R2. Model fitting and evaluation

2020 - 2021 Skagit River steelhead forecast.

Contents

This is version 0.21.01.05.

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.

```r
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")
}
if(!require("rstan")) {
  install.packages("rstan")
  library("rstan")
}
if(!require("RColorBrewer")) {
  install.packages("RColorBrewer")
  library("RColorBrewer")
}

## 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){
  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)
}

#generate summary stats file from MCMC object
sum_stats<-function(mcmclist){
  ESS<-apply(as.matrix(mcmclist),2,ess_bulk)
  Rhat<-apply(as.matrix(mcmclist),2,Rhat)
```

```r
  summary_stats<-summary(mcmclist)
  summary_stats<-data.frame(summary_stats$statistics,summary_stats$quantiles,ESS,Rhat)
}

# functions for approximate LFO
# many functions modified from:
# https://github.com/paul-buerkner/LFO-CV-paper/blob/master/case-study-LFO-CV.Rmd

#load complete model fits & model refits with subset data
loadmodfits<-function(modelnames){
  mod_fits<-list(NULL)
  for(i in 1:length(modelnames)){
    mod_fits[[i]] <- readRDS(file.path(savedir,paste0(modelnames[i],"_y",n_forecasts+1,".rds")))
    #mod_fits[[i]] <- readRDS(file.path(savedir,paste0("fit_",modelnames[i],".rds")))
  }
  return(mod_fits)
}

#refits
loadrefits<-function(refitname,N,L){
  numrefits<-N-L+1
  re_fits<-list()
  for(i in 1:numrefits){
     re_fits[[i]] <- readRDS(file.path(savedir,paste0(refitname,"_y",i,".rds")))
  }
  return(re_fits)
}

# 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(ll, ids = NULL) {
  if (!is.null(ids)) ll <- ll[, ids, drop = FALSE]
  - rowSums(ll)
}
# for printing comparisons later
rbind_print <- function(...) {
  round(rbind(...), digits = 2)
}

#function to extract log likelihood from fitted model
extract_log_lik<-function(m,esc_only,N,mod_fits){
  #extract pontwise log likelihoods
```

```r
  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
  }
  if(esc_only =="Yes"){
    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])
  }
  return(loglik)
}


approx_LFO<-function(N=N,L,m=m,esc_only,mod_fits,userefits,refitname,thres){
  loglik =  extract_log_lik(m=m, esc_only = esc_only,N=N,mod_fits = mod_fits)
  ## look at Pareto k's
  k_LOOIC<-pareto_k_values(loo(loglik))[(L+1):N]
  if(userefits=="Yes"){
    re_fits =loadrefits(refitname=refitname,N=N,L=L)
  }
  i_refit <- L
  refits <- L
  ks <- NULL
  approx_elpds_1sap <- rep(NA, N)
  for (i in (N - 1):L) {
    logratio <- sum_log_ratios(loglik, (i + 1):N)
    psis_obj <- suppressWarnings(psis(logratio))
    k<-pareto_k_values(psis_obj)
    ks <- c(ks, k)
    if(k>thres & userefits=="Yes"){
    #use_refit of model based on the first[i] observations
      i_refit <- i
      refits <- c(refits, i)
      loglik =  extract_log_lik(m=(i+1)-L+1, esc_only = esc_only,N=N,mod_fits = re_fits)
      approx_elpds_1sap[i + 1] <- log_mean_exp(loglik[, i + 1])
    }else{
      lw <- weights(psis_obj, normalize = TRUE)[, 1]
      approx_elpds_1sap[i + 1] <- log_sum_exp(lw + loglik[, i + 1])
    }
  }
```

```r
  results<-list(approx_elpds_1sap,ks,k_LOOIC)
  names(results)<-c("LFO","ks","k_LOOIC")
  return(results)
}

plot_ks <- function(ks, thres = 0.7,N,L) {
  ids = N:(L + 1)
  dat_ks <- data.frame(ks = ks, ids = ids)
  ggplot(dat_ks, aes(x = ids, y = ks)) +
    geom_point(aes(color = ks > thres), shape = 3, show.legend = FALSE) +
    geom_hline(yintercept = thres, linetype = 2, color = "red2") +
    scale_color_manual(values = c("cornflowerblue", "darkblue")) +
    labs(x = "Data point", y = "Pareto k") +
    ylim(-0.5, max(dat_ks$ks))
}
#function for printing out a read text file
processFile = function(filepath) {
  con = file(filepath, "r")
  while ( TRUE ) {
    line = readLines(con, n = 1)
    if ( length(line) == 0 ) {
      break
    }
    cat(paste0(noquote(line)),"\n")
  }
  close(con)
}
```

