Appendix S1

Supplementary model code for:

Simulating telemetry studies that estimate component mortality rates of imperiled juvenile salmonids

Authors: E. M. Greenheck, M. Peterson, T. Pilger, M. A. Djokic, J. Eschenroeder, T. R. Nelson

Table of Contents

Installing and loading packages, clearing the R workspace

```r
# Clear workspace
rm(list = ls())

# Required packages:
packs <- c("R2OpenBUGS", "jagsUI", "tidyverse", "purrr")

# Install packages
install.packages(packs, repos = "https://cloud.r-project.org")

# Load packages
lapply(packs, library, character.only = TRUE)
```

```
12
13 # Setting a working directory
14
15 # setwd('yourworkingdirectory')
```

Model Scenario 1: Low detection probablility (only passive acoustic telemetry array); three-state and three-observation multistate mark-recapture model (MSMR)

Section 1: Creating the simulation data

This code constructs the simulated capture histories and provides model code for a three-observation, three-state multistate mark-recapture model. Researchers interested in simulating data should modify the below code to fit their research needs starting at Section 1: Creating the simulation data. Researchers that have already collected empirical data and are looking to analyze it should load their capture histories and modify the model code from Section 2: Three-observation three-state capture-recapture model.

The capture history (CH) consists of reaches (columns) and individuals (rows), with each column indicating the state of a fish by the end of a reach (see Figure 1). The CH consists of 6 columns, including the release site where all fish are assumed alive (reach 1 or R1) and reaches 2-6 that represent "real" river stretches separated by receiver gates (see Figure 1).

These simulations vary the number of individuals (rows) within the study to illustrate the effects of increasing sample size on model estimates, relative error, and the coefficient of variation. In this example code, we will run models across five sample sizes: 50, 100, 150, 500, and 1000 individuals.

```
16 # Defining our number of reaches (columns)
17 n.reach <- 6
18 n.reach.obs <- n.reach + 1
19
20 # Defining our number of individuals to be modeled
21 ind_sims <- c(50, 100, 150, 500, 1000)  # our individual scenarios
22 nind_sims <- as.numeric(length(ind_sims))  # the number of individual scenarios
```

Setting up our state process

For this model, we set total (sum across all reaches) discrete survival (S_D) at roughly 0.5 (50%), with 0.288 total discrete predation mortality (P_D) and 0.216 total discrete unknown mortality (M_D). We provide known reach-specific values of instantaneous predation (P) and unknown mortality (M), which are used to estimate the state transition probabilities from each reach to the next. These transition probabilities produce the true reach-specific state of each fish, with all fish known alive at release (R1). The total number of elements in reach-specific estimates (P, M, etc.) need to equal n.reach in length; the data setup check code will identify any mismatches.

```
23 # Defining our number of model states
24 n.states <- 3
25
26 # Defining our estimates for predation mortality (P)
27
28 # Here, P is 0 in the first reach (it cannot actually be zero in
29 # the model code), reach 2 has high P, P is moderate in reaches 3
30 # and 5, and low in reaches 4 and 6.
31 P <- c(1e-05, 0.26, 0.05, 0.02, 0.05, 0.02)  # instantaneous P, sum = 0.4
32
33 # Defining our estimates for unknown mortality (M)
34
35 # Here, M is 0 in the first reach, elevated in reach 4, moderate in
```

```r
# reaches 3 and 5, and low in reaches 2 and 6.
M <- c(1e-05, 0.02, 0.05, 0.16, 0.05, 0.02)  # instantaneous M, sum = 0.3

# Calculating reach-specific total mortality (Z) and survival (S)
Z <- P + M  # reach-specific total instantaneous mortality
S <- exp(-Z)  # reach-specific discrete survival

# Calculating discrete and/or total estimates for all reaches
Z_D <- sum(Z)  # total instantaneous mortality
A_D <- 1 - exp(-Z_D)  # total discrete mortality; 0.503; roughly 50% of fish die
S_D <- exp(-sum(Z))  # total discrete survival; 0.497; roughly 50% of fish survive
P_D <- (sum(P) * A_D)/Z_D  # total discrete predation probability for all reaches
M_D <- (sum(M) * A_D)/Z_D  # total discrete unknown mortality for all reaches

# Creating the state process matrix
PSI.STATE <- array(NA, dim = c(n.states, n.states, n.reach))
for (r in 1:(n.reach)) {
    PSI.STATE[, , r] <- matrix(c(S[r], P[r] * (1 - S[r])/Z[r], M[r] *
        (1 - S[r])/Z[r], 0, 1, 0, 0, 0, 1), nrow = n.states, byrow = TRUE)
}  # r

# Printing the first two reaches for PSI.STATE
PSI.STATE[, , 1:2]
```

```
## , , 1
##
##         [,1]       [,2]       [,3]
## [1,] 0.99998 9.9999e-06 9.9999e-06
## [2,] 0.00000 1.0000e+00 0.0000e+00
## [3,] 0.00000 0.0000e+00 1.0000e+00
##
## , , 2
##
##           [,1]      [,2]       [,3]
## [1,] 0.7557837 0.2267722 0.01744402
## [2,] 0.0000000 1.0000000 0.00000000
## [3,] 0.0000000 0.0000000 1.00000000
```

