R2. Model fitting and evaluation

2020 - 2021 Skagit River steelhead forecast.

Contents

Requirements **1**

User inputs **4**

Loading the fish data **5**

Loading the covariates **6**

Specifying models in JAGS **6**

Beverton-Holt with covars 6

Beverton-Holt with covars 10

Beverton-Holt with covars 13

Fitting the models and generating the one year ahead forecasts **16**

Model diagnostics 20

Model diagnostics 21

Model diagnostics AR1 recruitment residuals 21

Model selection **21**

Model Selection Via Approximate leave-future-out cross validation **24**

This is version 0.20.12.21.

Requirements

All analyses require the R software (v3.4.3) for data retrieval, data processing, and summarizing model results, and the JAGS software (v4.2.0) for Markov chain Monte Carlo (MCMC) simulation. Please note that some of the R code below may not work with older versions of JAGS due to some changes in the ways that arrays are handled.

We also need a few packages that are not included with the base installation of R, so we begin by installing them (if necessary) and then loading them.

```
if(!require("here")) {
  install.packages("here")
  library("here")
}
if(!require("readr")) {
  install.packages("readr")
  library("readr")
}
if(!require("rjags")) {
  install.packages("rjags")
  library("rjags")
}
if(!require("loo")) {
  install.packages("loo")
  library("loo")
}
if(!require("ggplot2")) {
  install.packages("ggplot2")
  library("ggplot2")
}
if(!require("coda")) {
  install.packages("coda")
  library("coda")
}
if(!require("shinystan")) {
  install.packages("shinystan")
  library("shinystan")
}
if(!require("R2jags")) {
  install.packages("R2jags")
  library("R2jags")
}
if(!require("dclone")) {
  install.packages("dclone")
  library("dclone")
}
if(!require("snow")) {
  install.packages("snow")
  library("snow")
}
## set directory locations
datadir <- here("data")
jagsdir <- here("jags")
analdir <- here("analysis")
savedir <- here("analysis/cache")
```

We also need a couple of helper functions.

```
## better round
Re2prec <- function(x, map = "round", prec = 1) {
  ## 'map' can be "round", "floor", or "ceiling"
```

```
  ## 'prec' is nearest value (eg, 0.1 means to nearest tenth; 1 gives normal behavior)
  if(prec<=0) { stop("\"prec\" cannot be less than or equal to 0") }
  do.call(map,list(x/prec))*prec
}

## wrapper function to fit JAGS models & rearrange output
fit_jags <- function(model, data, params, inits, ctrl, dir = jagsdir) {
  jm <- jags.model(file.path(jagsdir, model),
                   data,
                   inits,
                   ctrl$chains,
                   ctrl$burn,
                   quiet = TRUE)
  return(coda.samples(jm, params, ctrl$length, ctrl$thin))
}

#alternative wrapper to fit model in parallel; one chain per core
fit_jags2<-function(model,data,params,inits,ctrl,dir=jagsdir){
  # jm<-jags.parallel(data=data,
  #                   inits=inits,
  #                   parameters.to.save=params,
  #                   model.file = file.path(jagsdir, model),
  #                   n.chains = ctrl$chains,
  #                   n.iter = ctrl$length,
  #                   n.burnin = ctrl$burn,
  #                   n.thin = ctrl$thin,
  #                   DIC = TRUE,
  # )
  # write.csv(jm$BUGSoutput$summary,file.path(savedir,paste(model,".csv",sep="")))
  # jm<-jags.parfit(data=data,
  #                   inits=inits,
  #                   parameters.to.save=params,
  #                   model.file = file.path(jagsdir, model),
  #                   n.chains = ctrl$chains,
  #                   n.iter = ctrl$length,
  #                   n.burnin = ctrl$burn,
  #                   n.thin = ctrl$thin,
  #                   DIC = TRUE,
  # )
  # return(as.mcmc.list(as.mcmc(jm)))
  cl <- makeCluster(3, type = "SOCK")
  inits2 <- jags.fit(data=data,
                     params=params,
                     model=file.path(jagsdir, model),
                     inits=inits,
                     n.chains=ctrl$chains,
                     n.adapt = 0,
                     n.update = 0,
                     n.iter = 0)$state(internal = TRUE)
  jm <- jags.parfit(cl=cl,
                    data = data,
                    params = params,
                    model = file.path(jagsdir, model),
```

```
                    inits = inits2,
                    n.adapt = ctrl$burn*0.5,
                    n.update = ctrl$burn*0.5,
                    n.iter = ctrl$length-ctrl$burn,
                    thin = ctrl$thin,
                    n.chains = ctrl$chains
                    )
  stopCluster(cl)
  return(jm)
}
```

User inputs

We begin by supplying values for the following parameters, which we need for model fitting and evaluation.

```
## first & last years of fish data
yr_frst <- 1978
yr_last <- 2020



## min & max adult age classes
age_min <- 3
age_max <- 8

## years (if any) of age-comp to skip; see below
age_skip <- 0

## number of years ahead for run forecasts from the most recent year of data
n_fore <- 1

## number of recent year forecasts
n_forecasts <- 10

## first year of 1 step ahead forecast
yr_begin <- 2011

## last year of 1 step ahead forecast
yr_end <- 2020

## upper threshold for Gelman & Rubin's potential scale reduction factor (Rhat).
Rhat_thresh <- 1.1
```

Next we specify the names of three necessary data files containing the following information:

1. observed total number of adult spawners (escapement) by year;
2. observed age composition of adult spawners by year;
3. observed total harvest by year;

```
## 1. file with escapement data
## [n_yrs x 2] matrix of obs counts; 1st col is calendar yr
```

```
fn_esc <- "skagit_sthd_esc.csv"

## 2. file with age comp data
## [n_yrs x (1+A)]; 1st col is calendar yr
fn_age <- "skagit_sthd_age.csv"

## 3. file with harvest data
## [n_yrs x 2] matrix of obs catch; 1st col is calendar yr
fn_harv <- "skagit_sthd_catch.csv"
```

Loading the fish data

Here we load in the first three data files and do some simple calculations and manipulations. First the spawner data:

```
## escapement
dat_esc <- read_csv(file.path(datadir, fn_esc))
## years of data
dat_yrs <- dat_esc$year

## number of years of data
n_yrs <- length(dat_yrs)

## log of escapement
ln_dat_esc <- c(log(dat_esc$escapement),rep(NA,n_fore))
```

