R2. Model fitting and evaluation

2021 - 2022 Skagit River steelhead forecast.

Contents

This is version 0.21.12.13.

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

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

## set directory locations
datadir <- here("data")
jagsdir <- here("jags")
analdir <- here("analysis")
savedir <- here("analysis/cache")
```

We also need a couple of helper functions.

```r
## 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,
```

```r
                    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)
  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
```

```r
# 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
  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])
    }
  }
  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)
}
#calculate stacking weights
find_stack_weights<-function(tau,metric,n,initial_weights,preds,obs){
  tweights<-initial_weights
  preds<-as.matrix(preds)
  obs<-obs
  tau=tau
  skill_list<-c(NULL)
  metric=metric
  for(i in 1:n){
    pred_trs_ensemble<- preds %*% as.vector(tweights)
    Error <- pred_trs_ensemble - obs
    SE <- Error^2
    PE <- Error/obs_trs
```

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

    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)})

    if(i==1){
      skill=get(metric)
      weights=tweights
    }
    if(get(metric)<skill){
      skill=get(metric)
      weights=tweights
    }
    skill_list<-c(skill_list,min(get(metric),skill))
    keep<-rbinom(1,prob=skill/get(metric),1)
    if(keep==1){tweights=tweights }else(tweights=weights)
    tweights = rdirichlet(n=1,alpha = tweights*tau+0.001)
  }
  results<-list(weights,skill,skill_list)
  return(results)
}

#==============================
#function to fit or load modelfits
#==================================
fit_load_mods<-function(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 up
  ## 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
```

```r
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_forecas
}
## 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]
```

```r
## ----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","Run"
                )

}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","Run"
                          )


          }#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)
  return(mod_fits)
}
```

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 <- 2021

## 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 <- 11

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

## last year of 1 step ahead forecast
yr_end <- 2021

## 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:

```r
## 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:

```r
## 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:

```r
## 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 evaluated a total of six different models, which we outline below, based on a beverton holt process model with covariates. Specifically, we evaluated three different methods to model recruitment residuals including a moving average process lagged 1 year (MA1), an auto-regressive moving average process lagged 1 year (AR1MA1), and an auto-regressive process lagged 1 year (AR1). In addition, we evaluated two methods for modeling maturation including a mean reverting process, and a random walk process.

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/cruff/Documents/projects/
## RWorkflow/Skagit-River-Steelhead-Forecast/jags/IPM_BH_cov_AR.txt'
```

Beverton-Holt with covars and AR1MA1 recruitment residuals

```r
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/cruff/Documents/projects/
## RWorkflow/Skagit-River-Steelhead-Forecast/jags/IPM_BH_cov_MA1_AR1.txt'
```

Beverton-Holt with covars and AR1 recruitment residuals

```r
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/cruff/Documents/projects/
## RWorkflow/Skagit-River-Steelhead-Forecast/jags/IPM_BH_cov_AR_resid.txt'
```

Fitting the models and generating the one year ahead forecasts

For the most recent 11 years (2011 - 2021), fit the model to data through year t-1 and generate a forecast for year t.

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
```

```r
## fit or load 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"
            )
n_mods<-length(models)
mod_fits <- fit_load_mods(models=models)
```

Model selection

We evaluated model performance following three separate approaches outlined below.

Model selection via root mean squared error (RMSE) and mean absolute percent error (MAPE) model performance statistics.

```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


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 = "")
}
```

```r
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(median(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

Detailed methods described here: **link**. This approach is better suited to evaluating the predictive performance of Bayesian time series models than the more traditional model performance metrics such as RMSE and MAPE.

```r
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.
```

```r
# 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.
```

```r
# 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.
```

```r
# 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.
```

```r
# 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.
```

```r
# 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.
```

```r
# 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)
        )
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)
```

Calculating Stacking Weights (finding model averaging weights based on one-step-ahead performance of weighted average models). This is an experimental calculation of stacking weights based on linear comninations of model forecasts to optimize one step ahead RMSE, MSA, or MAPE… it is not ready for prime time yet. An alternative is to figure out how to use methods similar to "stacking weights" here for LFOIC: **link**

```
stack_weights<-find_stack_weights(tau=1,
                   n=10000,
                   metric="MSA",
                   initial_weights=rep(1/length(models),length(models)),
                   preds=pred_trs,
                   obs=obs_trs
                   )
stacking_weights<-as.vector(round(unlist(stack_weights[[1]]),4))
model_selection$stacking_weights<-stacking_weights
```

Model Selection 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)
## 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 =

model_weights <- loo_model_weights(LOOIC, method = "stacking",optim_method = "BFGS", optim_control = lis

## 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),LOOIC_weight = as.matrix(model_weights))
tbl_LOOIC$delta_LOOIC<-tbl_LOOIC$looic-min(tbl_LOOIC$looic)
model_selection$LOOIC<-tbl_LOOIC$looic
model_selection$delta_LOOIC<-tbl_LOOIC$delta_LOOIC
model_selection$LOOIC_weight<-round(tbl_LOOIC$LOOIC_weight,4)
```

Model Averaging and 2021 - 2022 forecast

Here is weighted average forecast for 2021 - 2022 based on the model LFOIC weights. We also compare predictions for return years 2011 - 2021 generated from each of the six models evaluated including an ensemble (weighted) to observed run size.

