ESS 575: Dynamic Models Lab

Team England

01 December, 2022

Contents

Team England:

- Caroline Blommel
- Carolyn Coyle
- Bryn Crosby
- George Woolsey

cblommel@mail.colostate.edu, carolynm@mail.colostate.edu, brcrosby@rams.colostate.edu, george.woolsey@colostate.edu

Motivation

The Eurasian lynx (*Lynx lynx*) is a medium-sized predator with broad distribution in the boreal forests of Europe and Siberia. The lynx is classified as a threatened species throughout much of its range and there is controversy about the legal harvest of lynx in Sweden. Proponents of harvest argue that allowing hunting of lynx reduces illegal kill (poaching). Moreover, Sweden is committed to regulate lynx numbers to prevent excessive predation on reindeer because reindeer are critical to the livelihoods of indigenous pastoralists, the Sami. Many environmentalists oppose harvest, however, arguing that lynx are too rare to remove their fully protected status. A similar controversy surrounds management of wolves in the Western United States.

Fig. 1. A forecasting model for the abundance of lynx helps managers make decisions that can be justified to citizens. The model you will develop today is not a toy. It is currently used in Sweden and Norway to manage Lynx (H. Andren, N. T. Hobbs, M. Aronsson, H. Broseth, G. Chapron, J. D. C. Linnell, J. Odden, J. Persson, and E. B. Nilsen. Harvest models of small populations of a large carnivore using Bayesian forecasting. Ecological Applications, 30(3):e02063, 2020.)

You have data on the number of lynx family groups censused in a management unit as well as annual records of lynx harvested from the unit. You will model the population using the deterministic model:

$$N_t = \lambda(N_{t-1} - H_{t-1})$$

where N_t is the true, unobserved abundance of lynx and H_{t-1} is the number of lynx harvested during $t-1$ to t. The parentheses in this expression reflect the fact that harvest occurs immediately after census, such that the next years population increment comes from the post-harvest population size.

ADVANCED (for the population modelers) What would be the model if harvest occurred immediately before census? Three months after census? Continuously throughout the year?

Assume the harvest (H_t) is and the number of family groups (y_t) are observed without error. Harvest is closely regulated and all hunters who harvest a lynx are required by law to register the animal with the county. You are entitled to make the assumption that family groups are observed without error because your Scandinavian colleagues are amazing snow trackers and do a good job of estimating the number of family groups (if not the number of lynx) in a management region. The challenge in this problem is that the observations of lynx abundance (family groups) are not the same as the observation of harvest (number of lynx). Fortunately, you have prior information, hard won from radio-telemetry, on the proportional relationship between number of family groups and number of lynx in the population, i.e:

$$\phi = \frac{f}{N}$$

where f is the number of family groups and N is the population size, mean $\phi = 0.163$ with standard deviation of the mean = 0.012.

R libraries needed for this lab

You need to load the following libraries. Set the seed to 10 to compare your answers to ours. The data for this problem is located in the `LynxFamilies` data frame of the `BayesNSF` package.

```
# bread-and-butter
library(tidyverse)
library(lubridate)
library(viridis)
library(scales)
library(latex2exp)
# visualization
library(cowplot)
library(kableExtra)
# jags and bayesian
library(rjags)
library(MCMCvis)
library(HDInterval)
library(BayesNSF)
#set seed
set.seed(10)
```

Generating an Informed Prior for ϕ

We've provided you with a useful moment matching function below for converting the mean and standard deviation of ϕ to the parameters for the beta distribution you will use as an informed prior on ϕ.

```
# Function to get beta shape parameters from moments
shape_from_stats <- function(mu = mu.global, sigma = sigma.global) {
  a <-(mu^2 - mu^3 - mu * sigma^2) / sigma^2
  b <- (mu - 2 * mu^2 + mu^3 - sigma^2 + mu*sigma^2) / sigma^2
  shape_ps <- c(a, b)
  return(shape_ps)
}

# get parameters for distribution of population multiplier, 1/p
shapes = shape_from_stats(.163, .012)

# check prior on p using simulated data from beta distribution
x = seq(0, 1, .001)
p = dbeta(x, shapes[1], shapes[2])
plot(x, p, typ = "l", xlim = c(0, 1))
```

