as.mcmc(as.matrix(cmodel_mcmc$mvSamples))

  return(samples)
})
```

Finally, save the samples and stop the cluster:

```
# save samples
saveRDS(out, "savedsamples_IPM.rds")

stopCluster(cl)
```