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.

First data will be read in:

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

# read in time series constants
index <- readRDS("data/model_data/index.rds")
index_frac <- 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
mc_index <- 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({

  ########################
  # Prior distributions #
  ########################
```

```
##
# size selectivity parameters
##

# minnow max. probability of capture
trapm_pmax ~ dunif(0, 0.1)
# minnow max. size of capture
trapm_xmax ~ dunif(35, 60)
# minnow sigma of gaussian size selectivity curve
trapm_sigma ~ dunif(3, 10)
# fukui max. probability of capture
trapf_pmax ~ dunif(0, 0.1)
# fukui k of logistic size selectivity curve
trapf_k ~ dunif(0.1, 1.5)
# fukui midpoint of logistic size selectivity curve
trapf_midpoint ~ dunif(30, 100)
# shrimp max. probability of capture
traps_pmax ~ dunif(0, 0.1)
# shrimp k of logistic size selectivity curve
traps_k ~ dunif(0.1, 1.5)
# shrimp midpoint of logistic size selectivity curve
traps_midpoint ~ dunif(30, 100)

##
# IPM - natural mortality
##

# gamma distributions shape for instantaneous probability of mortality
nmort_shape_s ~ dunif(0, 50)
# gamma distributions rate for instantaneous probability of mortality
nmort_rate_s ~ dunif(0, 150)
# size-dependent overwinter natural mortality, shared across all sites
nmort_size ~ dunif(0, 10000)
# gamma distribution shape - instantaneous probability of overwinter mortality
wnmort_shape_s ~ dunif(0, 50)
# gamma distribution rate - instantaneous probability of overwinter mortality
wnmort_rate_s ~ dunif(0, 150)

##
# IPM - growth
##

# asymptotic size -- (from seasonal growth posterior)
growth_xinf ~ dnorm(95.4, sd = 5.52)
# growth rate -- (from seasonal growth posterior)
growth_k ~ dnorm(0.67, sd = 0.1)
# amplitude of growth oscillations -- (from seasonal growth posterior)
growth_C ~ dnorm(0.79, sd = 0.11)
# inflection point of growth oscillations -- (from seasonal growth posterior)
growth_ts ~ dnorm(-0.642, sd = 0.027)
# growth error
growth_sd ~ 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)
init_lmean_adult ~ dunif(3.25, 4.5)
init_lsd_adult ~ dunif(0.1, 1)

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

# abundance of recruits (mean and sd)
N_mu_recruit ~ dunif(1, 1000000)
N_sd_recruit ~ dunif(0, 10000)
for (m in 1:n_year) {
  N_recruit[m] ~ T(dnorm(N_mu_recruit, sd = N_sd_recruit), 0, Inf)
}

# adundance of adults in year 1
N_adult ~ dunif(1, 1000000)


########################
# mark-recapture data #
########################

# size-dependent hazard rates with mark-recapture data
hazard_mc[1:totalo_mc,
          1:n_size] <- calc_hazard(totalo_mc, n_size, trapf_pmax, trapf_k,
                                   trapf_midpoint, traps_pmax, traps_k,
                                   traps_midpoint, trapm_pmax, trapm_xmax,
                                   trapm_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])

# 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])

# apply growth kernel from time of marking to time of recapture
n1_project[1:n_size] <- gkernel(growth_xinf, growth_k,
                                growth_sd, growth_C, growth_ts,
                                mc_index[1], mc_index[2],
                                n_size, pi, y[1:n_size],
```

```
                                lower[1:n_size], upper[1:n_size]) %*%
  n1[1:n_size]

# draw recaptured samples, m2, from projected marked samples, n1_project,
# after applying natural mortality, nmortality_s
for (k in 1:n_size) {
  m2[k] ~ dbinom(size = round(n1_project[k] *
                                exp(-(nmortality_s[5] +
                                        nmort_size / y[k] ^ 2))),
                 prob = p_mc[k])
}


##########################################
# time_series data - intra-annual change #
##########################################

# calculate proportions in each size class of recruits
prop_recruit[1:n_size] <- init_sizes_gamma(init_mean_recruit,
                                           init_sd_r, lower[1:n_size],
                                           upper[1:n_size])

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

}

# calculate proportions in each size class of adults - year 1
prop_adult[1:n_size] <- init_sizes_lnorm(init_lmean_adult, init_lsd_adult,
                                         lower[1:n_size], upper[1:n_size])