Explaining the state-process (PSI.STATE)

PSI.STATE should produce an array of 3x3 matrices, and each matrix denotes the probability of a fish transitioning between true model states (see Table 1). For reach 1 the probability of an alive fish remaining alive [1,1] is 0.99998; the probability of an alive fish experiencing predation [1,2] is 9.9999e-06, and the probability of an alive fish dying from unknown causes [1,3] is 9.9999e-06. This is because our survival for reach 1 should be nearly 100%. Fish that experience predation [2,1] and fish that died from unknown causes [3,1] cannot be detected alive. Fish that experience predation will always remain a predation mortality, and dead fish from unknown causes will remain dead from unknown causes, as these are terminal states ([2,2] and [3,3]; respectively). Fish that experience predation cannot die from unknown causes [2,3], and fish that died from unknown causes cannot die from predation [3,2]. For reach 2 the probability of an alive fish (state 1) remaining alive (state 1) is 0.755..; meaning that fish alive at the end of reach 1 now have a 75% probability of surviving through the next reach. So, if 100 fish survived in reach 1, 75 will survive in reach 2. If 99 fish survived in reach 1, ~74 would survive in reach 2 (and so on).

Setting up our observation process

In this scenario, we observe predation mortality (P) and survival (S), and we are not observing fish that die of unknown causes. Additionally, this model scenario assumes that the researcher has created their capture histories using only data from a low density passive acoustic array with no supplementation from active tracking. We set detection probability to be equal across all reaches, aside from reach 1 (the release site) where all fish were assumed to be released alive with 100% detection probability.

```
# Defining our number of observation states
n.obs <- 3

# Defining our reach-specific detection probability (p)
p <- c(1, 0.65, 0.65, 0.65, 0.65, 0.65, 0.65)

# Creating the observation process matrix
PSI.OBS <- array(NA, dim = c(n.states, n.obs, n.reach.obs))
for (r in 1:n.reach.obs) {
    PSI.OBS[, , r] <- matrix(c(p[r], 0, 1 - p[r], 0, 1, 0, 0, 0, 1),
        nrow = n.states, byrow = TRUE)
}  # r

# Printing the first two reaches for PSI.OBS
PSI.OBS[, , 1:2]
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,] 0.65    0 0.35
## [2,] 0.00    1 0.00
## [3,] 0.00    0 1.00
```

Explaining the observation-process (PSI.OBS)

PSI.OBS should produce an array of 3x3 matrices, and each matrix denotes the probability of observing fish in each state. We released all fish alive in reach 1, so we assume that we detect all states with 100% accuracy (p=1). For reach 1, we assume that we observe alive fish that are truly alive with a 0.65 probability [1,1], and that we do not observe 0.35 (35%) of alive fish that are truly alive [1,3]. We do not observe alive fish that experienced predation [1,2], fish that experience predation are observed as such, and all fish that have died from unknown causes are unobserved ([2,2] and [3,3]; respectively). Thus, an alive fish's capture history with 0.65 detection probability would be similar to: 1-3-3-3-1-1 for reaches 1-6, where the fish is actually alive in the unobserved reaches (1,2,3).

