Appendix 4: Generic model code

Here we provide generic model code that closely follows the model description described in the main text, without the extra data structures to account for data heterogeneity (as found in Appendix 3).

Generic model code

```
model_code <- NimbleCode({

  #################
  #################
  # Process model #
  #################
  #################

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

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

  # Equation 2
  ## annual abundance of recruits
  for (m in 1:n_year) {
   lambda_R[m] ~ T(dnorm(mu_lambda, sd = sigma_lambda), 0, Inf)
  }

  # Equation 3
  ## 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)

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

  # intra-annual change
  for (i in 1:n_year) {
    for (t in 1:(Tmax-1)) {
```

```
    # Equation 5
    ## project
    N[t + 1, i, 1:n_size] <- kernel[1:n_size, 1:n_size, t] %*%
      (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 (Eq. 4)

    # Equation 6
    ## kernel
    kernel[1:n_size, 1:n_size, t] <- seasonal_growth(
      yinf, gk, sigma_G, C, ts, D[t, 1:2], n_size, pi, y[1:n_size],
      lower[1:n_size], upper[1:n_size]) * S[t, i, 1:n_size]
    # Equation 11
    ## natural survival
    S[t, i, 1:n_size] <- survival(alpha, beta[i], D[t, 1:2], y[1:n_size])
  }
}

# Equation 12
## year-specific natural mortality (process error, includes mark-recapture)
for (i in 1:(n_year + 1)) {
  beta[i] ~ dgamma(beta_alpha, beta_theta)
}

## inter-annual change
for (i in 1:(n_year - 1)) {
  for (k in 1:n_size) {

    # Equation 13
    ## overwinter survival and seasonal growth
    ## (use stochastic N node)
    N_stoch[i, k] ~ dbinom(size = seasonal_growth(yinf, gk, sigma_G, C, ts,
                                                  D[Tmax, 1:2], k, pi, y[k],
                                                  lower[k], upper[k]) %*%
                             (N[Tmax, i, k] - n_R[Tmax, i, k]),
                           prob = S_o[i, k])

  }

  # fill deterministic N node with stochastic N node
  N[1, i + 1, 1:n_size] <- N_stoch[i, 1:n_size]

  # Equation 14
  ## size- and density-dependent overwinter survival
  S_o[i, 1:n_size] <- overwinter_survival(alpha_o[i],
                                          sum(N[Tmax, i, 1:n_size]),
                                          y[1:n_size])

  # Equation 15
  ## year-specific intensity of overwinter mortality
  alpha_o[i] ~ dgamma(alpha_o_alpha, alpha_o_theta)
}

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

```
######################
# Observation model #
######################
######################

for (i in 1:n_year) {
  for (t in 1:Tmax) {
    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, i, k] <- p_C[t, 1:totalo, i, k] * n_p_dir
      n_C[t, 1:totalo, i, k] ~ ddirchmulti(alpha = alpha_D[t, 1:totalo, i, k],
                                           size = n_R[t, i, k])
    }

    # Equations 18 - 20
    ## calculate hazard rate
    hazard[t, 1:totalo, i, 1:n_size] <- calc_hazard(
      totalo, 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, i], s_index[t, 1:totalo, i],
      m_index[t, 1:totalo, i], soak_days[t, 1:totalo, i], y[1:n_size]
      )

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

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

  }
}


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

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

# Equation 22
```

```
## apply kernel from time of marking to time of recapture
n1_project[1:n_size] <- kernel_mc[1:n_size, 1:n_size] %*%
  n1[1:n_size]

kernel_mc[1:n_size, 1:n_size] <- seasonal_growth(yinf, gk, sigma_G, C, ts,
                                                 D_mc[1:2], n_size,  pi,
                                                 y[1:n_size], lower[1:n_size],
                                                 upper[1:n_size]) *
  survival(alpha, beta[5], D_mc[1:2], y[1:n_size])

# Equation 23
## 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 24
## 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[1:totalo_mc], y[1:n_size]
)

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

# incorporated hierarchically as informative priors in the unified model
# asymptotic size
yinf ~ dnorm(inf_prior[1, 1], sd = inf_prior[1, 2])
# growth rate
gk ~ dnorm(inf_prior[2, 1], sd = inf_prior[2, 2])
# amplitude of growth oscillations
C ~ dnorm(inf_prior[3, 1], sd = inf_prior[3, 2])
# inflection point of growth oscillations
ts ~ dnorm(inf_prior[4, 1], sd = inf_prior[4, 2])



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

##
# size selectivity parameters
```

```
##

# minnow max. hazard rate
h_M_max ~ dunif(priors[1, 1], priors[1, 2])
# minnow max. size of capture
h_M_A ~ dunif(priors[2, 1], priors[2, 2])
# minnow sigma of gaussian size selectivity curve
h_M_sigma ~ dunif(priors[3, 1], priors[3, 2])
# fukui max. hazard rate
h_F_max ~ dunif(priors[4, 1], priors[4, 2])
# fukui k of logistic size selectivity curve
h_F_k ~ dunif(priors[5, 1], priors[5, 2])
# fukui midpoint of logistic size selectivity curve
h_F_0 ~ dunif(priors[6, 1], priors[6, 2])
# shrimp max. hazard rate
h_S_max ~ dunif(priors[7, 1], priors[7, 2])
# shrimp k of logistic size selectivity curve
h_S_k ~ dunif(priors[8, 1], priors[8, 2])
# shrimp midpoint of logistic size selectivity curve
h_S_0 ~ dunif(priors[9, 1], priors[9, 2])