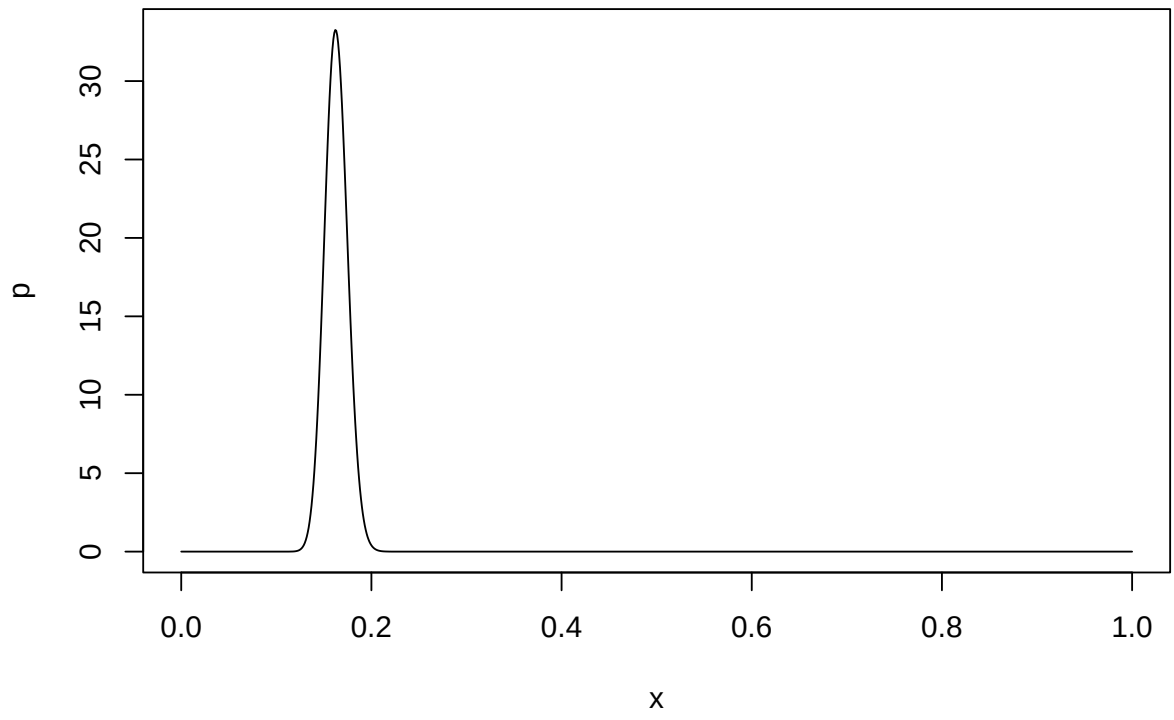
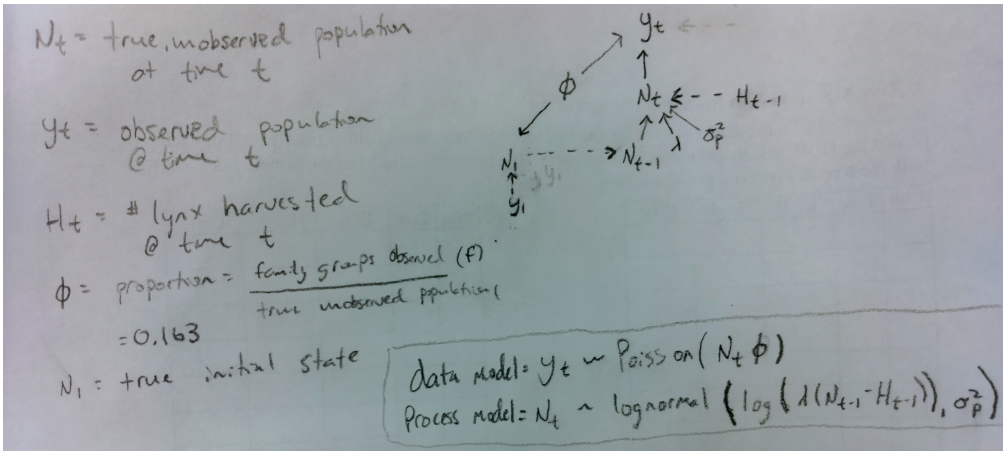


Diagram the Bayesian network

Question 1

Develop a hierarchical Bayesian model (also called a state space model) of the lynx population in the management unit. Diagram the Bayesian network (the DAG) of knowns and unknowns and write out the posterior and factored joint distribution. Use a lognormal distribution to model the true lynx population size over time. Use a Poisson distribution for the data model relating the true, unobserved state (the total population size) to the observed data (number of family groups).

