Appendix S2

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

```
# 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)

```

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

Model Scenario 3: High detection probability (passive acoustic telemetry array and active tracking); three-state and three-observation multistate mark-recapture model (MSMR) with covariates

Section 1: Creating the simulation data

See Appendix S1 (Model Scenarios 1 & 2) for information regarding the MSMRs.

The below code does not change from Appendix S1:

```
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
```

Model covariate parameters and generating estimates for P, M, and S using covariates

The below code defines a function called T.mats that simulates covariate-driven mortality and survival probabilities for the three-state three-observation MSMR. The function generates matrices for survival (S; state 1), predation mortality (P; state 2), and unknown mortality (M; state 3) with a relationship to length and temperature covariates.

Temperature Simulation (Lines 23-44)

Two groups of fish are simulated: one with cooler starting temperatures (temp.c) and one with warmer starting temperatures (temp.h). For each fish, temperature is simulated across reaches, with random increments added to simulate changes in temperature as fish move downstream. Temperature data can be acquired from the PDATs, hydrological gauges, etc.

Length Simulation (Lines 46-47)

Fish lengths are randomly drawn from a uniform distribution between 80 and 160 (cm). This range of values can be expanded depending on the target species and its typical length distribution.

Defining mortality-covariate parameters (Lines 64-70)

The function sets up parameters for two mortality models: a0 and a1 define a model for apparent survival probability (P) as a function of fish length while b0 and b1 define a model for natural mortality (M) as a function of temperature.

Creating the simulated capture histories (Lines 87-101)

For each fish and each reach, the function calculates: P: apparent survival probability (function of length) M: natural mortality probability (function of temperature) Z: total mortality (sum of P and M) S: survival probability (exponential of negative Z)

Defining our state and observation matrices (Lines 103-132)

See Appendix S1 for description on the state-process and observation-process. Note, in this model PSI.STATE indexes individual-level covariates, so state[j,r] where j is individual and r is reach. In Appendix 1, PSI.STATE only indexed per-reach states, so state[r].

```r
T.mats <- function(nind, n.reach, n.reach.obs){

# First half of fish released in cool season temp varies from ~ 5 to 15
  temp.c <- rnorm(nind/2, 5, 0.25) # normal distribution around 5 degrees
  temp.mat.c <- matrix(NA, nrow = nind/2, ncol = n.reach)
  temp.mat.c[,1] <- temp.c
  for (j in 1:nind/2) {
    increments <- rnorm(n.reach - 1, mean = 2, sd = 0.2)
    temp.mat.c[j, ] <- temp.mat.c[j] + c(0, cumsum(increments))
  } # j

# Second half of fish released in warm season temp varies from ~ 15 to 25
  temp.h <- rnorm(nind/2, 15, 0.25) # normal distribution around 15 degrees
  temp.mat.h <- matrix(NA, nrow = nind/2, ncol = n.reach)
  temp.mat.h[,1] <- temp.h
  for (j in 1:nind/2) {
    increments <- rnorm(n.reach - 1, mean = 2, sd = 0.2)
    temp.mat.h[j, ] <- temp.mat.h[j] + c(0, cumsum(increments))
  } # j

# Join temp data together
  temp.all <- rbind(temp.mat.c,temp.mat.h)

# Simulate length data
  length_vec <- round(runif(nind, min = 80, max = 160)) # norm. dist. from 80 to 160

# Scale and Center covariates prior to modeling.
# JAGS converges easier when covariates are centered and scaled, this is also necessary
# if more than one variable are included in a single GLM. Doing it here ensures that
# the simulation directly complements the model code below.

  temp.scl.vec <- scale(as.vector(temp.all))
  temp.cnt.value <- attr(temp.scl.vec,'scaled:center')
  temp.scl.value <- attr(temp.scl.vec,'scaled:scale')
  temp.scl.mat <- matrix(temp.scl.vec,nrow = nind, ncol = n.reach)

  length.scl.vec <- scale(length_vec)
  length.cnt.value <- attr(length.scl.vec,'scaled:center')
  length.scl.value <- attr(length.scl.vec,'scaled:scale')
  length.scl.vec <- as.numeric(scale(length_vec))

  # Defining mortality-covariate parameters
  # a = P model, a function of length
  # b = M model, a function of temperature
  a1.raw <- -0.03 # slope parameter for P; negative relationship with length
  a0.raw <- 0.9 # intercept parameter for P
  b1.raw <- 0.2 # slope parameter for M; positive relationship with temperature
  b0.raw <- -5 # intercept parameter for M

# Transform raw equation values to scaled scale for use in CH matrix and pull
# for back transformations below.
  a1 <- a1.raw*length.scl.value
  a0 <- a0.raw + (a1* (length.cnt.value/length.scl.value))
  b1 <- b1.raw*temp.scl.value
  b0 <- b0.raw + (b1* (temp.cnt.value/temp.scl.value))