Next the age composition data:

```
## age comp data
dat_age <- read_csv(file.path(datadir, fn_age))
## num of age classes
A <- age_max - age_min + 1

# ## drop year col & first age_min+age_skip rows
# dat_age <- dat_age[-(1:(age_min+age_skip)),-1]
#
# ## add row(s) of NA's for forecast years
# if(n_fore > 0) {
#   dat_age <- rbind(dat_age,
#                    matrix(0, n_fore, A,
#                           dimnames =list(n_yrs+seq(n_fore),
#                                          colnames(dat_age))))
# }
# ## total num of age obs by cal yr
# dat_age[,"sum"] <- apply(dat_age, 1, sum)
# ## row indices for any years with no obs age comp
# idx_NA_yrs <- which(dat_age$sum<A, TRUE)
# ## replace 0's in yrs w/o any obs with NA's
# dat_age[idx_NA_yrs,(1:A)] <- NA
# ## change total in yrs w/o any obs from 0 to A to help dmulti()
# dat_age[idx_NA_yrs,"sum"] <- A
```

```
# ## convert class
# dat_age <- as.matrix(dat_age)
```

And then the harvest data:

```
## harvest
dat_harv <- read_csv(file.path(datadir, fn_harv))
## drop year col & first age_max rows
#dat_harv <- c(dat_harv$catch,rep(NA,n_fore))
```

Loading the covariates

Our analysis investigates 5 covariates as possible drivers of the population's instrinic growth rate:

1. Maximum river discharge in winter;
2. Minimum river discharge in summer;
3. North Pacific Gyre Oscillation;

All of the covariates are contained in the file `/data/skagit_sthd_covars.csv`. We will load and then standardize them to have zero-mean and unit-variance.

```
dat_cvrs <- read_csv(file.path(datadir, "skagit_sthd_covars.csv"))
## drop year col
# dat_cvrs <- dat_cvrs[,-1]
# ## transform the covariates to z-scores
# scl_cvrs <- as.matrix(scale(dat_cvrs))
# ## total number of covariates
# n_cov <- dim(dat_cvrs)[2]
```

Specifying models in JAGS

Now we can specify the model in JAGS. We fit a total one model, which we outline below, based on a beverton holt process model with covariates.

Beverton-Holt with covars

```
cat("
    model {

    ##--------
    ## PRIORS
    ##--------
    ## alpha = intrinsic productivity
    alpha ~ dnorm(0,0.001) T(0,);
    mu_BH_a <- log(alpha);
    E_BH_a <- mu_BH_a + sigma_r/(2 - 2*phi^2);
```

```
## strength of dens depend
beta_inv ~ dnorm(0, 1e-9) T(0,);
beta <- 1/beta_inv;

## covariate effects
for(i in 1:n_cov) { gamma[i] ~ dnorm(0,0.01) }

## AR(1) coef for proc errors
#phi ~ dunif(-0.999,0.999);
#phi <- 0;
phi_prior ~ dbeta(2,2);
phi <- phi_prior*2-1;
#phi ~ dunif(0,0.999);

## innovation in first year
innov_1 ~ dnorm(0,tau_r*(1-phi*phi));

## process variance for recruits model
sigma_r ~ dnorm(0, 2e-2) T(0,);
tau_r <- 1/sigma_r;

## obs variance for spawners
tau_s <- 1/sigma_s;
sigma_s ~ dnorm(0, 0.001) T(0,);

## unprojectable early recruits;
## hyper mean across all popns
Rec_mu ~ dnorm(0,0.001);
## hyper SD across all popns
Rec_sig ~ dunif(0,100);
## precision across all popns
Rec_tau <- pow(Rec_sig,-2);
## multipliers for unobservable total runs
ttl_run_mu ~ dunif(1,5);
ttl_run_tau ~ dunif(1,20);

## get total cal yr returns for first age_min yrs
for(i in 1:(age_min+age_skip)) {
ln_tot_Run[i] ~ dnorm(ttl_run_mu*Rec_mu,Rec_tau/ttl_run_tau);
tot_Run[i] <- exp(ln_tot_Run[i]);
}

## maturity schedule
## unif vec for Dirch prior
theta <- c(1,10,10,5,1,1)
## hyper-mean for maturity
pi_eta ~ ddirch(theta);