Define the function to simulate multistate capture-recapture data

This code creates the input capture histories (CH) using the PSI.STATE and PSI.OBS matrices, where both true states (CH.TRUE) and observed state (CH) capture history matrices are created. Studies using real data would create their CH by interpreting their real telemetry data.

```r
simul.ms <- function(PSI.STATE, PSI.OBS, nind, unobservable = NA) {
    n.reach.obs <- dim(PSI.STATE)[3] + 1
    CH <- CH.TRUE <- matrix(NA, ncol = n.reach.obs, nrow = nind)
    for (i in 1:nind) {
        CH[, 1] <- CH.TRUE[, 1] <- 1  # first column all fish are released alive
        for (r in (2:n.reach.obs)) {
            # Multinomial trials for state transitions
            state <- which(rmultinom(1, 1, PSI.STATE[CH.TRUE[i, r - 1],
                , r - 1]) == 1)
            # which() = which state gets the 1 random draw at reach
            # r given state at r-1
            CH.TRUE[i, r] <- state  # then fills in true states
            # Multinomial trials for observation process
            event <- which(rmultinom(1, 1, PSI.OBS[CH.TRUE[i, r], , r]) ==
                1)
            # which observation gets the 1 random draw, given true
            # reach r state.
            CH[i, r] <- event
        }  # r
    }  # i
    return(list(CH = CH, CH.TRUE))
}
```

Setting our number of simulation repetitions

The number of repetitions provides the number of estimate outputs (your estimate sample size). Increasing the number of repetitions does not result in an eventual plateau, you'll never achieve the same results by altering this number. We suggest starting with a low number of repetitions (2-5) when initially verifying that the models are running and providing the desired results as processing time can be high (see Figure 6).

```r
# 100 is a good number of repetitions, and has been used in other
# studies (e.g., Hightower & Harris 2017)
Reps <- 100
```

Data setup check code

This code ensures that the vector lengths of the simulation data are equal.

```r
# Data set-up check

# For our number of individuals
if (nind_sims != length(ind_sims)) {
    print("Length mismatch for number of individuals")
} else {
    print("individual setup OK")
}
```

```
## [1] "individual setup OK"
```

```r
# For the state-process
if (n.reach != length(P) | n.reach != length(M)) {
    print("Length mismatch for state estimates")
} else {
    print("estimates setup OK")
}
```

```
## [1] "estimates setup OK"
```

```
# For p
if (n.reach.obs != length(p)) {
    print("Length mismatch for p")
} else {
    print("p setup OK")
}
```

```
## [1] "p setup OK"
```