```

```r
# Test plots: If you run the below lines outside of function you can visualize
# the relationship from GLM parameters above.

  # plot(length.scl.vec,exp(a0 + a1 * length.scl.vec))
  # plot(temp.scl.mat,exp(b0 + b1 * temp.scl.mat))
  # plot(temp.all, exp((b0 - (b1* (temp.cnt.value/temp.scl.value))) +
  # (b1/temp.scl.value)*temp.all))

  # Allocate mortality matrices for this simulation
  P <- matrix(NA, nrow = nind, ncol = n.reach)
  M <- matrix(NA, nrow = nind, ncol = n.reach)
  Z <- matrix(NA, nrow = nind, ncol = n.reach)
  S <- matrix(NA, nrow = nind, ncol = n.reach)

  # Fill matrices
  for (j in 1:nind) {
    for (r in 1:(n.reach)) {
      P[j, r] <- exp(a0 + a1 * length.scl.vec[j])
      M[j, r] <- exp(b0 + b1 * temp.scl.mat[j, r])
      Z[j, r] <- P[j, r] + M[j, r]
      S[j, r] <- exp(-Z[j, r])
    } # r
  } # j

# Defining our number of model states
n.states <- 3

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

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

# Creating the state process matrix
# For the covariate model, we are indexing covariates that are associated at the
# individual level so PSI.STATE now indexes [j,r] rather than only [r].
PSI.STATE <- array(NA, dim=c(n.states, n.states, nind, n.reach))
for(j in 1:nind){
for (r in 1:(n.reach)){
  PSI.STATE[,,j,r] <- matrix(c(
    S[j,r], P[j,r]*(1-S[j,r])/Z[j,r], M[j,r] *(1-S[j,r])/Z[j,r], #
    0, 1, 0,
    0, 0, 1), nrow = n.states, byrow = TRUE)
  } # r
} # j

# 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

return(list(PSI.STATE = PSI.STATE,PSI.OBS = PSI.OBS,
```

```
            length_vec = length_vec, temp.all = temp.all,
            length.scl.FL = length.scl.vec,
            temp.scl.C = temp.scl.mat,
            length.scl.value = length.scl.value,
            length.cnt.value = length.cnt.value,
            temp.scl.value = temp.scl.value,
            temp.cnt.value = temp.cnt.value))
} # final bracket for T.mats
```

Define the function to simulate multistate capture-recapture data

The below chunk is slightly edited from Appendix S1; our dim(PSI.STATE) now indexes from the 4th observation (Line 144) and the dimensions of PSI.STATE are edited from PSI.STATE[CH.TRUE[i,r-1],,r-1] to PSI.STATE[CH.TRUE[i,r-1],,i,r-1] (Line 156).

```
simul.ms <- function(PSI.STATE, PSI.OBS, nind, unobservable = NA) {
    n.reach.obs <- dim(PSI.STATE)[4] + 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],
                , i, 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 below code does not change from Appendix S1:

```
# 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

Since our p, P, and M values are derived within the T.mats function, the code can not be verified for vector length mismatch.

```
# 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")
}
```

Section 2: Three-observation three-state multistate mark-recapture model with covariates

The three-state multistate mark-recapture model with covariates

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. This model also incorporates biological (fork length) and environmental (temperature) covariates to predict predation (P) and unknown (M) mortality; respectively. 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. **Note** the probability state (ps) matrix is expanded to predict on the individual level, so the code from Appendix S1 is modified from ps[1,i,r,1] <- S[r] to ps[1,i,r,1] <- S[i,r], etc. The detection probability, (Lines 206-210, observation matrix (po; Lines 244-256), and likelihood function (Lines 259-271) code remains the same from Appendix S1.