```
## extract median 2021 forecast from each model
f_dat<-data.frame(
  sort(unlist(mod_fits[[1]][,paste0("Sp","[",n_yrs+n_fore,"]")])),
  sort(unlist(mod_fits[[2]][,paste0("Sp","[",n_yrs+n_fore,"]")])),
  sort(unlist(mod_fits[[3]][,paste0("Sp","[",n_yrs+n_fore,"]")])),
  sort(unlist(mod_fits[[4]][,paste0("Sp","[",n_yrs+n_fore,"]")])),
  sort(unlist(mod_fits[[5]][,paste0("Sp","[",n_yrs+n_fore,"]")])),
  sort(unlist(mod_fits[[6]][,paste0("Sp","[",n_yrs+n_fore,"]")]))
)
colnames(f_dat)<-models

model_selection[,"2021_forecast"] <- apply(f_dat,2,median)

weighted_forecast_dist <-(
  as.matrix(f_dat) %*% (as.vector(model_selection[,"LFOIC_weight"]))
)
ensemble_forecast_posterior<-data.frame(weighted_forecast_dist)
colnames(ensemble_forecast_posterior)<-c("ensemble_forecast_posterior")
write.csv(ensemble_forecast_posterior,file.path(savedir,"ensemble_forecast_posterior.csv"),row.names = F
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       1522.997 0.1280792 0.2319023 25.812104   0.1539514   0.1500705 0.09584554
## IPM_BH_cov_AR            1635.574 0.1414437 0.2534141 28.320493   0.1433549   0.1373313 0.08735636
## IPM_BH_cov_AR_resid      1514.595 0.1184708 0.2247478 26.135135   0.1548055   0.1548477 0.09466089
## IPM_BH_cov_MA1_AR1_age   1298.900 0.1095133 0.1862931 11.611937   0.1805124   0.1868115 0.21305448
## IPM_BH_cov_AR_age        1296.497 0.1263707 0.1906831  8.541025   0.1808469   0.1825106 0.28965787
## IPM_BH_cov_AR_resid_age  1257.003 0.1000047 0.1846945 11.274816   0.1865290   0.1884284 0.21942488
```

```
##                               ELPD     LFOIC delta_LFOIC LFOIC_weight stacking_weights  LOOIC
## IPM_BH_cov_MA1_AR1      7.890284 -15.78057    1.596708    0.1340650           0.0000 796.86
## IPM_BH_cov_AR           8.688638 -17.37728    0.000000    0.2978764           0.0000 791.09
## IPM_BH_cov_AR_resid     7.973363 -15.94673    1.430551    0.1456786           0.0000 802.78
## IPM_BH_cov_MA1_AR1_age  7.928046 -15.85609    1.521184    0.1392244           0.0000 821.22
## IPM_BH_cov_AR_age       7.816692 -15.63338    1.743892    0.1245531           0.4622 831.17
## IPM_BH_cov_AR_resid_age 8.058360 -16.11672    1.260555    0.1586025           0.5378 819.98
##                         delta_LOOIC LOOIC_weight 2021_forecast
## IPM_BH_cov_MA1_AR1             5.77       0.0000      4089.238
## IPM_BH_cov_AR                  0.00       0.2623      4177.088
## IPM_BH_cov_AR_resid           11.69       0.3656      4045.142
## IPM_BH_cov_MA1_AR1_age        30.13       0.3720      3432.963
## IPM_BH_cov_AR_age             40.08       0.0000      3517.067
## IPM_BH_cov_AR_resid_age       28.89       0.0001      3374.835
```

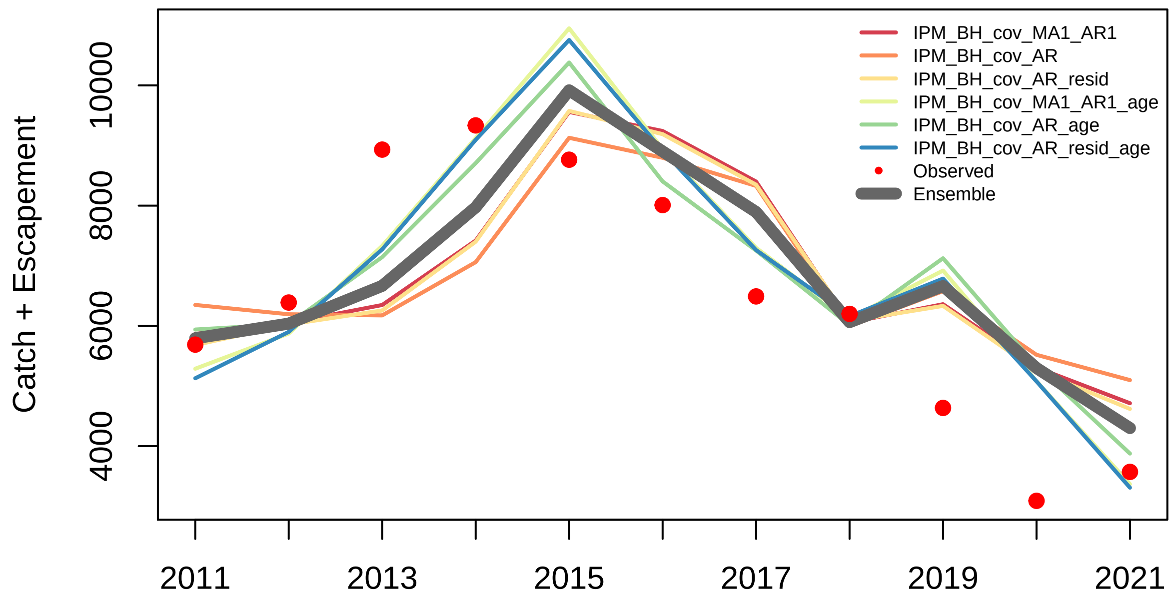
```r
print("The model-averaged forecast is:")
```

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