## hyper-prec for maturity
pi_tau ~ dnorm(0, 0.01) T(0,);
for(t in 1:(n_yrs-age_min+n_fore)) { pi_vec[t,1:A] ~ ddirch(pi_eta*pi_tau) }

## estimated harvest rate
for(t in 1:(n_yrs+n_fore)) { h_rate[t] ~ dunif(0,1) }
```

```
##------------
## LIKELIHOOD
##------------
## predicted recruits in BY t
covar[1] <- inprod(gamma,mod_cvrs[1,]);
ln_BH_a[1] <- mu_BH_a + covar[1];
E_ln_Rec[1] <- ln_BH_a[1] + ln_Sp[1] - log(1 + beta*Sp[1]) + phi*innov_1;
tot_ln_Rec[1] ~ dnorm(E_ln_Rec[1],tau_r);
res_ln_Rec[1] <- tot_ln_Rec[1] - E_ln_Rec[1];
w[1] <- phi * innov_1 + res_ln_Rec[1];

## median of total recruits
tot_Rec[1] <- exp(tot_ln_Rec[1]);

## R/S
ln_RS[1] <- tot_ln_Rec[1] - ln_Sp[1];

## brood-yr recruits by age
for(a in 1:A) {
Rec[1,a] <- tot_Rec[1] * pi_vec[1,a];
}

## brood years 2:(n_yrs-age_min)
for(t in 2:(n_yrs-age_min+n_fore)) {
## predicted recruits in BY t
covar[t] <- inprod(gamma, mod_cvrs[t,]);
ln_BH_a[t] <- mu_BH_a + covar[t];
E_ln_Rec[t] <- ln_BH_a[t] + ln_Sp[t] - log(1 + beta*Sp[t]) + phi*res_ln_Rec[t-1];
tot_ln_Rec[t] ~ dnorm(E_ln_Rec[t],tau_r);
res_ln_Rec[t] <- tot_ln_Rec[t] - E_ln_Rec[t];
w[t] <- phi * res_ln_Rec[t-1] + res_ln_Rec[t];

## median of total recruits
tot_Rec[t] <- exp(tot_ln_Rec[t]);

## R/S
ln_RS[t] <- tot_ln_Rec[t] - ln_Sp[t];

## brood-yr recruits by age
for(a in 1:A) {
Rec[t,a] <- tot_Rec[t] * pi_vec[t,a];
}
} ## end t loop over year

## get predicted calendar year returns by age
## matrix Run has dim [(n_yrs-age_min) x A]
## step 1: incomplete early broods
## first cal yr of this grp is first brood yr + age_min + age_skip

for(i in 1:(age_max-age_min-age_skip)) {
## projected recruits
for(a in 1:(i+age_skip)) {
Run[i,a] <- Rec[(age_skip+i)-a+1,a];
```

```
}

## imputed recruits
for(a in (i+1+age_skip):A) {
lnRec[i,a] ~ dnorm(Rec_mu,Rec_tau);
Run[i,a] <- exp(lnRec[i,a]);
}

## total run size
tot_Run[i+age_min+age_skip] <- sum(Run[i,1:A]);

## predicted age-prop vec for multinom
for(a in 1:A) {
age_v[i,a] <- Run[i,a] / tot_Run[i+age_min];
}

## multinomial for age comp
dat_age[i,1:A] ~ dmulti(age_v[i,1:A],dat_age[i,A+1]);
lp_age[i] <- logdensity.multi(dat_age[i,1:A],age_v[i,1:A],dat_age[i,A+1]);
}

## step 2: info from complete broods
## first cal yr of this grp is first brood yr + age_max
for(i in (A-age_skip):(n_yrs-age_min-age_skip+n_fore)) {
for(a in 1:A) {
Run[i,a] <- Rec[(age_skip+i)-a+1,a];
}

## total run size
tot_Run[i+age_min+age_skip] <- sum(Run[i,1:A]);

## predicted age-prop vec for multinom
for(a in 1:A) {
age_v[i,a] <- Run[i,a] / tot_Run[i+age_min];
}

## multinomial for age comp
dat_age[i,1:A] ~ dmulti(age_v[i,1:A],dat_age[i,A+1]);
lp_age[i] <- ifelse(i < n_yrs-age_min-age_skip+n_fore, logdensity.multi(dat_age[i,1:A],age_v[i,1:A]
}

## get predicted calendar year spawners
## first cal yr is first brood yr
for(t in 1:(n_yrs+n_fore)) {
## obs model for spawners
#Sp[t] <- max(10,tot_Run[t] - dat_harv[t]);
est_harv[t] = ifelse(t > n_yrs,1,h_rate[t] * tot_Run[t]);

dat_harv[t] ~ dlnorm(log(est_harv[t]), 20);
Sp[t] = tot_Run[t] - est_harv[t];
ln_Sp[t] <- log(Sp[t]);
ln_dat_esc[t] ~ dnorm(ln_Sp[t], tau_s);
```

```
    lp_esc[t] <- ifelse(t < n_yrs + 1,logdensity.norm(ln_dat_esc[t],ln_Sp[t], tau_s),0);
    }
  } ## end model description

  ", file=file.path(jagsdir, "IPM_BH_cov_AR.txt"))
```