Section 2: Three-observation three-state capture-recapture model

The three-state capture-recapture model

The below model code estimates instantaneous reach-specific predation (P), unknown mortality (M), and total mortality (Z), as well as reach-specific survival (S), using the capture histories simulated above and the probability of an individual transitioning from the alive state to either mortality source within each reach. Additionally, total study wide discrete estimates of predation (P_D), unknown mortality (M_D), and survival (S) are derived. The model has uninformative priors and was developed using Kéry & Schaub 2012 as well as Hightower and Harris 2017. Code annotation describe each aspect of the model.

```
# Model code for JAGS
mod = function() {
    # Priors

    # We are estimating our parameters on the reach-specific level
    # (r), e.g., our P estimate is P[r]. In models where
    # individual-level (i) indexing is necessary, this can be
    # expanded to P[r,i].

    # We use an uninformative prior for M & P on the natural log
    # scale.  We do this instead of the response scale (e.g.,
    # dunif(0,1)) because on the response scale, reach-specific
    # mortality is overestimated when values are very low (e.g. 0).
    # We recommend playing around with your priors, even if
    # uninformative.
    for (r in 1:(Reaches)) {
        ln.P[r] ~ dunif(-10, 1)  # uninformative prior for P on the natural log scale
        P[r] <- exp(ln.P[r])  # transforming the P prior to the response scale
        ln.M[r] ~ dunif(-10, 1)  # uninformative prior for M on the natural log scale
        M[r] <- exp(ln.M[r])  # transforming the M prior to the response scale
        Z[r] <- M[r] + P[r]  # total instantaneous reach-specific mortality
        S[r] <- exp(-Z[r])  # total reach-specific survival
    }  # r

    # Derived sums of discrete mortality for reaches 1-6.
    Z_D <- sum(Z[1:(Reaches)])  # total instantaneous mortality
    S_D <- exp(-Z_D)  # total discrete survival
    A_D <- 1 - exp(-Z_D)  # total discrete mortality
    P_D <- (sum(P[1:(Reaches)]) * A_D)/Z_D  # discrete total predation
    M_D <- (sum(M[1:(Reaches)]) * A_D)/Z_D  # discrete total other mortality

    # Detection probability
    p[1] <- 1  # we assume we detect 100% of fish in reach 1, so p = 100% (or 1)
```

```
    for (r in 2:(Reaches + 1)) {
        p[r] ~ dunif(0, 1)  # uninformative prior for p on the response scale
    }  # r

    # Define state-transition (ps) and observation matrices (po).
    for (i in 1:nFish) {
        # Define probabilities of State (r+1) given State (r).
        # First index is state at reach r, next is state at r+1.

        # The first[i]:(last[i]-1) code allows for staggered entry
        # into the system.  In this scenario, a staggered entry
        # would be releasing fish in reach 1 and reach 2 (for
        # example).  We release all fish in reach 1 in this study,
        # but we retain this code for flexibility in model
        # application.
        for (r in first[i]:(last[i] - 1)) {
            ps[1, i, r, 1] <- S[r]  # alive fish remains alive
            ps[1, i, r, 2] <- P[r] * (1 - S[r])/Z[r]  # alive fish experiences predation
            ps[1, i, r, 3] <- M[r] * (1 - S[r])/Z[r]  # alive fish dies of unknown causes
            ps[2, i, r, 1] <- 0
            ps[2, i, r, 2] <- 1
            ps[2, i, r, 3] <- 0
            ps[3, i, r, 1] <- 0
            ps[3, i, r, 2] <- 0
            ps[3, i, r, 3] <- 1
        }  # r
        for (r in first[i]:(last[i])) {
            # Define probabilities of Observed (r) given State (r).
            # First index is state, last index is observed
            po[1, i, r, 1] <- p[r]  # alive fish is detected alive
            po[1, i, r, 2] <- 0
            po[1, i, r, 3] <- 1 - p[r]  # alive fish is not detected
            po[2, i, r, 1] <- 0
            po[2, i, r, 2] <- 1  # fish is a '2' (predation) it stays a 2
            po[2, i, r, 3] <- 0
            po[3, i, r, 1] <- 0
            po[3, i, r, 2] <- 0
            po[3, i, r, 3] <- 1  # fish is a '3' (unknown M) it stays a 3
        }  # r
    }  # i

    # Likelihood process
    for (i in 1:nFish) {
        z[i, first[i]] <- 1  # individuals are alive at first occasion in study
        for (r in (first[i] + 1):last[i]) {
            z[i, r] ~ dcat(ps[z[i, r - 1], i, r - 1, ])
            # State process: draw State (r) given State (r-1)
        }  # r
        for (r in first[i]:last[i]) {
            y[i, r] ~ dcat(po[z[i, r], i, r, ])
            # Observation process: draw Observed (r) given State
            # (r)
        }  # r
    }  # i
}  # final bracket

```

```
# Write model to a text file
model.file = "model1.txt"
R2OpenBUGS::write.model(mod, model.file)
```

Initializing results storage

Below, we create empty vectors to store our model outputs for each repetition (Reps) and each number of individuals simulated (nind_sims):

```
# Vector to record the start & end time of each model run.
starttime <- vector("list", nind_sims)
endtime <- vector("list", nind_sims)

# Setting up our model data and writing a loop to iterate through
# our Reps and nind_sims.
Ests_D <- vector("list", nind_sims)  # empty vector for all discrete estimates
Ests_P <- vector("list", nind_sims)  # empty vector to store reach-specific P
Ests_M <- vector("list", nind_sims)  # empty vector to store reach-specific M
Ests_S <- vector("list", nind_sims)  # empty vector to store reach-specific S
det_p <- vector("list", nind_sims)  # empty vector to store our detection probabilities (p)
n_state_recs <- vector("list", nind_sims)

for (sim in 1:nind_sims) {
    starttime[[sim]] <- vector("list", Reps)
    endtime[[sim]] <- vector("list", Reps)
    Ests_D[[sim]] <- vector("list", Reps)
    Ests_P[[sim]] <- vector("list", Reps)
    Ests_M[[sim]] <- vector("list", Reps)
    Ests_S[[sim]] <- vector("list", Reps)
    det_p[[sim]] <- vector("list", Reps)
    n_state_recs[[sim]] <- vector("list", Reps)
}  # sim
```