```r
print(weighted_forecast_quantiles)
```

```
##     2.5%      25%      50%      75%    97.5%
## 1995.844 3081.315 3833.041 4725.639 7132.176
```

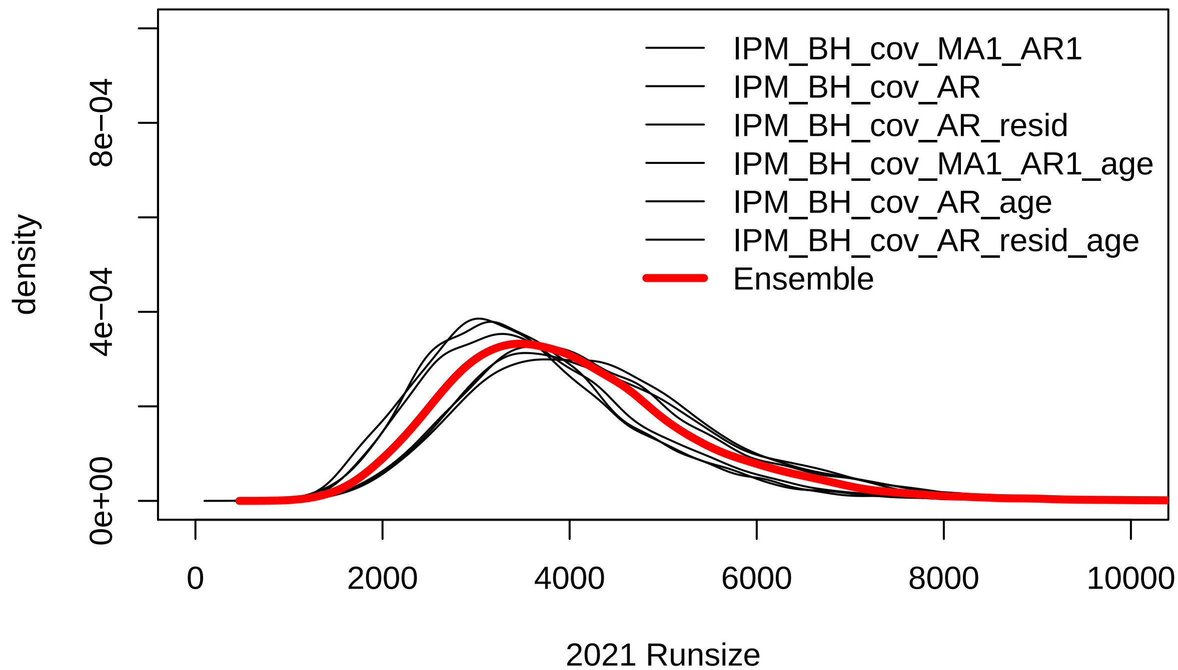
```r
ensemble_median<-as.matrix(pred_trs)%*%(as.vector(model_selection[,"LFOIC_weight"]))


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
```




```
#density plot for final forecasts and ensemble
res<-apply(f_dat,2,function(x) density(x))
plot(x=1,y=1,ylim=c(0,0.001),xlim=c(0,10000),ylab="density",xlab="2021 Runsize")
lapply(res,function(x) lines(x$y~x$x))
```

```
## $IPM_BH_cov_MA1_AR1
## NULL
##
## $IPM_BH_cov_AR
## NULL
##
## $IPM_BH_cov_AR_resid
## NULL
##
## $IPM_BH_cov_MA1_AR1_age
## NULL
##
## $IPM_BH_cov_AR_age
## NULL
##
## $IPM_BH_cov_AR_resid_age
## NULL
```

