R1. Model fitting

2021 Skagit S/F Chinook Spawner Recruit Model fitting

Contents

This is version 0.21.10.06.

```
## [1] TRUE
```

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")
}
```

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

management_unit <- "summer_fall"

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

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

```r
# 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,"_",run,".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,"_",run,".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
  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)
```

```r
  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)
}

#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)
}

#==============================
#function to fit or load modelfits
#==================================
fit_load_mods<-function(models){

  models <- 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 u
  ## 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
    model <- models[n]
```

```
  for(i in 1:(n_forecasts+1)){
    if(file.exists(file.path(savedir,paste(model,"_","y",i,"_",run,".rds",sep = "")))) {
      mod_fits[[t]] <- readRDS(file.path(savedir,paste(model,"_","y",i,"_",run,".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","_",run,".
      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_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)
```

```r
## 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.

  init_vals <- function() {
  list(alpha = 5,
       #beta_inv = exp(mean(ln_dat_esc, na.rm = TRUE)),
       beta=1/exp(mean(ln_dat_esc, na.rm=TRUE)),
       pi_tau = 10,
       pi_eta = rep(1,A),
       pi_vec = matrix(c(0.020,0.219,0.581,0.179),
                       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))
  }

  par_jags <- c("alpha","E_Rkr_a","mu_Rkr_a",
                "beta",
                "Sp","Rec","tot_ln_Rec","ln_RS","tot_Run",
                "pi_eta","pi_tau","pi_vec",
                "sigma_r","sigma_s","res_ln_Rec",
                "lp_age","lp_esc","Run")


#   init_vals <- function() {
```

```
#   list(alpha = 5,
#        beta_inv = exp(mean(ln_dat_esc, na.rm = TRUE)),
#        pi_tau = 10,
#        pi_eta = rep(1,A),
#        pi_vec = matrix(c(0.020,0.219,0.581,0.179),
#                        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)
#   }
#
#
# if(model == "IPM_BH_AR"){
#   ## params/states to return
#   par_jags<- c("alpha","E_BH_a","mu_BH_a",
#                "beta",
#                "Sp","Rec","tot_ln_Rec","ln_RS",
#                "pi_eta","pi_tau","pi_vec",
#                "sigma_r","sigma_s","w","res_ln_Rec",
#                "lp_age","lp_esc","phi","Run"
#                )
#
# }else{
#   ## params/states to return
#   par_jags <- c("alpha","E_Rkr_a","mu_Rkr_a",
#                "beta",
#                "Sp","Rec","tot_ln_Rec","ln_RS","tot_Run",
#                "pi_eta","pi_tau","pi_vec",
#                "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,
```

```
                              ctrl=mcmc_ctrl
                              )
          saveRDS(mod_fits[[t]], file.path(savedir,paste(model,"_","y",i,"_",run,".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","_",run,".c
          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 <- 1992
yr_last <- 2018




## min & max ad ult age classes
age_min <- 2
age_max <- 5

## 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 <- 0

## number of recent year forecasts
n_forecasts <- 0

## 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

## run sfck = summer/fall; spck = spring
run <- "sfck"
```

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 <- paste("skagit","_",run,"_","esc.csv",sep = "")
paste("skagit","_",run,"_","esc.csv",sep = "")
```

```
## [1] "skagit_sfck_esc.csv"
```

```
## 2. file with age comp data
## [n_yrs x (1+A)]; 1st col is calendar yr
fn_age <- paste("skagit","_",run,"_","age.csv",sep = "")

## 3. file with harvest data
## [n_yrs x 2] matrix of obs catch; 1st col is calendar yr
fn_harv <- paste("skagit","_",run,"_","catch.csv",sep = "")
```

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))))
```

```r
# }
# ## 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))
```

Specifying models in JAGS

Now we can specify the model in JAGS. For this effort, we are only evaluating a Ricker model with gaussian process errors.

Ricker with with gaussian process errors

```r
processFile(file.path(jagsdir, "IPM_RK.txt"))
```

```
## model {
##
##   ##--------
##   ## PRIORS
##   ##--------
##   ## alpha = exp(a) = intrinsic productivity
##   alpha ~ dunif(0.1,10);
##   mu_Rkr_a <- log(alpha);
##   E_Rkr_a <- mu_Rkr_a + sigma_r/2;
##
##
##   ## strength of dens depend
##   #beta_inv ~ dnorm(0, 1e-9) T(0,);
##   #beta <- 1/beta_inv;
##   beta ~ dunif(0,0.1);
##
##
##   ## process variance for recruits model
##   sd_r ~ dunif(0.001,20);
##   tau_r <- pow(sd_r,-2);
##   sigma_r <- pow(sd_r,2);
```

```
##
##   ## obs variance for spawners
##   sd_s ~ dunif(0.001,20);
##   tau_s <- pow(sd_s,-2);
##   sigma_s <- pow(sd_s,2);
##
##   ## 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);
##
##   ## maturity schedule
##   ## unif vec for Dirch prior
##   for(i in 1:A) { theta[i] <- 1 }
##   ## hyper-mean for maturity
##   pi_eta ~ ddirch(theta);
##   ## hyper-prec for maturity
##   pi_tau ~ dunif(0.001,1e3);
##   for(t in 1:(n_yrs-age_min)) { pi_vec[t,1:A] ~ ddirch(pi_eta*pi_tau) }
##
##   ##------------
##   ## LIKELIHOOD
##   ##------------
##   ## 1st brood yr requires different innovation
##   ## predicted recruits in BY t
##   E_ln_Rec[1] <- ln_Sp[1] - beta*Sp[1] + mu_Rkr_a;
##   tot_ln_Rec[1] ~ dnorm(E_ln_Rec[1],tau_r);
##   res_ln_Rec[1] <- tot_ln_Rec[1] - E_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] <- max(1,tot_Rec[1] * pi_vec[1,a]);
##   }
##
##   ## brood years 2:(n_yrs-age_min)
##   for(t in 2:(n_yrs-age_min)) {
##     ## predicted recruits in BY t
##     E_ln_Rec[t] <- ln_Sp[t] - beta*Sp[t] + mu_Rkr_a;
##     tot_ln_Rec[t] ~ dnorm(E_ln_Rec[t],tau_r);
##     res_ln_Rec[t] <- tot_ln_Rec[t] - E_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] <- max(1,tot_Rec[t] * pi_vec[t,a]);
##     }
##   } ## end t loop over year
##
##   ## 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]);
##   }
##
##   ## 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)) {
##     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]);
##   }
```

```
##
##   ## get predicted calendar year spawners
##   ## first cal yr is first brood yr
##   for(t in 1:(n_yrs)) {
##     ## obs model for spawners
##     Sp[t] <- max(1,tot_Run[t] * (1 - dat_harv[t]));
##     ln_Sp[t] <- log(Sp[t]);
##     ln_dat_esc[t] ~ dnorm(ln_Sp[t], tau_s);
##     lp_esc[t] <- logdensity.norm(ln_dat_esc[t],ln_Sp[t], tau_s);
##   }
##
## } ## end model description
```

Fitting the models and generating the one year ahead forecasts

For the most recent 10 years (2011 - 2020), 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 = 5e5,#200000,
  burn = 2e5,#100000,
  thin = 400 #100
)
## total number of MCMC samples after burnin
mcmc_samp <- mcmc_ctrl$length*mcmc_ctrl$chains/mcmc_ctrl$thin
```

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

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 52
##    Unobserved stochastic nodes: 68
##    Total graph size: 993
##
## Initializing model
##
##
## Parallel computation in progress
```