Beverton-Holt with covars

```
cat("
model {

  ##--------
  ## PRIORS
  ##--------
  ## alpha = intrinsic productivity
  alpha ~ dnorm(0,0.001) T(0,);
  mu_BH_a <- log(alpha);
  E_BH_a <- mu_BH_a + sigma_r/(2 - 2*phi^2);

  ## strength of dens depend
  beta_inv ~ dnorm(0, 1e-9) T(0,);
  beta <- 1/beta_inv;

  ## covariate effects
  for(i in 1:n_cov) { gamma[i] ~ dnorm(0,0.01) }

  ## AR(1) coef for recruitment residual
  #phi ~ dunif(-0.999,0.999);
  #phi <- 0;
  phi_prior ~ dbeta(2,2);
  phi <- phi_prior*2-1;
  #phi ~ dunif(0,0.999);

  ## MA(1) coef recruitment residual
  theta_res_prior ~ dbeta(2,2);
  theta_res <- theta_res_prior*2-1;
  #theta_res ~ dunif(0,0.999);

  ## innovation in first year
  #innov_1 ~ dnorm(0,tau_r*(1-phi*phi));#AR1
  innov_1 ~ dnorm(0,(1-phi^2)/((1+2*phi*theta_res+theta_res^2)*sigma_r^2));#AR1MA1

  ## process variance for recruits model
  sigma_r ~ dnorm(0, 2e-2) T(0,);
  tau_r <- 1/sigma_r;

  ## obs variance for spawners
  tau_s <- 1/sigma_s;
  sigma_s ~ dnorm(0, 0.001) T(0,);

  ## unprojectable early recruits;
```

```
## hyper mean across all popns
Rec_mu ~ dnorm(0,0.001);
## hyper SD across all popns
Rec_sig ~ dunif(0,100);
## precision across all popns
Rec_tau <- pow(Rec_sig,-2);
## multipliers for unobservable total runs
  ttl_run_mu ~ dunif(1,5);
  ttl_run_tau ~ dunif(1,20);

## get total cal yr returns for first age_min yrs
for(i in 1:(age_min+age_skip)) {
      ln_tot_Run[i] ~ dnorm(ttl_run_mu*Rec_mu,Rec_tau/ttl_run_tau);
      tot_Run[i] <- exp(ln_tot_Run[i]);
}

## maturity schedule
## unif vec for Dirch prior
theta <- c(1,10,10,5,1,1)
## hyper-mean for maturity
pi_eta ~ ddirch(theta);
## hyper-prec for maturity
pi_tau ~ dnorm(0, 0.01) T(0,);
for(t in 1:(n_yrs-age_min+n_fore)) { pi_vec[t,1:A] ~ ddirch(pi_eta*pi_tau) }

## estimated harvest rate
for(t in 1:(n_yrs+n_fore)) { h_rate[t] ~ dunif(0,1) }

##------------
## LIKELIHOOD
##------------
## predicted recruits in BY t
covar[1] <- inprod(gamma,mod_cvrs[1,]);
ln_BH_a[1] <- mu_BH_a + covar[1];
E_ln_Rec[1] <- ln_BH_a[1] + ln_Sp[1] - log(1 + beta*Sp[1]) + phi * innov_1 + theta_res * 0;
tot_ln_Rec[1] ~ dnorm(E_ln_Rec[1], tau_r);
res_ln_Rec[1] <- tot_ln_Rec[1] - E_ln_Rec[1];
w[1] <- phi * innov_1 + theta_res * 0 + res_ln_Rec[1]

## median of total recruits
tot_Rec[1] <- exp(tot_ln_Rec[1]);

## R/S
ln_RS[1] <- tot_ln_Rec[1] - ln_Sp[1];

## brood-yr recruits by age
for(a in 1:A) {
  Rec[1,a] <- tot_Rec[1] * pi_vec[1,a];
}

## brood years 2:(n_yrs-age_min)
for(t in 2:(n_yrs-age_min+n_fore)) {
  ## predicted recruits in BY t
```

```
  covar[t] <- inprod(gamma, mod_cvrs[t,]);
  ln_BH_a[t] <- mu_BH_a + covar[t];

  #=============================================
  #version 4; more similar to AR1 original model
  #=============================================
  E_ln_Rec[t] <- ln_BH_a[t] + ln_Sp[t] - log(1 + beta*Sp[t]) + phi * w[t-1] + theta_res * res_ln_Rec[
  tot_ln_Rec[t] ~ dnorm(E_ln_Rec[t], tau_r);
  res_ln_Rec[t] <- tot_ln_Rec[t] - E_ln_Rec[t];
  w[t] <- phi * w[t-1] + theta_res * res_ln_Rec[t-1] + res_ln_Rec[t];


  ## median of total recruits
  tot_Rec[t] <- exp(tot_ln_Rec[t]);
  ## R/S
  ln_RS[t] <- tot_ln_Rec[t] - ln_Sp[t];
  ## brood-yr recruits by age
  for(a in 1:A) {
    Rec[t,a] <- tot_Rec[t] * pi_vec[t,a];
  }
} ## end t loop over year

## get predicted calendar year returns by age
## matrix Run has dim [(n_yrs-age_min) x A]
## step 1: incomplete early broods
## first cal yr of this grp is first brood yr + age_min + age_skip
for(i in 1:(age_max-age_min-age_skip)) {
  ## projected recruits
  for(a in 1:(i+age_skip)) {
    Run[i,a] <- Rec[(age_skip+i)-a+1,a];
  }
  ## imputed recruits
  for(a in (i+1+age_skip):A) {
    lnRec[i,a] ~ dnorm(Rec_mu,Rec_tau);
    Run[i,a] <- exp(lnRec[i,a]);
  }
  ## total run size
  tot_Run[i+age_min+age_skip] <- sum(Run[i,1:A]);
  ## predicted age-prop vec for multinom
  for(a in 1:A) {
    age_v[i,a] <- Run[i,a] / tot_Run[i+age_min];
  }
  ## multinomial for age comp
  dat_age[i,1:A] ~ dmulti(age_v[i,1:A],dat_age[i,A+1]);
  lp_age[i] <- logdensity.multi(dat_age[i,1:A],age_v[i,1:A],dat_age[i,A+1]);
}

## step 2: info from complete broods
## first cal yr of this grp is first brood yr + age_max
for(i in (A-age_skip):(n_yrs-age_min-age_skip+n_fore)) {
  for(a in 1:A) {
    Run[i,a] <- Rec[(age_skip+i)-a+1,a];
  }
```

```
    ## total run size
    tot_Run[i+age_min+age_skip] <- sum(Run[i,1:A]);
    ## predicted age-prop vec for multinom
    for(a in 1:A) {
      age_v[i,a] <- Run[i,a] / tot_Run[i+age_min];
    }
    ## multinomial for age comp
    dat_age[i,1:A] ~ dmulti(age_v[i,1:A],dat_age[i,A+1]);
    #lp_age[i] <- logdensity.multi(dat_age[i,1:A],age_v[i,1:A],dat_age[i,A+1]);
    lp_age[i] <- ifelse(i < n_yrs-age_min-age_skip+n_fore,
    logdensity.multi(dat_age[i,1:A],age_v[i,1:A],dat_age[i,A+1]),0)
  }

  ## get predicted calendar year spawners
  ## first cal yr is first brood yr
  for(t in 1:(n_yrs+n_fore)) {
    ## obs model for spawners
    # Sp[t] <- max(10,tot_Run[t] - dat_harv[t]);
    est_harv[t] = h_rate[t] * tot_Run[t];
    dat_harv[t] ~ dlnorm(log(est_harv[t]), 20);
    Sp[t] = tot_Run[t] - est_harv[t];
    ln_Sp[t] <- log(Sp[t]);
    ln_dat_esc[t] ~ dnorm(ln_Sp[t], tau_s);
    lp_esc[t] <- ifelse(t < n_yrs + 1,logdensity.norm(ln_dat_esc[t],ln_Sp[t], tau_s),0);
  }

} ## end model description

  ", file=file.path(jagsdir, "IPM_BH_cov_MA1_AR1.txt"))
```