Main simulation loop and extracting results for each simulation and repetition

Finally, the model will estimate survival (S), predation (P) and unknown mortality (M) using our simulated capture histories. The empty vectors created above will store all model outputs.

```
for (sim in 1:nind_sims) {
    nind <- ind_sims[sim]
    for (rep in 1:Reps) {
        simCH <- simul.ms(PSI.STATE, PSI.OBS, nind)
        CH <- simCH$CH
        CH.TRUE <- simCH[[2]]
        y = CH
        nFish = dim(CH)[1]
        Reaches = dim(CH)[2] - 1
        first <- numeric()
        for (i in 1:dim(y)[1]) {
            first[i] <- min(which(y[i, ] != 0))
        }  # i
        last <- numeric()
        for (i in 1:dim(y)[1]) {
            last[i] <- max(which(y[i, ] != 0))
```

```
250         }  # i
251         f <- first
252         l <- last
253 
254         # Initialize reach-state count matrix Rows: periods
255         # (reaches), Cols: states (1,2,3)
256         n_state_record <- matrix(NA, nrow = ncol(CH.TRUE), ncol = 3,
257             dimnames = list(paste0("R", 1:ncol(CH.TRUE)), c("S", "P",
258                 "M")))
259 
260         for (r in 1:ncol(CH.TRUE)) {
261             n_state_record[r, 1] <- sum(CH.TRUE[, r] == 1, na.rm = TRUE)  # alive
262             n_state_record[r, 2] <- sum(CH.TRUE[, r] == 2, na.rm = TRUE)  # predation mort.
263             n_state_record[r, 3] <- sum(CH.TRUE[, r] == 3, na.rm = TRUE)  # unknown mort
264             # if you have NA, they won't be counted
265         }  # r
266 
267         # Store in list
268         n_state_recs[[sim]][[rep]] <- n_state_record
269 
270         jags.data <- list(y = y, nFish = nFish, first = first, last = last,
271             Reaches = Reaches)
272 
273         # Initial function - this aids in model speed by creating
274         # rules around how the unobserved state should be
275         # 'back-filled'. For example, if a fish has a capture
276         # history of 1-3-3-1-1-1, we know the fish in reaches 2 and
277         # 3 is actually alive, so this function fills the 3s as 1s
278         # and dings our detection probability in reaches 2 and 3.
279         ms.init.z <- function(y, f) {
280             y.iv <- y
281             y.iv[y.iv == 3] <- NA  # change this to the unobserved state
282 
283             for (i in 1:nrow(y.iv)) {
284                 if (max(y.iv[i, ], na.rm = TRUE) == 1) {
285                   y.iv[i, (f[i] + 1):l[i]] <- 1
286                 }
287                 # not detected dead so initialize as alive
288 
289                 if (max(y.iv[i, ], na.rm = TRUE) == 2)
290                   {
291                     m <- min(which(y.iv[i, ] == 2))
292                     y.iv[i, f[i]:(m - 1)] <- 1
293                     # predation not detected so initialize as alive
294                     y.iv[i, m:l[i]] <- 2
295                   }  # predation detected, terminal until end of CH
296             }  # i
297 
298             for (i in 1:dim(y.iv)[1]) {
299                 y.iv[i, 1:f[i]] <- NA
300             }  # i
301             return(y.iv)
302         }  # final initial function contains 1s and 2s
303 
304         jags.inits <- function() {
305             list(ln.M = runif(Reaches, -10, 1), ln.P = runif(Reaches,
```

```
                -10, 1), z = ms.init.z(y, f))
        }

        # Parameters to monitor during model run

        # For this model, we are interested in our final estimates
        # for P, M, and S (P_D, M_D, S_D), our reach-specific
        # estimates (P, M, S), and detection probability (p).
        params <- c("P_D", "M_D", "S_D", "p", "P", "M", "S")

        # Run model in JAGS

        # NOTE: R2Jags and jagsUI both use a function called
        # 'autojags'. We used the jagsUI package,the syntax will
        # give an error if the R2Jags package is used instead.
        starttime[[sim]][[rep]] <- Sys.time()  # record start time
        jagsfit <- jagsUI::autojags(data = jags.data, inits = jags.inits,
            parameters.to.save = params, model.file, n.chains = 3, n.adapt = NULL,
            iter.increment = 3000, n.burnin = 10000, n.thin = 1, save.all.iter = FALSE,
            modules = c("glm"), parallel = TRUE, DIC = TRUE, store.data = FALSE,
            codaOnly = FALSE, bugs.format = FALSE, Rhat.limit = 1.05,
            max.iter = 5e+05, verbose = TRUE)
        endtime[[sim]][[rep]] <- Sys.time()  # record end time

        # For quantiles & parameter extracts
        est_names <- c("q2.5", "q25", "q50", "q75", "q97.5")
        param_names_D <- str_subset(params, "_D$")

        Est_D <- array(NA, dim = c(length(est_names), length(param_names_D)),
            dimnames = list(est_names, param_names_D))

        for (param in param_names_D) {
            for (quant in est_names) {
                Est_D[quant, param] <- jagsfit[[quant]][[param]]
            }  # quant
        }  # param

        Ests_D[[sim]][[rep]] <- Est_D

        # For p extract R0 is a place holder for the transition
        # from release to reach 2
        reaches <- c("R0", "R1", "R2", "R3", "R4", "R5", "R6")
        det_p_ <- matrix(NA, nrow = length(est_names), ncol = n.reach.obs,
            dimnames = list(est_names, reaches))

        for (quant in est_names) {
            det_p_[quant, ] <- jagsfit[[quant]]$p
        }  # quant

        det_p[[sim]][[rep]] <- det_p_

        # For P extract
        Est_P <- matrix(NA, nrow = length(est_names), ncol = n.reach,
            dimnames = list(est_names, reaches[-1]))

        for (quant in est_names) {
```

```
            Est_P[quant, ] <- jagsfit[[quant]]$P
        }  # quant

        Ests_P[[sim]][[rep]] <- Est_P

        # For M extract
        Est_M <- matrix(NA, nrow = length(est_names), ncol = n.reach,
            dimnames = list(est_names, reaches[-1]))

        for (quant in est_names) {
            Est_M[quant, ] <- jagsfit[[quant]]$M
        }  # quant

        Ests_M[[sim]][[rep]] <- Est_M

        # For S extract
        Est_S <- matrix(NA, nrow = length(est_names), ncol = n.reach,
            dimnames = list(est_names, reaches[-1]))

        for (quant in est_names) {
            Est_S[quant, ] <- jagsfit[[quant]]$S
        }  # quant

        Ests_S[[sim]][[rep]] <- Est_S

    }  # rep
}  # sim

save.image("model1.RData")
```