```
lines(density(weighted_forecast_dist)$y~density(weighted_forecast_dist)$x,lwd=4,col="red")
legend("topright",legend=c(models,"Ensemble"),lwd=c(rep(1,length(models)),4),col=c(rep("black",length(mo
```



Now lets graph model estimated spawners, recruits, and recruits per spawner:

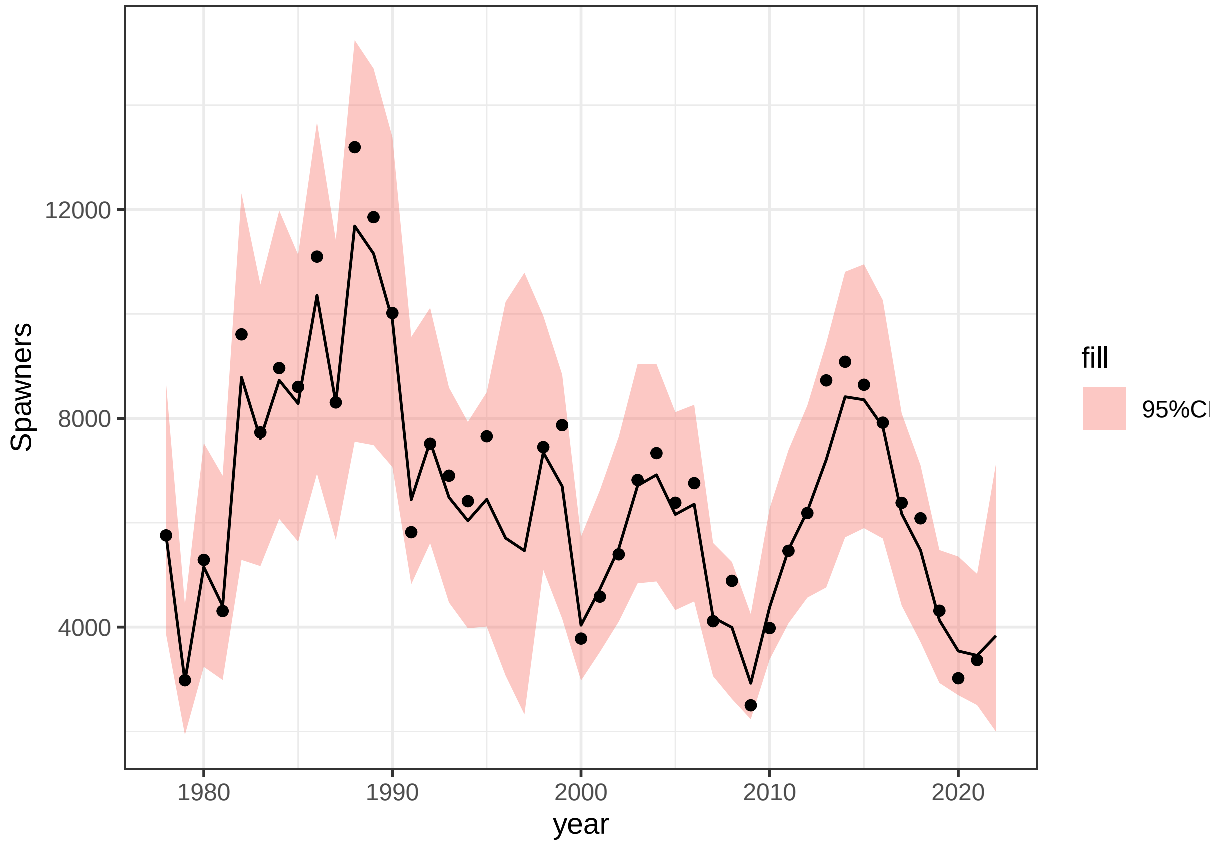
```
RSdat<-function(var,yrs){
  sims<-length(sort(unlist(mod_fits[[1]][,paste0(var,"[",i,"]")])))
  R_mat<-matrix(NA,ncol=length(yrs),nrow=sims)
  for(i in 1:length(yrs)){
    Rdat<-data.frame(
      sort(unlist(mod_fits[[1]][,paste0(var,"[",min(yrs)+i-1,"]")])),
      sort(unlist(mod_fits[[2]][,paste0(var,"[",min(yrs)+i-1,"]")])),
      sort(unlist(mod_fits[[3]][,paste0(var,"[",min(yrs)+i-1,"]")])),
      sort(unlist(mod_fits[[4]][,paste0(var,"[",min(yrs)+i-1,"]")])),
      sort(unlist(mod_fits[[5]][,paste0(var,"[",min(yrs)+i-1,"]")])),
      sort(unlist(mod_fits[[6]][,paste0(var,"[",min(yrs)+i-1,"]")]))
    )
    dim(Rdat)
    weighted_R <-(
      as.matrix(Rdat) %*% (as.vector(model_selection[,"LFOIC_weight"]))
    )
    R_mat[,i]<-weighted_R
    colnames(R_mat)<-yrs
  }
  return(R_mat)
}
Sdat<-RSdat(var="Sp",yrs=c(1:(n_yrs+n_fore)))
Rdat<-exp(RSdat(var="tot_ln_Rec",yrs=c(1:(n_yrs+n_fore-3))))
RSdat<-exp(RSdat(var="ln_RS",yrs=c(1:(n_yrs+n_fore-3))))
```