Bayesian network (the DAG)



Bayesian network for lynx population

Process model:

$$N_t \sim \mathsf{lognormal}\Big(\log\big(\lambda(N_{t-1} - H_{t-1})\big), \sigma_p^2\Big)$$

Data model:

$$y_t \sim \mathsf{Poisson}\big(N_t \cdot \phi\big)$$

Posterior and Joint:

$$
\begin{aligned}
\left[\boldsymbol{N}, \phi, \lambda, \sigma_p^2 \mid \boldsymbol{y}\right] \propto & \prod_{t=2}^{T} \mathsf{Poisson}\left(y_t \mid N_t \cdot \phi\right) \\
& \times \mathsf{lognormal}\left(N_t \;\middle|\; \log\left(\lambda(N_{t-1} - H_{t-1})\right), \sigma_p^2\right) \\
& \times \mathsf{normal}\left(N_1 \mid \frac{y_1}{\phi}\right) \\
& \times \mathsf{beta}\left(\phi \mid 154, 792\right) \\
& \times \mathsf{uniform}\left(\lambda \mid 0, 1\right) \\
& \times \mathsf{uniform}\left(\sigma_p^2 \mid 0, 1\right)
\end{aligned}
$$

Question 2

An alternative approach, which is slightly more difficult to code, is to model the process as:

$$\text{negative binomial}(N_t \mid \lambda(N_{t-1} - H_{t-1}), \rho))$$

and model the data as:

$$\text{binomial}(y_t \mid \text{round}(N_t \cdot \phi), p)$$

where p is a detection probability. Explain why this second formulation *might* be better than the formulation you are using. (It turns out they give virtually identical results.)

Response

Using the negative binomial distribution for the process model would model the true population (N_t) as a count value (i.e., integer) occurring randomly over time or space. By comparison, the lognormal distribution would treat the true population (N_t) as a continuous quantity and it is not possible to have a "partial" individual (e.g., 0.5 of a lynx) in a population. Using the binomial distribution for the data model would allow for the adjustment of the variance independently from the mean which would allow us to account for the possibility that the data was not observed without error.

Fitting the Model

Now you'll estimate the marginal posterior distribution of the unobserved, true state over time ($\boldsymbol{N}$), the parameters in the model λ and ϕ as well as the process variance and observation variance. You'll also summarize the marginal posterior distributions of the parameters and unobserved states. A note about the data. Each row in the data file gives the observed number of family groups for that year in column 2 and that year's harvest in column 3. The harvest in each row influences the population size in the next row. So, for example, the 2016 harvest influences the 2017 population size.

Before you begin it's very helpful to use simulated data to the verify initial values and model. We simulate the true state by choosing some biologically reasonable values for model parameters and "eyeballing" the fit of the true state to the data. You can then use these simulated values for initial conditions (see the `inits` list below). This is of particular importance because failing to give reasonable initial conditions for