Beverton-Holt with covars

```
cat("
model {

  ##--------
  ## PRIORS
  ##--------
  ## alpha = intrinsic productivity
  alpha ~ dnorm(0,0.001) T(0,);
  mu_BH_a <- log(alpha);
  E_BH_a <- mu_BH_a + sigma_r/(2 - 2*phi^2);

  ## strength of dens depend
  beta_inv ~ dnorm(0, 1e-9) T(0,);
  beta <- 1/beta_inv;

  ## covariate effects
  for(i in 1:n_cov) { gamma[i] ~ dnorm(0,0.01) }

  ## AR(1) coef for recruitment residual
```

```
#phi ~ dunif(-0.999,0.999);
#phi <- 0;
phi_prior ~ dbeta(2,2);
phi <- phi_prior*2-1;
#phi ~ dunif(0,0.999);

## innovation in first year
innov_1 ~ dnorm(0,tau_r*(1-phi*phi));#AR1

## process variance for recruits model
sigma_r ~ dnorm(0, 2e-2) T(0,);
tau_r <- 1/sigma_r;

## obs variance for spawners
tau_s <- 1/sigma_s;
sigma_s ~ dnorm(0, 0.001) T(0,);

## unprojectable early recruits;
## hyper mean across all popns
Rec_mu ~ dnorm(0,0.001);
## hyper SD across all popns
Rec_sig ~ dunif(0,100);
## precision across all popns
Rec_tau <- pow(Rec_sig,-2);
## multipliers for unobservable total runs
  ttl_run_mu ~ dunif(1,5);
  ttl_run_tau ~ dunif(1,20);

## get total cal yr returns for first age_min yrs
for(i in 1:(age_min+age_skip)) {
      ln_tot_Run[i] ~ dnorm(ttl_run_mu*Rec_mu,Rec_tau/ttl_run_tau);
      tot_Run[i] <- exp(ln_tot_Run[i]);
}

## maturity schedule
## unif vec for Dirch prior
theta <- c(1,10,10,5,1,1)
## hyper-mean for maturity
pi_eta ~ ddirch(theta);
## hyper-prec for maturity
pi_tau ~ dnorm(0, 0.01) T(0,);
for(t in 1:(n_yrs-age_min+n_fore)) { pi_vec[t,1:A] ~ ddirch(pi_eta*pi_tau) }

## estimated harvest rate
for(t in 1:(n_yrs+n_fore)) { h_rate[t] ~ dunif(0,1) }

##------------
## LIKELIHOOD
##------------
## predicted recruits in BY t
covar[1] <- inprod(gamma,mod_cvrs[1,]);
ln_BH_a[1] <- mu_BH_a + covar[1];
E_ln_Rec[1] <- ln_BH_a[1] + ln_Sp[1] - log(1 + beta*Sp[1]) + phi * innov_1;
```

```
tot_ln_Rec[1] ~ dnorm(E_ln_Rec[1], tau_r);
res_ln_Rec[1] <- tot_ln_Rec[1] - E_ln_Rec[1];
w[1] <- phi * innov_1 + res_ln_Rec[1];

## median of total recruits
tot_Rec[1] <- exp(tot_ln_Rec[1]);

## R/S
ln_RS[1] <- tot_ln_Rec[1] - ln_Sp[1];

## brood-yr recruits by age
for(a in 1:A) {
  Rec[1,a] <- tot_Rec[1] * pi_vec[1,a];
}

## brood years 2:(n_yrs-age_min)
for(t in 2:(n_yrs-age_min+n_fore)) {
  ## predicted recruits in BY t
  covar[t] <- inprod(gamma, mod_cvrs[t,]);
  ln_BH_a[t] <- mu_BH_a + covar[t];
  E_ln_Rec[t] <- ln_BH_a[t] + ln_Sp[t] - log(1 + beta*Sp[t]) + phi * w[t-1];
  tot_ln_Rec[t] ~ dnorm(E_ln_Rec[t], tau_r);
  res_ln_Rec[t] <- tot_ln_Rec[t] - E_ln_Rec[t];
  w[t] <- phi * w[t-1] + res_ln_Rec[t];

  ## median of total recruits
  tot_Rec[t] <- exp(tot_ln_Rec[t]);
  ## R/S
  ln_RS[t] <- tot_ln_Rec[t] - ln_Sp[t];
  ## brood-yr recruits by age
  for(a in 1:A) {
    Rec[t,a] <- tot_Rec[t] * pi_vec[t,a];
  }
} ## end t loop over year

## get predicted calendar year returns by age
## matrix Run has dim [(n_yrs-age_min) x A]
## step 1: incomplete early broods
## first cal yr of this grp is first brood yr + age_min + age_skip
for(i in 1:(age_max-age_min-age_skip)) {
  ## projected recruits
  for(a in 1:(i+age_skip)) {
    Run[i,a] <- Rec[(age_skip+i)-a+1,a];
  }
  ## imputed recruits
  for(a in (i+1+age_skip):A) {
    lnRec[i,a] ~ dnorm(Rec_mu,Rec_tau);
    Run[i,a] <- exp(lnRec[i,a]);
  }
  ## total run size
  tot_Run[i+age_min+age_skip] <- sum(Run[i,1:A]);
  ## predicted age-prop vec for multinom
  for(a in 1:A) {
```

```
      age_v[i,a] <- Run[i,a] / tot_Run[i+age_min];
    }
    ## multinomial for age comp
    dat_age[i,1:A] ~ dmulti(age_v[i,1:A],dat_age[i,A+1]);
    lp_age[i] <- logdensity.multi(dat_age[i,1:A],age_v[i,1:A],dat_age[i,A+1]);
  }

  ## step 2: info from complete broods
  ## first cal yr of this grp is first brood yr + age_max
  for(i in (A-age_skip):(n_yrs-age_min-age_skip+n_fore)) {
    for(a in 1:A) {
      Run[i,a] <- Rec[(age_skip+i)-a+1,a];
    }
    ## total run size
    tot_Run[i+age_min+age_skip] <- sum(Run[i,1:A]);
    ## predicted age-prop vec for multinom
    for(a in 1:A) {
      age_v[i,a] <- Run[i,a] / tot_Run[i+age_min];
    }
    ## multinomial for age comp
    dat_age[i,1:A] ~ dmulti(age_v[i,1:A],dat_age[i,A+1]);
    #lp_age[i] <- logdensity.multi(dat_age[i,1:A],age_v[i,1:A],dat_age[i,A+1]);
    lp_age[i] <- ifelse(i < n_yrs-age_min-age_skip+n_fore,
    logdensity.multi(dat_age[i,1:A],age_v[i,1:A],dat_age[i,A+1]),0)
  }

  ## get predicted calendar year spawners
  ## first cal yr is first brood yr
  for(t in 1:(n_yrs+n_fore)) {
    ## obs model for spawners
    # Sp[t] <- max(10,tot_Run[t] - dat_harv[t]);
    est_harv[t] = h_rate[t] * tot_Run[t];
    dat_harv[t] ~ dlnorm(log(est_harv[t]), 20);
    Sp[t] = tot_Run[t] - est_harv[t];
    ln_Sp[t] <- log(Sp[t]);
    ln_dat_esc[t] ~ dnorm(ln_Sp[t], tau_s);
    lp_esc[t] <- ifelse(t < n_yrs + 1,logdensity.norm(ln_dat_esc[t],ln_Sp[t], tau_s),0);
  }

} ## end model description

  ", file=file.path(jagsdir, "IPM_BH_cov_AR_resid.txt"))
```