# get initial size-structured abundance of recruits - year 1
N_init[1:n_size] <- prop_adult[1:n_size] * N_adult

# project to first observed time period - year 1
N[1, 1, 1:n_size] <- gkernel(growth_xinf, growth_k, growth_sd, growth_C,
                             growth_ts, 0, index_frac[1, 1], n_size, pi,
                             y[1:n_size], lower[1:n_size],
                             upper[1:n_size]) %*% N_init[1:n_size]

# project to first observed time period - year 2:4
for (i in 2:n_year) {
  N[1, i, 1:n_size] <- gkernel(growth_xinf, growth_k, growth_sd, growth_C,
                               growth_ts, 0, index_frac[1, i], n_size, pi,
                               y[1:n_size], lower[1:n_size],
                               upper[1:n_size]) %*%
    N_overwinter[i - 1, 1:n_size]
}

# get observations for t = 1
for (i in 1:n_year) {
```

```
  # calculate hazard rate at t = 1
  hazard[1, 1:totalo[1, i],
         i, 1:n_size] <- calc_hazard(totalo[1, i], n_size, trapf_pmax,
                                     trapf_k, trapf_midpoint, traps_pmax,
                                     traps_k, traps_midpoint, trapm_pmax,
                                     trapm_xmax, trapm_sigma,
                                     f_index[1, 1:totalo[1, i], i],
                                     s_index[1, 1:totalo[1, i], i],
                                     m_index[1, 1:totalo[1, i], i],
                                     soak_days[1, 1:totalo[1, i], i],
                                     y[1:n_size])

  # conditional probability of capture at t = 1
  pic_trap[1, 1:totalo[1, i],
           i, 1:n_size] <- calc_cond_prob(totalo[1, i], n_size,
                                          hazard[1, 1:totalo[1, i],
                                                 i, 1:n_size])
  # total capture probability at t = 1
  p[1, i, 1:n_size] <- calc_prob(totalo[1, i], n_size,
                                 hazard[1, 1:totalo[1, i], i, 1:n_size])

  for (k in 1:n_size) {

    # dirichlet-multinomial mixture
    # conditional probability of capture at t = 1
    alpha_dir[1, 1:totalo[1, i], i, k] <- pic_trap[1, 1:totalo[1, i],
                                                   i, k] * n_p_dir
    counts[1, 1:totalo[1, i],
           i, k] ~ ddirchmulti(alpha = alpha_dir[1, 1:totalo[1, i], i, k],
                               size = n_cap[1, i, k])

    # binomial distribution with total crabs removed, n_cap
    n_cap[1, i, k] ~ dbinom(size = round(N[1, i, k]), prob = p[1, i, k])
  }
}

# t = 2+
for (i in 1:n_year) {
  for (t in 2:totalt[i]) {

    # calculate hazard rate, t = 2+
    hazard[t, 1:totalo[t, i],
           i, 1:n_size] <- calc_hazard(totalo[t, i], n_size, trapf_pmax,
                                       trapf_k, trapf_midpoint, traps_pmax,
                                       traps_k, traps_midpoint, trapm_pmax,
                                       trapm_xmax, trapm_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])

    # conditional probability of capture at t = 2+
```

```
    pic_trap[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])
    # total capture probability at t = 2+
    p[t, i, 1:n_size] <- calc_prob(totalo[t, i], n_size,
                                   hazard[t, 1:totalo[t, i], i, 1:n_size])

    for (k in 1:n_size) {

      # dirichlet-multinomial mixture
      # conditional probability of capture at t = 2+
      alpha_dir[t, 1:totalo[t, i],
                i, k] <- pic_trap[t, 1:totalo[t, i], i, k] * n_p_dir
      counts[t, 1:totalo[t, i],
             i, k] ~ ddirchmulti(alpha = alpha_dir[t, 1:totalo[t, i], i, k],
                                 size = n_cap[t, i, k])

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

      # apply natural morality, nmortality_s (with process error)
      N[t, i, k] <- next_N[t - 1, i, k] *
        exp(-(nmortality_s[i] + nmort_size / y[k] ^ 2)) +
        recruit_intro[t, i] * N_init_recruit[i, k] # introduce recruits
    }