User inputs

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

```r
## 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 and AR1 process errors (MA1 recruitment residuals). Here we will print out the model (contained in a separate text file)

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

```
##
##     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
##     }
##
##     ## 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
##
```

```
## Warning in readLines(con, n = 1): incomplete final line found on 'C:/Users/buehrtwb/OneDrive -
## Washington State Executive Branch Agencies/Documents/Scripts/Skagit-River-Steelhead-Forecast-Laptop/
## jags/IPM_BH_cov_AR.txt'
```

Beverton-Holt with covars and AR1MA1 recruitment residuals

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

```
##
## 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_R
##     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] = 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
##

## Warning in readLines(con, n = 1): incomplete final line found on 'C:/Users/buehrtwb/OneDrive -
## Washington State Executive Branch Agencies/Documents/Scripts/Skagit-River-Steelhead-Forecast-Laptop/
## jags/IPM_BH_cov_MA1_AR1.txt'
```

Beverton-Holt with covars and AR1 recruitment residuals

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

```
##
## 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] = 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
##

## Warning in readLines(con, n = 1): incomplete final line found on 'C:/Users/buehrtwb/OneDrive -
## Washington State Executive Branch Agencies/Documents/Scripts/Skagit-River-Steelhead-Forecast-Laptop/
## jags/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 = 2000,#200000,#5e5,
  burn = 1000,#100000,#2e5,
  thin = 1#100#400
)
## total number of MCMC samples after burnin
mcmc_samp <- mcmc_ctrl$length*mcmc_ctrl$chains/mcmc_ctrl$thin
```

```
## models
models <- c("IPM_BH_cov_MA1_AR1",
            "IPM_BH_cov_AR",
            "IPM_BH_cov_AR_resid",
            "IPM_BH_cov_MA1_AR1_age",
            "IPM_BH_cov_AR_age",
            "IPM_BH_cov_AR_resid_age"
            )
## empty list for fits
n_mods <- length(models)
## empty list for fits
mod_fits <- vector("list", n_mods*(n_forecasts+1))


## 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))


      ## 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 = 0.5,theta_res_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","tot_Run",
                 "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[1:(n_yrs-age_min+1), 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 = "")))
summary_stats<-NULL
```

```
        summary_stats<-sum_stats(mcmclist= mod_fits[[t]])
        write.csv(summary_stats,file.path(savedir, paste(model,"_","y",i,"_summary_stats.csv",sep = ""))
        c <- c + 1
        t <- t + 1
    }## end if

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

Model selection

```
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


pred_trs <- NULL
for(n in 1:n_mods){
  #n <- 1
  pred_esc <- NULL
  for(i in 1:(n_forecasts)){
    #i <- 1
  mod_res<-NULL
  mod_res<-as.matrix(readRDS(file.path(savedir,paste0(models[n],"_y",i,".rds"))))
  p_dat <- mod_res[,grep("Sp", colnames(mod_res))]
  p_dat <- round(median(p_dat[,dim(p_dat)[2]]))

  pred_esc[i] <- p_dat

  }

  pred_trs_mod <- pred_esc + 1#+ dat_harv_forecast$catch #you don't need to add catch in because it is

  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 <- pred_trs - obs_trs
SE <- Error^2
PE <- Error/obs_trs
```

```
APE <- abs(PE)
LAR <- log(obs_trs/pred_trs)


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)})
MSA <- apply(LAR,2,function(x){100*(exp(mean(abs(x))-1))})

model_selection <- data.frame(RMSE,MPE,MAPE,MSA)
weights<-apply(model_selection[,!colnames(model_selection)=="MPE"], 2,function(x) (1/x)/sum(1/x))
colnames(weights)<-paste0(colnames(weights),"_weight")
model_selection<-data.frame(model_selection,weights)
```