##
# IPM - natural mortality
##

# gamma distributions shape for instantaneous intensity of mortality
beta_alpha ~ dunif(priors[10, 1], priors[10, 2])
# gamma distributions rate for instantaneous intensity of mortality
beta_theta ~ dunif(priors[11, 1], priors[11, 2])
# size-dependent overwinter natural mortality, shared across all sites
alpha ~ dunif(priors[12, 1], priors[12, 2])
# gamma distribution shape - instantaneous intensity of overwinter mortality
alpha_o_alpha ~ dunif(priors[13, 1], priors[13, 2])
# gamma distribution rate - instantaneous intensity of overwinter mortality
alpha_o_theta ~ dunif(priors[14, 1], priors[14, 2])

##
# IPM - growth
##

# growth error
sigma_G ~ dunif(priors[15, 1], priors[15, 2])

##
# observation process
##
# dirichlet multinomial (overdispersion in count data)
ro_dir ~ dbeta(priors[16, 1], priors[16, 2])
n_p_dir <- (1 - ro_dir) / ro_dir

##
# initial population density and annual recruitment
##
```

```
  # initial adult size (mean and sd)
  mu_A ~ dunif(priors[17, 1], priors[17, 2])
  sigma_A ~ dunif(priors[18, 1], priors[18, 2])

  # initial recruit size (mean and sd)
  mu_R ~ dunif(priors[19, 1], priors[19, 2])
  sigma_R ~ dunif(priors[20, 1], priors[20, 2])

  # abundance of recruits (mean and sd)
  mu_lambda ~ dunif(priors[21, 1], priors[21, 2])
  sigma_lambda ~ dunif(priors[22, 1], priors[22, 2])

  # abundance of adults in year 1
  lambda_A ~ dunif(priors[23, 1], priors[23, 2])


})
```

Nimble functions

Process model functions

Initial population density of adults

This function corresponds to equation 1 in main text.

```
# 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))

    # moment match from normal to gamma
    var <- sigma_A ^ 2
    shape <- mu_A ^ 2 / var
    rate <- mu_A / var
    prop_adult <- pgamma(q = upper, shape = shape, rate = rate) -
      pgamma(q = lower, shape = shape, rate = rate)

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

    return(out)
  }
)
```

Size-structured recruitment

This function corresponds to equation 3 in main text.

```
# 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)

    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)
  }
)
```

Seasonal growth

This function corresponds to equations 7-10 in main text.

```
# function for seasonal growth kernel
seasonal_growth <- nimbleFunction (
  run = function(yinf = double(0), k = double(0),
                 sigma_G = double(0), C = double(0),
                 ts = double(0), D = double(1),
                 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)

    # season adjusted params
    S_t <- (C * k / (2 * pi)) * sin(2 * pi * (D[2] - (1 + ts)))
    S_t0 <- (C * k / (2 * pi)) * sin(2 * pi * (D[1] - (1 + ts)))
```

```
    # p(y"|y)
    for (i in 1:n_size) {
      increment <- (yinf - y[i]) *
        (1 - exp(-k * (D[2] - D[1]) - 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
    for (i in 1:n_size) {
      array[, i] <- array[, i] / sum(array[, i])
    }
  return(array)
}
)
```

Natural survival

This function corresponds to equation 11 in main text.

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

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

    # get survival rate
    out <- exp(-(D[2] - D[1]) * (beta + alpha / y ^ 2))

    return(out)
  }
)
```

Density-dependent overwinter survival

This function corresponds to equation 14 in main text.

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

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

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

    return(out)
```

```
  }
)
```

Observation model functions

Conditional multinomial observation model

This function corresponds to equations 18-20 in main text.

```
# calculate trap hazard rate of obs j, based on trap type and soak days
calc_hazard <- nimbleFunction (

  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), f_index = double(1),
                 s_index = double(1), m_index = 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) *
        f_index[j] * soak_days[j] +
        size_sel_log(pmax = h_S_max, k = h_S_k, midpoint = h_S_0, y = y) *
        s_index[j] * soak_days[j] +
        size_sel_norm(pmax = h_M_max, xmax = h_M_A, sigma = h_M_sigma, y = y) *
        m_index[j] * soak_days[j]
    }
      return(array)
  }
)
```

These functions correspond to equation 21 in main text

```
# calculate conditional probability of capture
calc_cond_prob <- nimbleFunction (
  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)
  }
)

# calculate total capture probability
calc_prob <- nimbleFunction (

  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)
  }
)
```

Size-selective hazard rates

These functions correspond to equations 18-20 in main text.

```
# bell-shaped size selective hazard rate
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)
  }
)

# logistic size selective hazard rate
size_sel_log <- nimbleFunction (
  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)
```

```
  }
)
```

Dirichlet-Multinomial mixture

These functions correspond to equation 17 in main text.

```
# 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))
  })
# random 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))
  })
```