Converting outputs to exportable flat files (.csv)

```
# The individual simulations code as 1-length(nind_sims); so this
# codes them back into the actual number of individuals
nind_ <- tibble(sim = c(1:length(ind_sims)), ind_sims)

# Storing run times for each model
i <- 1
starttime_df <- list()
for (r in 1:Reps) {
    for (s in 1:nind_sims) {
        df <- tibble(starttime = as.POSIXct(starttime[[s]][[r]])) %>%
            mutate(ind_sims = ind_sims[s], rep = r)
        starttime_df[[i]] <- df
        i <- i + 1
    }  # s
}  # r

starttime_df <- do.call("rbind", starttime_df)

i <- 1
endtime_df <- list()
for (r in 1:Reps) {
    for (s in 1:nind_sims) {
```

```
        df <- tibble(endtime = as.POSIXct(endtime[[s]][[r]])) %>%
            mutate(ind_sims = ind_sims[s], rep = r)
        endtime_df[[i]] <- df
        i <- i + 1
    }  # s
}  # r

endtime_df <- do.call("rbind", endtime_df)
runtimes <- full_join(starttime_df, endtime_df)
runtimes <- runtimes %>%
    mutate(runtime_seconds = as.numeric(endtime - starttime))

# Unpacking the reach-specific number of individuals in each state
N_State_recs_df <- purrr::map2_dfr(n_state_recs, seq_along(n_state_recs),
    function(s_list, s) {
        purrr::map2_dfr(s_list, seq_along(s_list), function(est_array,
            r) {
            as.data.frame(as.table(est_array)) %>%
                rename(reach = Var1, parameter = Var2, N = Freq) %>%
                mutate(sim = s, rep = r)
        })
    }) %>%
    left_join(nind_)

# Unpacking the model estimates and calculating percent relative
# error (MRE) and the coefficient of variation (CV)
Ests_df <- purrr::map2_dfr(Ests_D, seq_along(Ests_D), function(s_list,
    s) {
    purrr::map2_dfr(s_list, seq_along(s_list), function(est_array, r) {
        as.data.frame(as.table(est_array)) %>%
            dplyr::rename(quantile = Var1, parameter = Var2, value = Freq) %>%
            mutate(sim = s, rep = r)
    })
}) %>%
    dplyr::rename(Est = value) %>%
    mutate(true_value = case_when(parameter == "S_D" ~ S_D, parameter ==
        "P_D" ~ P_D, parameter == "M_D" ~ M_D, )) %>%
    mutate(MRE = (Est - true_value)/true_value * 100) %>%
    dplyr::group_by(sim, quantile, parameter) %>%
    mutate(CV = ((sd(Est)/mean(Est)) * 100)) %>%
    mutate(df = "Est") %>%
    left_join(nind_)

# Unpacking the reach-specific detection probability estimates
p_df <- purrr::map2_dfr(det_p, seq_along(det_p), function(s_list, s) {
    purrr::map2_dfr(s_list, seq_along(s_list), function(est_array, r) {
        as.data.frame(as.table(est_array)) %>%
            dplyr::rename(quantile = Var1, reach = Var2, value = Freq) %>%
            mutate(sim = s, rep = r)
    })
}) %>%
    mutate(df = "p") %>%
    left_join(nind_)

ests_list <- list(M = Ests_M, P = Ests_P, S = Ests_S)