Model Selection Via Approximate Leave-Future-Out Cross Validation following the methods here: link.

```
N=yr_last-yr_frst+1
L=N-n_forecasts
thres=0.1
esc_only="No"
userefits="Yes"
mod_fits<-loadmodfits(modelnames=models)


LFO1<-approx_LFO(N=N,L=L,m=1,esc_only=esc_only,mod_fits=mod_fits,userefits=userefits,refitname=models[1
```

```
## Warning: Relative effective sample sizes ('r_eff' argument) not specified.
## For models fit with MCMC, the reported PSIS effective sample sizes and
## MCSE estimates will be over-optimistic.

## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
# plot_ks(LFO1$ks,N=N,L=L,thres=thres)
# plot_ks(LFO1$k_LOOIC,N=N,L=L,thres=thres)

LFO2<-approx_LFO(N=N,L=L,m=2,esc_only=esc_only,mod_fits=mod_fits,userefits=userefits,refitname=models[2
```

```
## Warning: Relative effective sample sizes ('r_eff' argument) not specified.
## For models fit with MCMC, the reported PSIS effective sample sizes and
## MCSE estimates will be over-optimistic.

## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
# plot_ks(LFO2$ks, N=N,L=L,thres=thres)
# plot_ks(LFO2$k_LOOIC, N=N,L=L,thres=thres)

LFO3<-approx_LFO(N=N,L=L,m=3,esc_only=esc_only,mod_fits=mod_fits,userefits=userefits,refitname=models[3
```

```
## Warning: Relative effective sample sizes ('r_eff' argument) not specified.
## For models fit with MCMC, the reported PSIS effective sample sizes and
## MCSE estimates will be over-optimistic.

## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
# plot_ks(LFO3$ks, N=N,L=L,thres=thres)
# plot_ks(LFO3$k_LOOIC, N=N,L=L,thres=thres)

LFO4<-approx_LFO(N=N,L=L,m=4,esc_only=esc_only,mod_fits=mod_fits,userefits=userefits,refitname=models[4]
```

```
## Warning: Relative effective sample sizes ('r_eff' argument) not specified.
## For models fit with MCMC, the reported PSIS effective sample sizes and
## MCSE estimates will be over-optimistic.

## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
# plot_ks(LFO3$ks, N=N,L=L,thres=thres)
# plot_ks(LFO3$k_LOOIC, N=N,L=L,thres=thres)

LFO5<-approx_LFO(N=N,L=L,m=5,esc_only=esc_only,mod_fits=mod_fits,userefits=userefits,refitname=models[5]
```

```
## Warning: Relative effective sample sizes ('r_eff' argument) not specified.
## For models fit with MCMC, the reported PSIS effective sample sizes and
## MCSE estimates will be over-optimistic.

## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
# plot_ks(LFO3$ks, N=N,L=L,thres=thres)
# plot_ks(LFO3$k_LOOIC, N=N,L=L,thres=thres)

LFO6<-approx_LFO(N=N,L=L,m=6,esc_only=esc_only,mod_fits=mod_fits,userefits=userefits,refitname=models[6]
```

```
## Warning: Relative effective sample sizes ('r_eff' argument) not specified.
## For models fit with MCMC, the reported PSIS effective sample sizes and
## MCSE estimates will be over-optimistic.

## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
# plot_ks(LFO3$ks, N=N,L=L,thres=thres)
# plot_ks(LFO3$k_LOOIC, N=N,L=L,thres=thres)



ELPD<-c(sum(LFO1$LFO,na.rm=T),
        sum(LFO2$LFO,na.rm=T),
        sum(LFO3$LFO,na.rm=T),
        sum(LFO4$LFO,na.rm=T),
        sum(LFO5$LFO,na.rm=T),
        sum(LFO6$LFO,na.rm=T)
        )
```

```r
LFOIC<--2*(ELPD)
delta_LFOIC<-LFOIC-min(LFOIC)
LFOIC_weight<-exp(ELPD)/sum(exp(ELPD))
LFOIC_results<-data.frame(ELPD,LFOIC,delta_LFOIC, LFOIC_weight)
rownames(LFOIC_results)<-models
model_selection<-data.frame(model_selection,LFOIC_results)
```