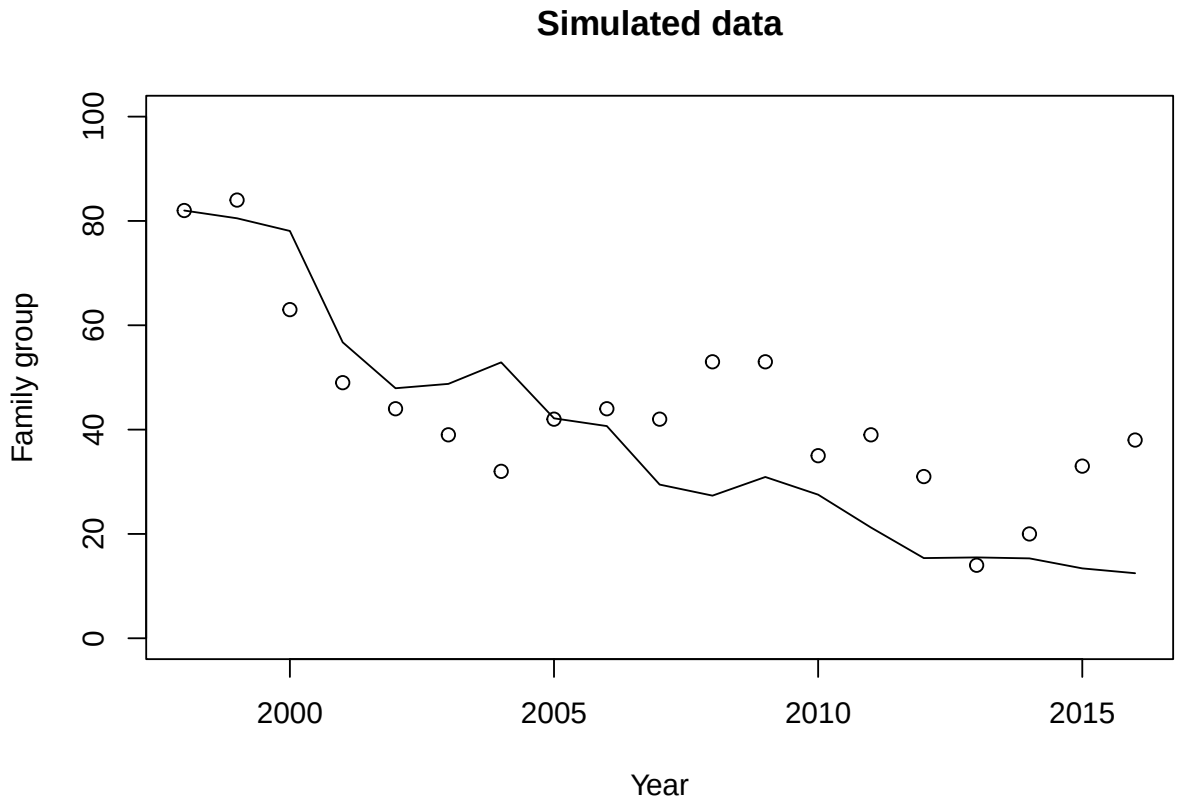
dynamic models can cause problems in model fitting. Remember, supply initial conditions for *all* unobserved quantities in the posterior distribution (even those that do not have priors).

```
y <- BayesNSF::LynxFamilies
endyr <- nrow(y)
n <- numeric(endyr + 1)
mu <- numeric(endyr + 1)
fg <- numeric(endyr + 1)
phi <- 0.16
lambda <- 1.07
sigma.p <- 0.2

n[1] <- y$census[1] / phi # n in the unit of individuals
mu[1] <- n[1] # mean from deterministic model to simulate
fg[1] <- n[1] * phi # Nt in the unit of

for (t in 2:(endyr + 1)) {
  mu[t] <- lambda * (n[t - 1] - y$harvest[t - 1])
  n[t] <- rlnorm(1, log(mu[t]), sigma.p)
  fg[t] <- n[t] * phi
}

plot(y$year, y$census, ylim = c(0, 100), xlab = "Year", ylab = "Family group", main = "Simulated data")
lines(y$year, fg[1:length(y$year)])
```