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

    # The glms used to define P and M match our simulated
    # equations, where length is the explanatory covariate for P
    # and temperature is the explanatory covariate for M.
    for (i in 1:nFish) {
        for (r in 1:(Reaches)) {
            log(P[i, r]) <- a0 + a1 * length.scl.FL[i]
            log(M[i, r]) <- b0 + b1 * temp.scl.C[i, r]
            Z[i, r] <- P[i, r] + M[i, r]
            S[i, r] <- exp(-Z[i, r])
        }  # r
    }  # i

    # Priors for a0, a1, b0, and b1
    b0 ~ dunif(-10, 1)
    b1 ~ dnorm(0, 0.01)

    a0 ~ dunif(-10, 1)
    a1 ~ dnorm(0, 0.01)

    # Convert model parameters back to raw scale
    a0.raw <- a0 - (a1 * (length.cnt.value/length.scl.value))
    a1.raw <- a1/length.scl.value

    b0.raw <- b0 - (b1 * (temp.cnt.value/temp.scl.value))
    b1.raw <- b1/temp.scl.value

    # 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

    # Predicted values

    # Using our estimated values for a0, a1, b0, and b1 we predict
    # P and M across a vector of lengths and temperatures;
    # respectively.
    for (j in 1:length(length.scl.predict)) {
        log(P.pred[j]) <- a0 + a1 * length.scl.predict[j]
        log(M.pred[j]) <- b0 + b1 * temp.scl.predict[j]
    }  # j

    # 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[i, r]  # alive fish remains alive
            ps[1, i, r, 2] <- P[i, r] * (1 - S[i, r])/Z[i, r]  # alive fish experiences pred.
            ps[1, i, r, 3] <- M[i, r] * (1 - S[i, r])/Z[i, r]  # alive fish dies of unk. mort
            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 (t) given State (t).
            # 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 = "model3.txt"
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_mod_params <- vector("list", nind_sims)
P.pred.sims <- vector("list", nind_sims)
M.pred.sims <- vector("list", nind_sims)