Model Averaging and 2020 forecast

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

f_dat<-data.frame(
  sort(do.call("c", mod_fits[[1]][,paste0("Sp","[",n_yrs+n_fore,"]")])),
  sort(do.call("c", mod_fits[[2]][,paste0("Sp","[",n_yrs+n_fore,"]")])),
  sort(do.call("c", mod_fits[[3]][,paste0("Sp","[",n_yrs+n_fore,"]")])),
  sort(do.call("c", mod_fits[[4]][,paste0("Sp","[",n_yrs+n_fore,"]")])),
  sort(do.call("c", mod_fits[[5]][,paste0("Sp","[",n_yrs+n_fore,"]")])),
  sort(do.call("c", mod_fits[[6]][,paste0("Sp","[",n_yrs+n_fore,"]")]))
)


model_selection[,"2020_forecast"] <- apply(f_dat,2,median)

weighted_forecast_dist <-(
  as.matrix(f_dat) %*% (as.vector(model_selection[,"RMSE_weight"]))
)
weighted_forecast_quantiles<-quantile(weighted_forecast_dist,c(0.025,0.25,0.50,0.75,0.975))
weighted_forecast<-weighted_forecast_quantiles[3]
print(model_selection)
```

```
##                             RMSE        MPE      MAPE      MSA RMSE_weight MAPE_weight MSA_weight
## IPM_BH_cov_MA1_AR1      1568.000 0.10751942 0.2240939 45.11414   0.1465274   0.1488298  0.1632765
## IPM_BH_cov_AR           1642.006 0.11174421 0.2396936 45.71892   0.1399233   0.1391437  0.1611166
## IPM_BH_cov_AR_resid     1525.042 0.09448217 0.2155400 44.83824   0.1506548   0.1547364  0.1642812
## IPM_BH_cov_MA1_AR1_age  1154.355 0.08237172 0.1773716 43.20713   0.1990331   0.1880338  0.1704830
## IPM_BH_cov_AR_age       1319.431 0.12904289 0.1969258 43.55082   0.1741319   0.1693626  0.1691376
## IPM_BH_cov_AR_resid_age 1210.961 0.05841099 0.1668480 42.89957   0.1897294   0.1998937  0.1717052
##                             ELPD      LFOIC delta_LFOIC LFOIC_weight 2020_forecast
## IPM_BH_cov_MA1_AR1      6.974438 -13.948876    1.703443  0.161254698      4692.778
## IPM_BH_cov_AR           7.826160 -15.652320    0.000000  0.377929157      5094.126
## IPM_BH_cov_AR_resid     7.200095 -14.400190    1.252130  0.202075605      4610.069
## IPM_BH_cov_MA1_AR1_age  5.351941 -10.703881    4.948438  0.031832494      3446.285
## IPM_BH_cov_AR_age       7.309776 -14.619551    1.032768  0.225500545      3916.261
## IPM_BH_cov_AR_resid_age 2.233269  -4.466538   11.185782  0.001407501      2312.011
```