```r
library(tidyverse)
```

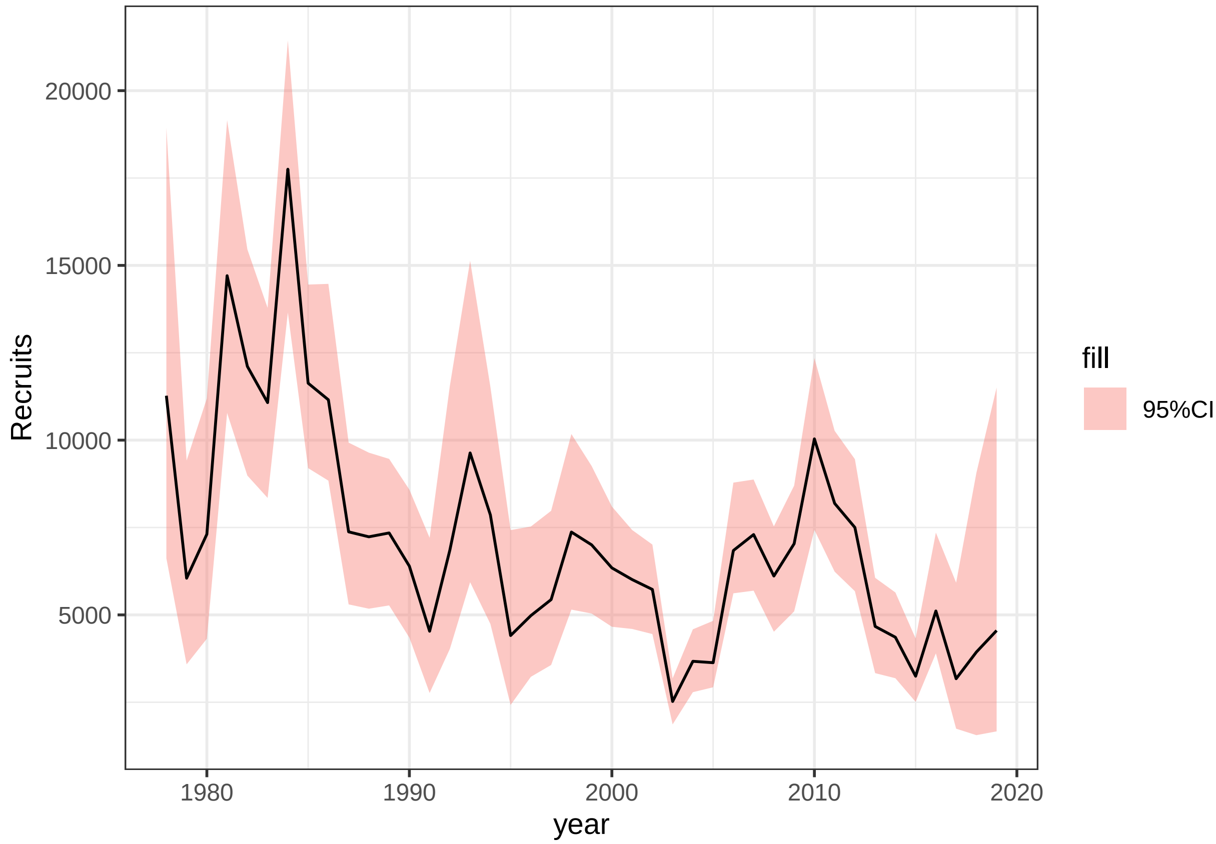
```
## Warning: package 'tidyverse' was built under R version 4.0.5

## Warning: package 'tidyr' was built under R version 4.0.5

## Warning: package 'dplyr' was built under R version 4.0.5

## Warning: package 'forcats' was built under R version 4.0.5
```

```r
Sdat<-as.data.frame(Sdat)%>%
  pivot_longer(cols=everything())%>%
  rename(year=name)%>%
  group_by(year) %>%
  summarise(SpawnerAbundance = quantile(value, c(0.025, 0.5, 0.975)), q = c(0.025, 0.5, 0.975))%>%
  mutate(year=as.numeric(year)+yr_frst-1)%>%
  pivot_wider(names_from = q, values_from = SpawnerAbundance)

Rdat<-as.data.frame(Rdat)%>%
  pivot_longer(cols=everything())%>%
  rename(year=name)%>%
  group_by(year) %>%
  summarise(RecruitAbundance = quantile(value, c(0.025, 0.5, 0.975)), q = c(0.025, 0.5, 0.975))%>%
  mutate(year=as.numeric(year)+yr_frst-1)%>%
  pivot_wider(names_from = q, values_from = RecruitAbundance)