Fitting the models and generating the one year ahead forecasts

Before fitting the model in JAGS, we need to specify the MCMC control parameters.

```
## 1. MCMC control params
mcmc_ctrl <- list(
```

```
  chains = 4,
  length = 200000,#5e5,
  burn = 100000,#2e5,
  thin = 100#400
)
## total number of MCMC samples after burnin
mcmc_samp <- mcmc_ctrl$length*mcmc_ctrl$chains/mcmc_ctrl$thin
```

```
## empty list for fits
n_mods <- 3
## empty list for fits
mod_fits <- vector("list", n_mods*(n_forecasts+1))
## models
models <- c("IPM_BH_cov_MA1_AR1","IPM_BH_cov_AR","IPM_BH_cov_AR_resid")

## counter to index fitted jags models (33 in total: 3 models x 11 1 year ahead forecasts including upc
## return year)
t <- 1

for(n in 1:n_mods){
  ## counter to index data to feed model for year specific forecasts
  ## first forecast will be for 10 years prior to the most recent return year;
  ## last forecast will be current forecast for the upcoming return year
  c <- 0
  #n <-2
  model <- models[n]

  for(i in 1:(n_forecasts+1)){
    if(file.exists(file.path(savedir, paste(model,"_","y",i,".rds",sep = "")))) {
      mod_fits[[t]] <- readRDS(file.path(savedir, paste(model,"_","y",i,".rds",sep = "")))
      c <- c + 1
      t <- t + 1
  } else { ## else, fit & save
        ## cnt & time stamp
        cat("Count =", t, "; Time =", round(((proc.time()-timer_start)/60)["elapsed"], 1), "\n",
            file="cnt_time.txt", append=TRUE)

        #range of years. Last year in range
        dat_yrs <- seq(yr_frst,(yr_last - n_forecasts + c),1)

        ## number of years of data
        n_yrs <- length(dat_yrs)


        ## get first & last years
        yr_frst_forecast <- min(dat_yrs)
        yr_last_forecast <- max(dat_yrs)

        ## get escapement data
        dat_esc_forecast <- dat_esc[which(dat_esc$year %in% dat_yrs),]

        ## log of escapement
        ln_dat_esc <- c(log(dat_esc_forecast$escapement),rep(NA,n_fore))
```

```r
## get age data
dat_age_forecast <- dat_age[which(dat_age$year %in% dat_yrs),]
## drop year col & first age_min+age_skip rows
dat_age_forecast <- dat_age_forecast[-(1:(age_min+age_skip)),-1]

## add row(s) of NA's for forecast years
if(n_fore > 0) {
  dat_age_forecast <- rbind(dat_age_forecast,
                            matrix(0, n_fore, A,
                                   dimnames = list(n_yrs+seq(n_fore),colnames(dat_age_forecast))))
}
## total num of age obs by cal yr
dat_age_forecast[,"sum"] <- apply(dat_age_forecast, 1, sum)
## row indices for any years with no obs age comp
idx_NA_yrs <- which(dat_age_forecast$sum<A, TRUE)
## replace 0's in yrs w/o any obs with NA's
dat_age_forecast[idx_NA_yrs,(1:A)] <- NA
## change total in yrs w/o any obs from 0 to A to help dmulti()
dat_age_forecast[idx_NA_yrs,"sum"] <- A
## convert class
dat_age_forecast <- as.matrix(dat_age_forecast)

## get harvest data
dat_harv_forecast <- dat_harv[which(dat_harv$year %in% dat_yrs),]
## drop year col & first age_max rows
dat_harv_forecast <- c(dat_harv_forecast$catch,rep(NA,n_fore))

## get covariate data
dat_cvrs_forecast <- dat_cvrs[which(dat_cvrs$year <= yr_last + n_fore - age_min),1:4]
## drop year col
dat_cvrs_forecast <- dat_cvrs_forecast[,-1]
## transform the covariates to z-scores
scl_cvrs_forecast <- scale(dat_cvrs_forecast)
## total number of covariates
n_cov <- dim(dat_cvrs_forecast)[2]

## ----jags_setup-----------------------------------------------------------
## 1. Data to pass to JAGS
dat_jags <- list(dat_age = dat_age_forecast,
                 ln_dat_esc = ln_dat_esc,
                 dat_harv = dat_harv_forecast,
                 A = A,
                 age_min = age_min,
                 age_max = age_max,
                 age_skip = age_skip,
                 n_yrs = n_yrs,
                 n_fore = n_fore)

## 2. Model params/states for JAGS to return
##    These are specific to the process model,
##    so we define them in 'par_jags' below.
```

```
if(model == "IPM_BH_cov_AR"|model == "IPM_BH_cov_AR_resid"){
  init_vals_cov <- function() {
  list(alpha = 5,
       beta_inv = exp(mean(ln_dat_esc, na.rm = TRUE)),
       gamma = rep(0, 3),
       pi_tau = 10,
       pi_eta = rep(1,A),
       pi_vec = matrix(c(0.01,0.35,0.47,0.15,0.01,0.01),
                       n_yrs-age_min+n_fore, A,
                       byrow = TRUE),
       Rec_mu = log(1000),
       Rec_sig = 0.1,
       tot_ln_Rec = rep(log(1000), n_yrs - age_min + n_fore),
       phi_prior = 0.5,
       innov_1 = 0)
  }

  ## params/states to return
  par_jags<- c("alpha","E_BH_a","ln_BH_a",
               "beta",
               "gamma",
               "Sp","Rec","tot_ln_Rec","ln_RS",
               "pi_eta","pi_tau",
               "sigma_r","sigma_s","w","res_ln_Rec",
               "lp_age","lp_esc","phi"
               )

}else{
  init_vals_cov <- function() {
    list(alpha = 5,
         beta_inv = exp(mean(ln_dat_esc, na.rm = TRUE)),
         gamma = rep(0, 3),
         pi_tau = 10,
         pi_eta = rep(1,A),
         pi_vec = matrix(c(0.01,0.35,0.47,0.15,0.01,0.01),
                         n_yrs-age_min+n_fore, A,
                         byrow = TRUE),
         Rec_mu = log(1000),
         Rec_sig = 0.1,
         tot_ln_Rec = rep(log(1000), n_yrs - age_min + n_fore),
         phi_prior = 0.75,theta_res_prior = 0.75,
         innov_1 = 0)
    }

  ## params/states to return
  par_jags <- c("alpha","E_BH_a","ln_BH_a",
               "beta",
               "gamma",
               "Sp","Rec","tot_ln_Rec","ln_RS",
               "pi_eta","pi_tau",
               "sigma_r","sigma_s","res_ln_Rec","w","theta_res","phi",
               "lp_age","lp_esc"
               )
```

```
        }#endif



        ## set of multi-covariate models
        cset <- colnames(scl_cvrs_forecast)
        dat_jags$n_cov <- length(cset)
        dat_jags$mod_cvrs <- scl_cvrs_forecast[, cset]


        ## fit model & save it
        # mod_fits[[t]] <- fit_jags(paste(model,".txt",sep = ""), dat_jags, par_jags,
        #                           init_vals_cov, mcmc_ctrl)
        mod_fits[[t]]<-fit_jags2(model=paste(model,".txt",sep = ""),
                          data=dat_jags,
                          params=par_jags,
                          inits=init_vals_cov,
                          ctrl=mcmc_ctrl
                          )
        saveRDS(mod_fits[[t]], file.path(savedir,paste(model,"_","y",i,".rds",sep = "")))

        c <- c + 1
        t <- t + 1
    }## end if

  }##next forecast year(i)
}## next model(n)
```

Save the output for all of the forecasts.

```
# save(mod_fits, file = file.path(savedir, "forecasts.rda"))
```

Model diagnostics

Here is a table of the Gelman & Rubin statistics (R_{hat}) for the estimated parameters. Recall that we set an upper threshold of 1.1, so values larger than that deserve some additional inspection.

```
## params of interest
par_conv <- c("alpha","beta",paste0("gamma[",seq(3),"]"),
              "sigma_r","sigma_s","pi_tau","theta_res",paste0("pi_eta[",seq(A-1),"]"))
## Gelman-Rubin
#gelman.diag(mod_fits[[1]][,par_conv])
## Autocorrelation
# t(round(autocorr.diag(mod_fits[[1]][,par_conv],
#                       lags = seq(mcmc_ctrl$thin, 4*mcmc_ctrl$thin, mcmc_ctrl$thin),
#                       relative=FALSE), 2))
## Use ShinyStan to look at effective draws, Gelman-Rubin, Autocorrelation
fit_bh_cov_MA1_AR1 <- readRDS(file.path(savedir,"IPM_BH_cov_MA1_AR1_y11.rds"))
my_sso2 <- launch_shinystan(as.shinystan(fit_bh_cov_MA1_AR1))
summary_stats2<-data.frame(lapply(c("rhat","neff","mean","sd","quantiles"),function(x) retrieve(my_sso2
```

```
colnames(summary_stats2)[1:4]<-c("rhat","neff","mean","sd")
write.csv(summary_stats2,file.path(savedir,"Summary_stats_AR1_MA1.csv"))
```