```r
print("The model-averaged forecast is:")
```

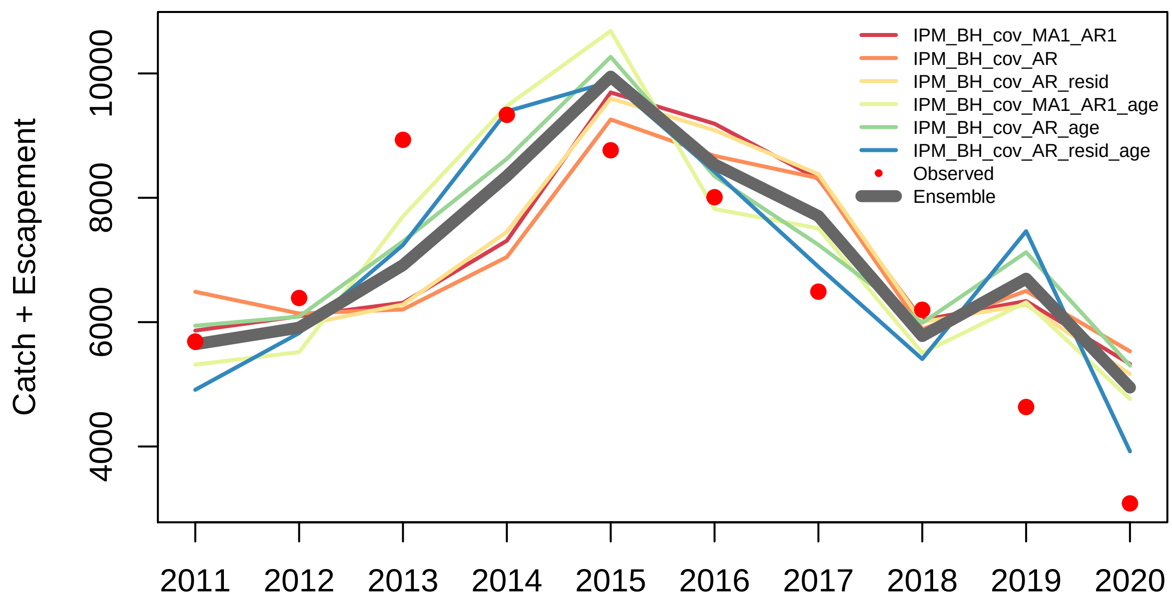
```
## [1] "The model-averaged forecast is:"
```

```r
print(weighted_forecast_quantiles)
```

```
##     2.5%      25%      50%      75%    97.5%
## 1927.004 3001.551 3901.464 4954.940 7704.933
```

```r
ensemble_median<-pred_trs%*%(as.vector(model_selection[,"RMSE_weight"]))

cols<-brewer.pal(length(models),"Spectral")
matplot(as.matrix(data.frame(pred_trs,ensemble_median)),type="l",lty=1,col=c(cols,"grey40"),lwd=c(rep(2
axis(1,1:n_forecasts,(yr_last-n_forecasts+1):(yr_last))
points(x=1:n_forecasts,y=obs_trs,cex=1.5,pch=20,col="red")
legend("topright",legend=c(models,"Observed","Ensemble"),lty=c(rep(1,length(models)),NA),col=c(cols,"re
```



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.

```r
LOOIC <- vector("list", n_mods)
## extract log densities from JAGS objects
for(i in 1:n_mods) {
  #i <- 1
  ## convert mcmc.list to matrix
  tmp_lp <- as.matrix(readRDS(file.path(savedir,paste0(models[i],"_y",11,".rds"))))
  ## 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 = models,
                   as.data.frame(tbl_LOOIC),pseudo_bma_weight = as.matrix(model_weights))
tbl_LOOIC[order(tbl_LOOIC[,"looic"]), ]
```

```
##                      model elpd_diff se_diff elpd_loo se_elpd_loo  p_loo se_p_loo  looic se_looic
## 2           IPM_BH_cov_AR      0.00    0.00  -395.32       48.96 138.77    10.38 790.64    97.91
## 1       IPM_BH_cov_MA1_AR1    -8.91    6.69  -404.23       49.97 147.25    13.57 808.47    99.94
## 3      IPM_BH_cov_AR_resid   -14.32   11.02  -409.64       50.51 151.14    16.84 819.28   101.01
## 4  IPM_BH_cov_MA1_AR1_age   -18.19   11.90  -413.51       54.25 136.34    12.92 827.02   108.49
## 5        IPM_BH_cov_AR_age   -22.70   12.08  -418.03       54.55 140.64    13.02 836.05   109.10
## 6 IPM_BH_cov_AR_resid_age   -86.84   11.29  -482.17       49.14 179.09    10.16 964.33    98.27
##   pseudo_bma_weight
## 2      8.827773e-01
## 1      4.049169e-02
## 3      2.752584e-02
## 4      4.596981e-02
## 5      3.235345e-03
## 6      3.228228e-29
```

```
## 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.