RSdat<-as.data.frame(RSdat)%>%
  pivot_longer(cols=everything())%>%
  rename(year=name)%>%
  group_by(year) %>%
  summarise(RS = quantile(value, c(0.025, 0.5, 0.975)), q = c(0.025, 0.5, 0.975))%>%
  mutate(year=as.numeric(year)+yr_frst-1)%>%
  pivot_wider(names_from = q, values_from = RS)

ggplot(Sdat,aes(x=year,y=`0.5`))+
  geom_ribbon(aes(ymin = `0.025`, ymax = `0.975`,fill="95%CI"),alpha=0.4)+
  geom_line()+
  geom_point(dat_esc,mapping=aes(y=escapement,x=year))+
  theme_bw()+
  ylab("Spawners")
```

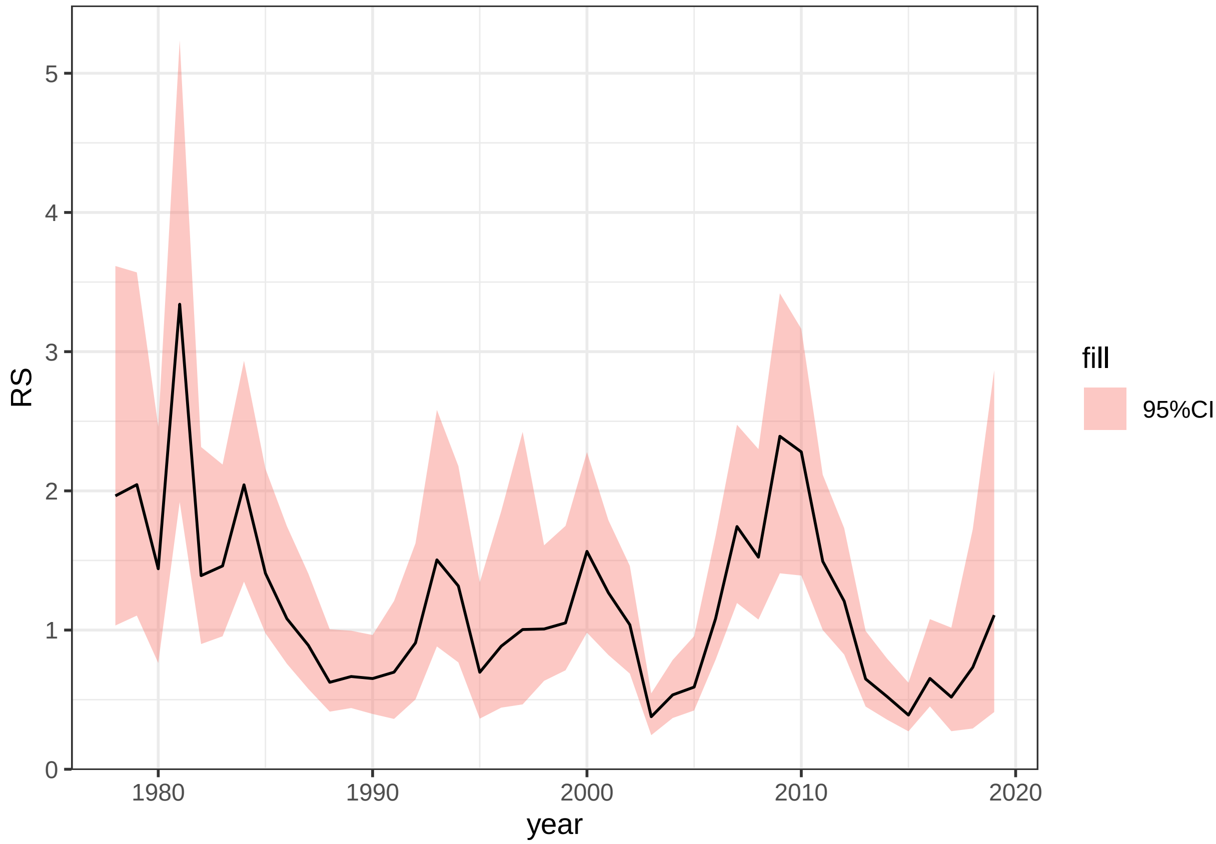
```
## Warning: Removed 2 rows containing missing values (geom_point).
```




```r
ggplot(Rdat,aes(x=year,y=`0.5`))+
  geom_ribbon(aes(ymin = `0.025`, ymax = `0.975`,fill="95%CI"),alpha=0.4)+
  geom_line()+
  theme_bw()+
  ylab("Recruits")
```




```
ggplot(RSdat,aes(x=year,y=`0.5`))+
  geom_ribbon(aes(ymin = `0.025`, ymax = `0.975`,fill="95%CI"),alpha=0.4)+
  geom_line()+
  theme_bw()+
  ylab("RS")
```



Compare sib adjusted forecasts with unadjusted

Currently implemented for only two of the six model including the AR1MA1 recruitment residuals with random walk maturation and the AR1 recruitment residuals with random walk maturation.