Model diagnostics

Here is a table of the Gelman & Rubin statistics (R_{hat}) for the estimated parameters. Recall that we set an upper threshold of 1.1, so values larger than that deserve some additional inspection.

```
## params of interest
par_conv <- c("alpha","beta",paste0("gamma[",seq(3),"]"),
              "sigma_r","sigma_s","pi_tau","phi",paste0("pi_eta[",seq(A-1),"]"))
## Gelman-Rubin
#gelman.diag(mod_fits[[2]][,par_conv])
## Autocorrelation
# t(round(autocorr.diag(mod_fits[[2]][,par_conv],
#                       lags = seq(mcmc_ctrl$thin, 4*mcmc_ctrl$thin, mcmc_ctrl$thin),
#                       relative=FALSE), 2))
## Use ShinyStan to look at effective draws, Gelman-Rubin, Autocorrelation
fit_bh_cov_AR <- readRDS(file.path(savedir,"IPM_BH_cov_AR_y11.rds"))
my_sso <- launch_shinystan(as.shinystan(fit_bh_cov_AR))
summary_stats1<-data.frame(lapply(c("rhat","neff","mean","sd","quantiles"),function(x) retrieve(my_sso,
colnames(summary_stats1)[1:4]<-c("rhat","neff","mean","sd")
write.csv(summary_stats1,file.path(savedir,"Summary_stats_AR.csv"))
```

Model diagnostics AR1 recruitment residuals

Here is a table of the Gelman & Rubin statistics (R_{hat}) for the estimated parameters. Recall that we set an upper threshold of 1.1, so values larger than that deserve some additional inspection.

```
## params of interest
par_conv <- c("alpha","beta",paste0("gamma[",seq(3),"]"),
              "sigma_r","sigma_s","pi_tau","phi",paste0("pi_eta[",seq(A-1),"]"))
## Gelman-Rubin
#gelman.diag(mod_fits[[3]][,par_conv])
## Autocorrelation
# t(round(autocorr.diag(mod_fits[[3]][,par_conv],
#                       lags = seq(mcmc_ctrl$thin, 4*mcmc_ctrl$thin, mcmc_ctrl$thin),
#                       relative=FALSE), 2))
## Use ShinyStan to look at effective draws, Gelman-Rubin, Autocorrelation
fit_bh_cov_AR_resid <- readRDS(file.path(savedir,"IPM_BH_cov_AR_resid_y11.rds"))
my_sso <- launch_shinystan(as.shinystan(fit_bh_cov_AR_resid))
summary_stats3<-data.frame(lapply(c("rhat","neff","mean","sd","quantiles"),function(x) retrieve(my_sso,
colnames(summary_stats3)[1:4]<-c("rhat","neff","mean","sd")
write.csv(summary_stats3,file.path(savedir,"Summary_stats_AR_resid.csv"))
```

Model selection

```r
tot_mods <- n_forecasts*n_mods


# get escapement data
dat_esc_forecast <- dat_esc[which(dat_esc$year %in% seq(yr_begin,yr_end,1)),]

## get harvest data
dat_harv_forecast <- dat_harv[which(dat_harv$year %in% seq(yr_begin,yr_end,1)),]

## observed terminal run size
obs_trs <- dat_esc_forecast$escapement + dat_harv_forecast$catch

mod_res<- do.call("rbind", mod_fits)

c <-1
pred_trs <- NULL
for(n in 1:n_mods){
  #n <- 1
  pred_esc <- NULL
  for(i in 1:(n_forecasts)){
    #i <- 1
  p_dat <- mod_res[[c]][,grep("Sp", colnames(mod_res[[c]]))]
  p_dat <- round(median(p_dat[,dim(p_dat)[2]]))

  pred_esc[i] <- p_dat

  c <- c+1

  }

  pred_trs_mod <- pred_esc + dat_harv_forecast$catch

  pred_trs <- cbind(pred_trs,pred_trs_mod)
  #names(pred_trs) <- paste(models[n],"_","pred_trs",sep = "")

}

colnames(pred_trs) <- models

## compute model performance statistics
Error <- obs_trs - pred_trs
SE <- Error^2
PE <- Error/obs_trs
APE <- abs(PE)


RMSE <- apply(SE,2,function(x){sqrt(mean(x))})
MPE <- apply(PE,2,function(x){mean(x)})
MAPE <- apply(APE,2,function(x){mean(x)})

model_selection <- cbind(RMSE,MPE,MAPE)
model_selection <- data.frame(model_selection)
model_selection[,"RMSE_weight"] <- (1/model_selection[,"RMSE"])/sum(1/model_selection[,"RMSE"])
```

```
## extract median 2020 forecast from each model
n_yrs <- length(dat_yrs)

fit_bh_cov_MA1_AR1 <- readRDS(file.path(savedir, "IPM_BH_cov_MA1_AR1_y11.rds"))
fit_bh_cov_AR1 <- readRDS(file.path(savedir, "IPM_BH_cov_AR_y11.rds"))
fit_bh_cov_resid <- readRDS(file.path(savedir, "IPM_BH_cov_AR_resid_y11.rds"))

mod_res_MA1_AR1 <- do.call("rbind", fit_bh_cov_MA1_AR1)
mod_res_AR1 <- do.call("rbind", fit_bh_cov_AR1)
mod_res_AR1_resid <- do.call("rbind", fit_bh_cov_resid)


p_dat_AR1MA1 <- mod_res_MA1_AR1[,grep("Sp", colnames(mod_res_MA1_AR1))]
p_dat_AR1MA1_2020 <- round(quantile(p_dat_AR1MA1[,n_yrs+n_fore],probs = c(0.025,0.5,0.975)))

p_dat_AR1 <- mod_res_AR1[,grep("Sp", colnames(mod_res_AR1))]
p_dat_AR1_2020 <- round(quantile(p_dat_AR1[,n_yrs+n_fore],probs = c(0.025,0.5,0.975)))

p_dat_AR1_resid <- mod_res_AR1_resid[,grep("Sp", colnames(mod_res_AR1_resid))]
mod_res_AR1_resid_2020 <- round(quantile(p_dat_AR1_resid[,n_yrs+n_fore],probs = c(0.025,0.5,0.975)))

model_selection[,"2020_forecast"] <- c(p_dat_AR1MA1_2020[2],p_dat_AR1_2020[2],mod_res_AR1_resid_2020[2])

weighted_forecast <- sum(model_selection[,"2020_forecast"]*model_selection[,"RMSE_weight"])

print(model_selection)
```

```
##                          RMSE        MPE      MAPE RMSE_weight 2020_forecast
## IPM_BH_cov_MA1_AR1  3585.031  0.4431874 0.4431874   0.2840495          2274
## IPM_BH_cov_AR       2339.910 -0.1004097 0.3727213   0.4351989          5094
## IPM_BH_cov_AR_resid 3627.144  0.3329551 0.3376668   0.2807516          2158
```