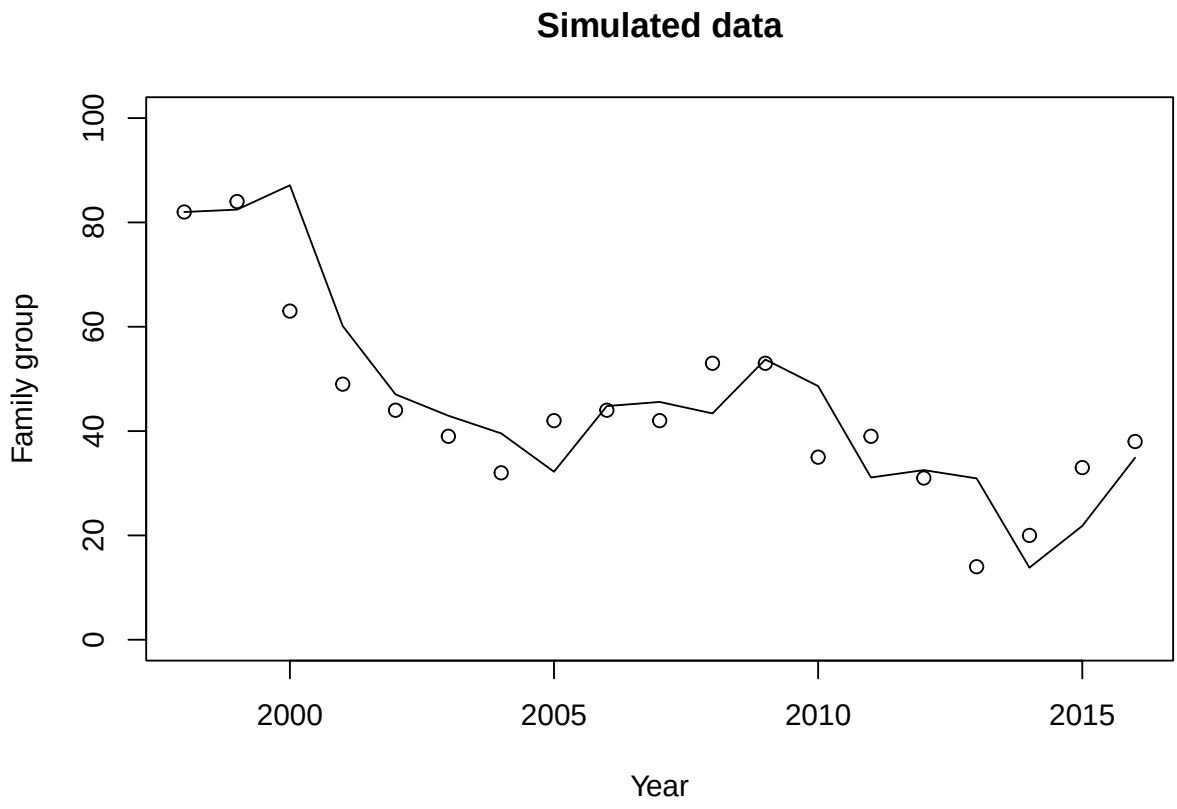


Visually match simulated data with observations for initial conditions:

```
## visually match simulated data with observations for initial conditions
endyr = nrow(y)
n = numeric(endyr + 1)
mu = numeric(endyr + 1) #use this for family groups
lambda = 1.1
sigma.p = .00001
n[1] = y$census[1]

for(t in 2:(endyr + 1)) {
  n[t] <- lambda * (y$census[t - 1] - .16 * y$harvest[t - 1])  # use this for family groups
}

plot(y$year, y$census, ylim = c(0, 100), xlab = "Year", ylab = "Family group", main = "Simulated data")
lines(y$year, n[1:length(y$year)])
```



Here's your starting code:

```
data = list(
    y.endyr = endyr,
    y.a = shapes[1],
    y.b = shapes[2],
```

```
    y.H = y$harvest,
    y = y$census)

inits = list(
    list(lambda = 1.2, sigma.p = .01, N = n),
    list(lambda = 1.01,sigma.p = .2, N = n * 1.2),
    list(lambda = .95, sigma.p = .5, N = n * .5))
```

Question 1

Write the JAGS model to estimate the marginal posterior distribution of the unobserved, true state over time ($\boldsymbol{N}$), the parameters in the model λ and ϕ as well as the process variance and observation variance. Include a summary the marginal posterior distributions of the parameters and unobserved states.