```
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"
            )
n_mods<-length(models)

mod_fits <- fit_load_mods(models=models
)
#####new version 1.7
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)),]
```

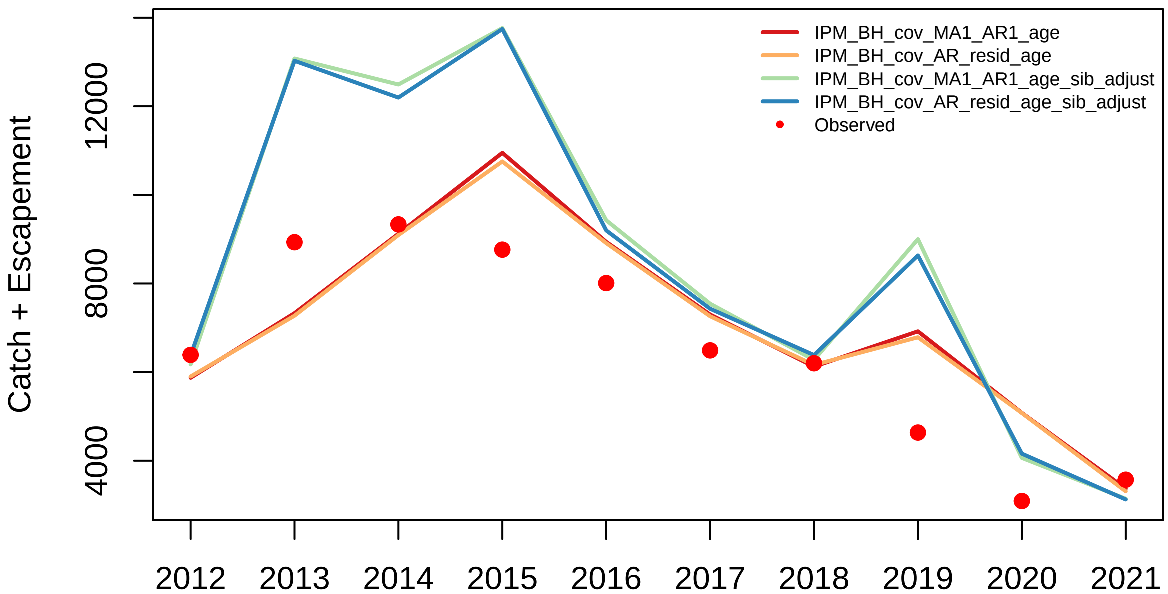
```r
## observed terminal run size
obs_trs <- dat_esc_forecast$escapement + dat_harv_forecast$catch
pred_trs <- NULL
pred_trs_adj <- NULL
for(n in 1:n_mods){
  #n <- 1
  pred_esc <- NULL
  pred_esc_adj <- NULL

  c <- 0
  f <- 2
  adj<- NULL
  for(i in 1:(n_forecasts)){
    #i <- 1

    #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)


  mod_res<-NULL

  #model refit and one step ahead forecast
  mod_res_pred<-as.matrix(readRDS(file.path(savedir,paste0(models[n],"_y",i,".rds"))))

  #model refit for year i+1 to extract "observed" state
  mod_res_obs<-as.matrix(readRDS(file.path(savedir,paste0(models[n],"_y",i + 1,".rds"))))
  #
  p_dat_pred <- mod_res_pred[,grep("Sp", colnames(mod_res_pred))]
  p_dat_pred <- round(median(p_dat_pred[,n_yrs + 1]))


  p_dat_obs <- mod_res_obs[,grep("Sp", colnames(mod_res_obs))]
  p_dat_obs <- round(median(p_dat_obs[,n_yrs + 2]))

  pred_esc[i] <- p_dat_pred

  #take same model and implement sibling adjustment
  ## forecast for year t
  p_dat_pred_adj <- cbind(((mod_res_pred[,paste0("Run","[",n_yrs - age_min + 1,",",1,"]")])),
               ((mod_res_pred[,paste0("Run","[",n_yrs - age_min + 1,",",2,"]")])),
               ((mod_res_pred[,paste0("Run","[",n_yrs - age_min + 1,",",3,"]")])),
               ((mod_res_pred[,paste0("Run","[",n_yrs - age_min + 1,",",4,"]")])),
               ((mod_res_pred[,paste0("Run","[",n_yrs - age_min + 1,",",5,"]")])),
               ((mod_res_pred[,paste0("Run","[",n_yrs - age_min + 1,",",6,"]")])))



  p_dat_pred_adj_sum <- apply(p_dat_pred_adj,1,FUN = "sum")
  median(p_dat_pred_adj_sum)
```