Via `loo()` and `compare()` with full table of results. Note that `elpd_diff` will be negative (positive) if the expected predictive accuracy for the first (second) model is higher.

```
LOOIC <- vector("list", n_mods)
model_indices <- c(11,22,33)
## extract log densities from JAGS objects
for(i in 1:n_mods) {
  #i <- 1
  ## convert mcmc.list to matrix
  tmp_lp <- as.matrix(mod_fits[[model_indices[i]]])
  ## extract pointwise likelihoods
  tmp_lp <- tmp_lp[,grepl("lp_", colnames(tmp_lp))]
  ## if numerical underflows, convert -Inf to 5% less than min(likelihood)
  if(any(is.infinite(tmp_lp))) {
    tmp_lp[is.infinite(tmp_lp)] <- NA
    tmp_min <- min(tmp_lp, na.rm = TRUE)
    tmp_lp[is.na(tmp_lp)] <- tmp_min * 1.05
  }
  ## calculate LOOIC
```

```
  LOOIC[[i]] <- loo(tmp_lp)
}

## compute pseudo weights
model_weights <- loo_model_weights(LOOIC, method = "pseudobma",optim_method = "BFGS", optim_control = li

## LOOIC for all data
tbl_LOOIC <- round(loo_compare(x = LOOIC), 2)
rownames(tbl_LOOIC) <- sub("model", "", rownames(tbl_LOOIC))
tbl_LOOIC <- tbl_LOOIC[order(as.numeric(rownames(tbl_LOOIC))), ]
tbl_LOOIC <- cbind(model = c("B-H","B-H","B-H"),
                   error = c("MA1_AR1","AR1","AR1_resid"),
                   as.data.frame(tbl_LOOIC),pseudo_bma_weight = as.matrix(model_weights))
tbl_LOOIC[order(tbl_LOOIC[,"looic"]), ]
```

```
##   model     error elpd_diff se_diff elpd_loo se_elpd_loo  p_loo se_p_loo  looic se_looic
## 2   B-H       AR1       0.0    0.00  -395.32       48.96 138.77    10.38 790.64    97.91
## 3   B-H AR1_resid      -3.7    5.07  -399.02       49.15 143.49    11.74 798.04    98.30
## 1   B-H   MA1_AR1      -4.4    5.33  -399.72       48.96 142.71    12.32 799.43    97.92
##   pseudo_bma_weight
## 2         0.6800957
## 3         0.1789055
## 1         0.1409988
```

```
## best model
best_i <- which(tbl_LOOIC[,"looic"] == min(tbl_LOOIC[,"looic"]))
best_fit <- mod_fits[[best_i]]
```

These results show that the Beverton-Holt model with AR1 error has the lowest LOOIC value. All results will be derived from model averaging based on pseudo bayesian model average weights.

Model Selection Via Approximate leave-future-out cross validation

Description here

```
#
# m=1
# N <- 43
# L <- 20
# k_thres <- 0.7
# approx_elpds_1sap <- rep(NA, N)
#
# # more stable than log(sum(exp(x)))
# log_sum_exp <- function(x) {
#   max_x <- max(x)
#   max_x + log(sum(exp(x - max_x)))
# }
#
# # more stable than log(mean(exp(x)))
# log_mean_exp <- function(x) {
#   log_sum_exp(x) - log(length(x))
```

```
# }
#
# # compute log of raw importance ratios
# # sums over observations *not* over posterior samples
# sum_log_ratios <- function(loglik, ids = NULL) {
#   if (!is.null(ids)) loglik <- loglik[, ids, drop = FALSE]
#   rowSums(loglik)
# }
#
# # for printing comparisons later
# rbind_print <- function(...) {
#   round(rbind(...), digits = 2)
# }
#
# #extract pontwise log likelihoods
# tmp_lp <- as.matrix(mod_fits[[m]])
# ## extract pointwise likelihoods
# tmp_lp <- tmp_lp[,grepl("lp_", colnames(tmp_lp))]
# ## if numerical underflows, convert -Inf to 5% less than min(likelihood)
# if(any(is.infinite(tmp_lp))) {
#   tmp_lp[is.infinite(tmp_lp)] <- NA
#   tmp_min <- min(tmp_lp, na.rm = TRUE)
#   tmp_lp[is.na(tmp_lp)] <- tmp_min * 1.05
# }
#
# tmp_lp<-tmp_lp[,grepl("esc", colnames(tmp_lp))]
# #get yrs assoc
# names_loglik<-data.frame(strsplit(colnames(tmp_lp),"\\[|\\]"))
# yrnames<-as.numeric(names_loglik[2,])
#
# loglik <- matrix(NA,ncol=N,nrow=dim(tmp_lp)[1])
# for(i in 1:N){
#   if(!is.null(ncol(tmp_lp[,yrnames==i]))){
#     loglik[,i] = apply(tmp_lp[,yrnames==i],1,sum)
#   }else(loglik[,i] = tmp_lp[,yrnames==i])
# }
#
# # initialize the process for i = L
# past <- 1:L
# oos <- L + 1
# # df_past <- df[past, , drop = FALSE]
# # df_oos <- df[c(past, oos), , drop = FALSE]
# # fit_past <- update(fit, newdata = df_past, recompile = FALSE)
# #loglik <- log_lik(fit_past, newdata = df_oos, oos = oos)
# approx_elpds_1sap[L + 1] <- log_mean_exp(loglik[, oos])
#
# # iterate over i > L
# i_refit <- L
# refits <- L
# ks <- NULL
# for (i in (L + 1):(N - 1)) {
#   past <- 1:i
#   oos <- i + 1
```

```
#   # df_past <- df[past, , drop = FALSE]
#   # df_oos <- df[c(past, oos), , drop = FALSE]
#   # loglik <- log_lik(fit_past, newdata = df_oos, oos = oos)
#
#   logratio <- sum_log_ratios(loglik, (i_refit + 1):i)
#   psis_obj <- suppressWarnings(psis(logratio))
#   k <- pareto_k_values(psis_obj)
#   ks <- c(ks, k)
#   # if (k > k_thres) {
#   #   # refit the model based on the first i observations
#   #   i_refit <- i
#   #   refits <- c(refits, i)
#   #   fit_past <- update(fit_past, newdata = df_past, recompile = FALSE)
#   #   loglik <- log_lik(fit_past, newdata = df_oos, oos = oos)
#   #   approx_elpds_1sap[i + 1] <- log_mean_exp(loglik[, oos])
#   # } else {
#     lw <- weights(psis_obj, normalize = TRUE)[, 1]
#     approx_elpds_1sap[i + 1] <- log_sum_exp(lw + loglik[, oos])
#   #}
# }
#
# approx_elpd_1sap <- sum(approx_elpds_1sap, na.rm = TRUE)
# print(paste("approx LFO =" ,approx_elpd_1sap))
# print(ks)
```