```

```
469 # Function for processing each list for the reach-specific
470 # estimates
471 process_ests <- function(x) {
472     map2_dfr(x, seq_along(x), function(s_list, s) {
473         map2_dfr(s_list, seq_along(s_list), function(est_array, r) {
474             as.data.frame(as.table(est_array)) %>%
475                 rename(quantile = Var1, reach = Var2, value = Freq) %>%
476                 mutate(sim = s, rep = r)
477         })
478     })
479 }
480
481 # Apply to all reach-specific estimates, output named list: M_df,
482 # P_df, S_df
483 result <- imap(ests_list, ~process_ests(.x) %>%
484     mutate(parameter = .y))
485 result <- do.call("rbind", result)
486
487 trueests_p <- tibble(parameter = rep(c("M", "P", "S"), each = 6), true_value = c(M,
488     P, S), reach = rep(c("R1", "R2", "R3", "R4", "R5", "R6"), times = 3))
489 Ests_P_df <- result %>%
490     dplyr::rename(Est = value) %>%
491     left_join(trueests_p) %>%
492     mutate(MRE = (Est - true_value)/true_value * 100) %>%
493     dplyr::group_by(sim, quantile, parameter, reach) %>%
494     mutate(CV = ((sd(Est)/mean(Est)) * 100)) %>%
495     mutate(df = "Est") %>%
496     left_join(nind_)
497
498 # Saving data
499 Ests_df %>%
500     write_csv("model1_estimates.csv")
501 Ests_P_df %>%
502     write_csv("model1_reachestimates.csv")  # reach-specific estimates
503 p_df %>%
504     write_csv("model1_detectionprobability.csv")
505 N_State_recs_df %>%
506     write_csv("model1_Nfishstate.csv")
507 runtimes %>%
508     write_csv("model1_runtimes.csv")
```

Model Scenario 2: High detection probablility (passive acoustic telemetry array and active tracking); three-state and three-observation multistate mark-recapture model (MSMR)

Increasing the detection probability

To increase the detection probability, the researcher only needs to amend one line of the code from Model Scenario 1; Line 63, where we increase our "low" detection probability p=0.65 to a "high" detection probability, p=0.9.

```
62 # Defining our reach-specific detection probability (p)
63 p <- c(1, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9)
```

Once p is updated, the Model Scenario 1 code can be run as is for Model Scenario 2. Ensure that the model.file (Line 209), RData image (Line 391) and data saving (Lines 499-509) are updated to "model2".