```r
  ## observation for year t+1
  p_dat_obs_adj <- cbind(((mod_res_obs[,paste0("Run","[",n_yrs - age_min + 1,",",1,"]")])),
                   ((mod_res_obs[,paste0("Run","[",n_yrs - age_min + 1,",",2,"]")])),
                   ((mod_res_obs[,paste0("Run","[",n_yrs - age_min + 1,",",3,"]")])),
                   ((mod_res_obs[,paste0("Run","[",n_yrs - age_min + 1,",",4,"]")])),
                   ((mod_res_obs[,paste0("Run","[",n_yrs - age_min + 1,",",5,"]")])),
                   ((mod_res_obs[,paste0("Run","[",n_yrs - age_min + 1,",",6,"]")])))


  ## forecast for year t+1 to be adjusted
  p_dat_pred_t <- cbind(((mod_res_obs[,paste0("Run","[",n_yrs - age_min + 2,",",1,"]")])),
                   ((mod_res_obs[,paste0("Run","[",n_yrs - age_min + 2,",",2,"]")])),
                   ((mod_res_obs[,paste0("Run","[",n_yrs - age_min + 2,",",3,"]")])),
                   ((mod_res_obs[,paste0("Run","[",n_yrs - age_min + 2,",",4,"]")])),
                   ((mod_res_obs[,paste0("Run","[",n_yrs - age_min + 2,",",5,"]")])),
                   ((mod_res_obs[,paste0("Run","[",n_yrs - age_min + 2,",",6,"]")])))


  #median(p_dat_obs_adj[,a-1])/median(p_dat_pred_adj[,a-1])


    p_dat_pred_adj_a <- NULL
    #adj_a <- NULL
    for (a in 2:A){
      #a <- 3
      #adj_a[a] <- median(as.vector(p_dat_obs_adj[,a-1])/as.vector(p_dat_pred_adj[,a-1]))
      p_dat_pred_adj_a_temp <- as.vector(p_dat_pred_t[,a])*(as.vector(p_dat_obs_adj[,a-1])/as.vector(p_d
      p_dat_pred_adj_a <- cbind(p_dat_pred_adj_a,p_dat_pred_adj_a_temp)


    }

    #adj <- rbind(adj,t(adj_a))
    p_dat_pred_adj_a <- apply(p_dat_pred_adj_a,1,FUN = sum)

    pred_esc_adj[f] <- round(median(p_dat_pred_adj_a))
    #pred_esc_adj[f] <- median(apply(p_dat_pred_t,1,FUN = "sum"))

    f <- f+1
    c <- c+1
  }#next forecast

  pred_trs_mod <- pred_esc + 1#+ dat_harv_forecast$catch #you don't need to add catch in because it is i
  pred_trs_mod_adj <- pred_esc_adj #not adjusted because adjusted prediction comes from "Run" rather th

  pred_trs <- cbind(pred_trs,pred_trs_mod)
  pred_trs_adj <- cbind(pred_trs_adj,pred_trs_mod_adj)
  #names(pred_trs) <- paste(models[n],"_","pred_trs",sep = "")
}
colnames(pred_trs) <- models
colnames(pred_trs_adj) <- paste(models,"_","sib_adjust",sep = "")
pred_trs_adj <- pred_trs_adj[-11,]
```

```r
pred_trs<-data.frame(pred_trs,pred_trs_adj)
pred_trs<-pred_trs[-1,]

## compute model performance statistics
Error <- pred_trs - obs_trs[-1]
SE <- Error^2
PE <- Error/obs_trs[-1]
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)
print(model_selection)
```

```
##                                        RMSE       MPE      MAPE      MSA RMSE_weight MAPE_weight
## IPM_BH_cov_MA1_AR1_age             1356.383 0.1275145 0.1978725 39.88643   0.3282396   0.3015716
## IPM_BH_cov_AR_resid_age            1306.408 0.1198504 0.1933187 46.60243   0.3407961   0.3086754
## IPM_BH_cov_MA1_AR1_age_sib_adjust  2748.429 0.2828013 0.3135843 63.19327   0.1619902   0.1902924
## IPM_BH_cov_AR_resid_age_sib_adjust 2634.835 0.2738471 0.2991705 68.37404   0.1689740   0.1994606
##                                    MSA_weight
## IPM_BH_cov_MA1_AR1_age              0.3256878
## IPM_BH_cov_AR_resid_age             0.2787521
## IPM_BH_cov_MA1_AR1_age_sib_adjust   0.2055681
## IPM_BH_cov_AR_resid_age_sib_adjust  0.1899920
```

```r
#sib adjust plot
cols<-brewer.pal(length(models)*2,"Spectral")
matplot(as.matrix(data.frame(pred_trs)),type="l",lty=1,col=c(cols,"grey40"),lwd=c(rep(2,length(models)*
axis(1,1:(n_forecasts-1),(yr_last-n_forecasts+2):(yr_last))
points(x=1:(n_forecasts-1),y=obs_trs[-1],cex=1.5,pch=20,col="red")
legend("topright",legend=c(rownames(model_selection),"Observed"),lty=c(rep(1,length(models)*2),NA),col=
```



It looks like applying the sibling adjustment improved the forecast for only 2 of the 10 years and resulted in poorer performance overall.