    # apply growth kernel to project to next time period
    next_N[t - 1, i, 1:n_size] <- gkernel(growth_xinf, growth_k, growth_sd,
                                          growth_C, growth_ts,
                                          index_frac[t - 1, i],
                                          index_frac[t, i], n_size, pi,
                                          y[1:n_size], lower[1:n_size],
                                          upper[1:n_size]) %*%
      (N[t - 1, i, 1:n_size] - n_cap[t - 1, i, 1:n_size])
  }
}

# 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]) {

    # project to next time period with growth kernel
    next_N[t, year_short[i],
           1:n_size] <- gkernel(growth_xinf, growth_k, growth_sd, growth_C,
                                growth_ts, index_frac[t, year_short[i]],
                                index_frac[t + 1, year_short[i]], n_size, pi,
                                y[1:n_size], lower[1:n_size],
                                upper[1:n_size]) %*%
      N[t, year_short[i], 1:n_size]

    for (k in 1:n_size) {
```

```
        # apply natural mortality, nmortality_s (with process error)
        N[t + 1, year_short[i], k] <- next_N[t, year_short[i], k] *
          exp(-(nmortality_s[year_short[i]] + nmort_size / y[k] ^ 2))
      }
    }
  }

  # gamma distributed natural mortality, year-specific process error
  for (m in 1:(n_year + 1)) {
    nmortality_s[m] ~ dgamma(nmort_shape_s, nmort_rate_s)
  }

  ########################################
  # time_series data - inter-annual change #
  ########################################

  # gamma distributed overwinter mortality, year-specific process error
  for (m in 1:(n_year - 1)) {
    wnmortality_s[m] ~ dgamma(wnmort_shape_s, wnmort_rate_s)
  }