for (sim in 1:nind_sims) {
    starttime[[sim]] <- vector("list", Reps)
    endtime[[sim]] <- vector("list", Reps)
    Ests_mod_params[[sim]] <- vector("list", Reps)
    P.pred.sims[[sim]] <- vector("list", Reps)
    M.pred.sims[[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 with explanatory covariates. The empty vectors created above will store all model outputs.

```
for (sim in 1:nind_sims) {
    nind <- ind_sims[sim]  # number of individuals
    for (rep in 1:Reps) {
        mats <- T.mats(nind, n.reach, n.reach.obs)  # function defined in Section 1
        simCH <- simul.ms(PSI.STATE = mats$PSI.STATE, PSI.OBS = mats$PSI.OBS,
            nind)
        CH <- simCH$CH
        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))
```

```
307         }  # i
308         last <- numeric()
309         for (i in 1:dim(y)[1]) {
310             last[i] <- max(which(y[i, ] != 0))
311         }  # i
312         f <- first
313         l <- last
314 
315         lengthpredict <- seq(80, 160, length.out = 100)
316         length.scl.predict <- (lengthpredict - mats$length.cnt.value)/mats$length.scl.value
317 
318         temppredict <- seq(5, 25, length.out = 100)
319         temp.scl.predict <- (temppredict - mats$temp.cnt.value)/mats$temp.scl.value
320 
321         jags.data <- list(y = y, nFish = nFish, first = first, last = last,
322             Reaches = Reaches, length.cnt.value = mats$length.cnt.value,
323             length.scl.value = mats$length.scl.value, temp.cnt.value = mats$temp.cnt.value,
324             temp.scl.value = mats$temp.scl.value, length.scl.FL = mats$length.scl.FL,
325             temp.scl.C = mats$temp.scl.C, length.scl.predict = length.scl.predict,
326             temp.scl.predict = temp.scl.predict)
327 
328         # Initial function - this aids in model speed by creating
329         # rules around how the unobserved state should be
330         # 'back-filled'. For example, if a fish has a capture
331         # history of 1-3-3-1-1-1, we know the fish in reaches 2 and
332         # 3 is actually alive, so this function fills the 3s as 1s
333         # and dings our detection probability in reaches 2 and 3.
334         ms.init.z <- function(y, f) {
335             y.iv <- y
336             y.iv[y.iv == 3] <- NA  # change this to the unobserved state
337 
338             for (i in 1:nrow(y.iv)) {
339                 if (max(y.iv[i, ], na.rm = TRUE) == 1) {
340                   y.iv[i, (f[i] + 1):l[i]] <- 1
341                 }
342                 # not detected dead so initialize as alive
343 
344                 if (max(y.iv[i, ], na.rm = TRUE) == 2)
345                   {
346                     m <- min(which(y.iv[i, ] == 2))
347                     y.iv[i, f[i]:(m - 1)] <- 1
348                     # predation not detected so initialize as alive
349                     y.iv[i, m:l[i]] <- 2
350                   }  # predation detected, terminal until end of CH
351             }  # i
352 
353             for (i in 1:dim(y.iv)[1]) {
354                 y.iv[i, 1:f[i]] <- NA
355             }  # i
356             return(y.iv)
357         }  # final initial function contains 1s and 2s
358 
359         jags.inits <- function() {
360             list(z = ms.init.z(y, f))
361         }
362 
```

```
        # Parameters to monitor during model run
        params <- c("a0.raw", "a1.raw", "b0.raw", "b1.raw", "P.pred",
            "M.pred")

        # 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 = 10000, n.burnin = 3000, 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 start time

        # For quantiles & parameter extracts
        est_names <- c("q2.5", "q25", "q50", "q75", "q97.5")
        param_names <- params[1:4]  #all GLM parameters

        Est_mod_params <- array(NA, dim = c(length(est_names), length(param_names)),
            dimnames = list(est_names, param_names))

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

        Ests_mod_params[[sim]][[rep]] <- Est_mod_params

        # For P.pred extract
        P.pred <- array(NA, dim = c(length(lengthpredict), length(est_names) +
            3), dimnames = list(1:100, c("n", "rep", "length_predict",
            est_names)))
        P.pred <- as.data.frame(P.pred)

        P.pred$n <- ind_sims[sim]
        P.pred$rep <- rep
        P.pred$length_predict <- lengthpredict

        for (quant in est_names) {
            P.pred[quant] <- jagsfit[[quant]]$P.pred
        }  # quant

        P.pred.sims[[sim]][[rep]] <- P.pred

        # For M.pred extract
        M.pred <- array(NA, dim = c(length(tempppredict), length(est_names) +
            3), dimnames = list(1:100, c("n", "rep", "temp_predict",
            est_names)))
        M.pred <- as.data.frame(M.pred)

        M.pred$n <- ind_sims[sim]
```

```
        M.pred$rep <- rep
        M.pred$temp_predict <- temppredict

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

        M.pred.sims[[sim]][[rep]] <- M.pred

    }  # rep
}  # sims

save.image(file = "model3.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

# Unpacking the model run times.
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 model estimates and calculating percent relative
# error (MRE) and the coefficient of variation (CV). These values
# are copied from Lines 67-70.
```

```r
a1.raw <- -0.03
a0.raw <- 0.9
b1.raw <- 0.2
b0.raw <- -5

Ests_df <- purrr::map2_dfr(Ests_mod_params, seq_along(Ests_mod_params),
    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 == "a0.raw" ~ a0.raw, parameter ==
        "a1.raw" ~ a1.raw, parameter == "b0.raw" ~ b0.raw, parameter ==
        "b1.raw" ~ b1.raw, )) %>%
    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 the predicted values
i <- 1
preds_df <- list()
for (r in 1:Reps) {
    for (s in 1:nind_sims) {
        M <- tibble(M.pred.sims[[s]][[r]])
        M <- M %>%
            rename(predicted_values = temp_predict) %>%
            mutate(Variable = "Temperature", parameter = "M")
        P <- tibble(P.pred.sims[[s]][[r]])
        P <- P %>%
            rename(predicted_values = length_predict) %>%
            mutate(Variable = "Length", parameter = "P")
        df <- full_join(M, P)
        preds_df[[i]] <- df
        i <- i + 1
    }  # s
}  # r
preds_df <- do.call("rbind", preds_df)

# Saving data
Ests_df %>%
    write_csv("model3_estimates.csv")
runtimes %>%
    write_csv("model3_runtimes.csv")
preds_df %>%
    write_csv("model3_predictions.csv")
```