  # project to new year with seasonal growth & size-dependent natural mortality
  for (i in 1:(n_year - 1)) {
    wgrowth_N[i, 1:n_size] <- gkernel(growth_xinf, growth_k,
                                      growth_sd, growth_C, growth_ts,
                                      max_index_frac, 1,
                                      n_size, pi, y[1:n_size],
                                      lower[1:n_size],
                                      upper[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 = exp(-(wnmortality_s[i] *
                                               wgrowth_N_sum[i] / y[k] ^ 2)))
    }
  }
})
```

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(counts)[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
  index_frac = index_frac,
  # maximum calendar date index
  max_index_frac = max(index_frac, 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
  mc_index = mc_index,
  pi = pi
)

data <- list(
  n_cap = ncap, # total captured within at time t, year i, size y
  counts = counts, # counts 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(counts)[3] - 1, ncol = length(y))
mean_adult <- c(log(75), log(78.5), log(82))
mean_recruit <- c(log(55), log(55), log(52))
N_adult <- c(500, 400, 100)
N_recruit <- c(300, 75, 400)
adult_sd <- 0.08
recruit_sd <- 0.1
for (i in 1:(dim(counts)[3] - 1)) {
  prob_r <- dlnorm(y, mean_recruit[i], recruit_sd) /
    sum(dlnorm(y, mean_recruit[i], recruit_sd))
  prob_a <- dlnorm(y, mean_adult[i], adult_sd) /
    sum(dlnorm(y, mean_adult[i], adult_sd))
  N_overwinter[i, ] <- round(prob_a * N_adult[i] + prob_r * N_recruit[i])
}

# initial values
inits <- function() {
  list(
    trapm_pmax = runif(1, 0.0001, 0.0008), trapm_xmax = 44, trapm_sigma = 6.67,
    trapf_pmax = runif(1, 0.0001, 0.0008), trapf_k = 0.2, trapf_midpoint = 45,
    traps_pmax = runif(1, 0.001, 0.005), traps_k = 0.2, traps_midpoint = 45,
    ro_dir = 0.01, nmort_shape_s = 2, nmort_rate_s = 1, nmort_size = 0.1,
    wnmort_shape_s = 2, wnmort_rate_s = 1, growth_k = 1, growth_xinf = 85,
    growth_C = 0.79, growth_ts = -0.64, growth_sd = 2.5, init_sd_r = 1,
    init_mean_recruit = 18, init_lmean_adult = 4, init_lsd_adult = 0.2,
    N_adult = 1800, N_recruit = c(1000, 100, 1000, 100), N_mu_recruit = 500,
    N_sd_recruit = 100, nmortality_s = rep(0.001, 5),
    wnmortality_s = 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"))

# Create a function with all the needed code
out <- clusterEvalQ(cl, {
  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), trapf_pmax = double(0),
                 trapf_k = double(0), trapf_midpoint = double(0),
                 traps_pmax = double(0), traps_k = double(0),
                 traps_midpoint = double(0), trapm_pmax = double(0),
                 trapm_xmax = double(0), trapm_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 = trapf_pmax, k = trapf_k,
                                         midpoint = trapf_midpoint, y = y) *
        obs_ref_f[j] * soak_days[j] +
        size_sel_log(pmax = traps_pmax, k = traps_k,
                     midpoint = traps_midpoint, y = y) *
        obs_ref_s[j] * soak_days[j] +
        size_sel_norm(pmax = trapm_pmax, xmax = trapm_xmax,
                      sigma = trapm_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
gkernel <- nimbleFunction (
  # input and output types
  run = function(growth_xinf = double(0), growth_k = double(0),
                 growth_sd = double(0), growth_C = double(0),
                 growth_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))
  {
    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 <- (growth_C * growth_k / (2 * pi)) *
        sin(2 * pi * (t2 - (1 + growth_ts)))
      S_t0 <- (growth_C * growth_k / (2 * pi)) *
        sin(2 * pi * (t1 - (1 + growth_ts)))

      # p(y"|y)
      for (i in 1:n_size) {
        increment <- (growth_xinf - y[i]) *
          (1 - exp(-growth_k * (t2 - t1) - S_t + S_t0))
        mean <- y[i] + increment
        array[1:n_size, i] <- (pnorm(upper, mean, sd = growth_sd) -
                               pnorm(lower, mean, sd = growth_sd))
      }

      # normalize
```

```
      for (i in 1:n_size) {
        array[, i] <- array[, i] / sum(array[, i])
      }
    }
    return(array)
  }
)
assign("gkernel", gkernel, envir = .GlobalEnv)

# gamma distribution for initial size distribution of recruits
init_sizes_gamma <- nimbleFunction (
  #input and output types
  run = function(init_mean = double(0), init_sd = double(0),
                 lower = double(1), upper = double(1))
  {
    returnType(double(1))

    var <- init_sd ^ 2
    shape <- init_mean ^ 2 / var
    rate <- init_mean / var
    out <- pgamma(q = upper, shape = shape, rate = rate) -
      pgamma(q = lower, shape = shape, rate = rate)

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

# lognormal distribution for initial size distribution of recruits
init_sizes_lnorm <- nimbleFunction (
  #input and output types
  run = function(init_lmean = double(0), init_lsd = double(0),
                 lower = double(1), upper = double(1))
  {
    returnType(double(1))

    out <- plnorm(q = upper, meanlog = init_lmean, sdlog = init_lsd) -
      plnorm(q = lower, meanlog = init_lmean, sdlog = init_lsd)

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


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


# build the MCMC
```

```
  mcmcConf_myModel <- configureMCMC(
    myModel,
    monitors = c("trapm_pmax", "trapm_xmax", "trapm_sigma", "trapf_pmax",
                 "trapf_k", "trapf_midpoint", "traps_pmax", "traps_k",
                 "traps_midpoint", "nmort_shape_s", "nmort_rate_s", "growth_k",
                 "growth_xinf", "growth_C", "growth_ts", "growth_sd",
                 "init_sd_r", "init_mean_recruit", "init_lmean_adult",
                 "init_lsd_adult", "N_mu_recruit", "N_sd_recruit",
                 "N_recruit", "N_adult", "nmortality_s", "wnmortality_s",
                 "nmort_size", "nmort_shape_s", "nmort_rate_s",
                 "wnmort_shape_s", "wnmort_rate_s", "N_overwinter",
                 "wgrowth_N_sum", "ro_dir"),
    useConjugacy = FALSE, enableWAIC = TRUE)

  # add block sampler for nmort params
  mcmcConf_myModel$removeSamplers(c("nmort_shape_s", "nmort_rate_s"))
  mcmcConf_myModel$addSampler(c("nmort_shape_s", "nmort_rate_s"),
                              type = "RW_block")
  mcmcConf_myModel$removeSamplers(c("wnmort_shape_s", "wnmort_rate_s"))
  mcmcConf_myModel$addSampler(c("wnmort_shape_s", "wnmort_rate_s"),
                             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)
```