JAGS Model

Write out the JAGS code for the model.

```
## JAGS Model
model{
  ############################
  # priors
  ############################
    phi ~ dbeta(y.a, y.b)
    lambda ~ dunif(0, 10)
    sigma.p ~ dunif(0, 100)
    tau <- 1/sigma.p^2
    # initial conditions informed priors
    N[1] ~ dlnorm(log(y[1]/phi) , tau)
    fam_grps[1] <- N[1] * phi # mean family groups (mu)

  ############################
  # likelihood
  ############################
    # Process model:
    for(t in 2:(y.endyr+1)){
      alpha[t] <- lambda * (N[t-1] - y.H[t-1])
      N[t] ~ dlnorm(log(max(alpha[t], 0.000001)), tau) # can't take the log of alpha <= 0
      # calculate the mean for use in the data model
        # include in this loop to get the forecast value for #6
      fam_grps[t] <- N[t] * phi # mean family groups (mu)
    }
    # Data model:
    for(t in 2:y.endyr){
      # returns density (for continuous) because l.h.s. is data (deterministic b/c defined in data)
        y[t] ~ dpois(fam_grps[t])
    }

  ############################
  # Derived quantities
  ############################
    # sum of squares calculation
```

```
    # have to put this in own loop because y[1] defined separately
    for(t in 1:y.endyr){
      # returns random number generator because l.h.s. is not data (i.e. it is unknown: stochastic no
        y_sim[t] ~ dpois(fam_grps[t])
      # sum of squares
        sq[t] <- (y[t]-fam_grps[t])^2
        sq_sim[t] <- (y_sim[t]-fam_grps[t])^2
      # autocorrelation
        # Assure yourself that the process model adequately accounts for...
          # ...temporal autocorrelation in the residuals - allowing the assumption...
          # ...that they are independent and identically distributed.
          # To do this, include a derived quantity:
        e[t] <- (y[t]-fam_grps[t])
    }
  #posterior predictive checks
    # test statistics y
    mean_y <- mean(y)
    sd_y <- sd(y)
    fit_y <- sum(sq)
    # test statistics y_sim
    mean_y_sim <- mean(y_sim)
    sd_y_sim <- sd(y_sim)
    fit_y_sim <- sum(sq_sim)
    # p-values
    p_val_mean <- step(mean_y_sim - mean_y)
    p_val_sd <- step(sd_y_sim - sd_y)
    p_val_fit <- step(fit_y_sim - fit_y)
}
```

Implement JAGS Model

```
#########################################################################
# insert JAGS model code into an R script
#########################################################################
{ # Extra bracket needed only for R markdown files - see answers
  sink("LynxJAGS.R") # This is the file name for the jags code
  cat("
  ## JAGS Model
  model{
    ############################
    # priors
    ############################
      phi ~ dbeta(y.a, y.b)
      lambda ~ dunif(0, 10)
      sigma.p ~ dunif(0, 100)
      tau <- 1/sigma.p^2
      # initial conditions informed priors
      N[1] ~ dlnorm(log(y[1]/phi) , tau)
      fam_grps[1] <- N[1] * phi # mean family groups (mu)

    ############################
    # likelihood
```

```
    ############################
      # Process model:
      for(t in 2:(y.endyr+1)){
          alpha[t] <- lambda * (N[t-1] - y.H[t-1])
          N[t] ~ dlnorm(log(max(alpha[t], 0.000001)), tau) # can't take the log of alpha <= 0
          # calculate the mean for use in the data model
            # include in this loop to get the forecast value for #6
          fam_grps[t] <- N[t] * phi # mean family groups (mu)
        }
      # Data model:
      for(t in 2:y.endyr){
        # returns density (for continuous) because l.h.s. is data (deterministic b/c defined in data)
            y[t] ~ dpois(fam_grps[t])
      }

    ############################
    # Derived quantities
    ############################
      # sum of squares calculation
        # have to put this in own loop because y[1] defined separately
        for(t in 1:y.endyr){
          # returns random number generator because l.h.s. is not data (i.e. it is unknown: stochastic
            y_sim[t] ~ dpois(fam_grps[t])
          # sum of squares
            sq[t] <- (y[t]-fam_grps[t])^2
            sq_sim[t] <- (y_sim[t]-fam_grps[t])^2
          # autocorrelation
            # Assure yourself that the process model adequately accounts for...
              # ...temporal autocorrelation in the residuals - allowing the assumption...
              # ...that they are independent and identically distributed.
              # To do this, include a derived quantity:
            e[t] <- (y[t]-fam_grps[t])
        }
      #posterior predictive checks
        # test statistics y
        mean_y <- mean(y)
        sd_y <- sd(y)
        fit_y <- sum(sq)
        # test statistics y_sim
        mean_y_sim <- mean(y_sim)
        sd_y_sim <- sd(y_sim)
        fit_y_sim <- sum(sq_sim)
        # p-values
        p_val_mean <- step(mean_y_sim - mean_y)
        p_val_sd <- step(sd_y_sim - sd_y)
        p_val_fit <- step(fit_y_sim - fit_y)
  }
  ", fill = TRUE)
  sink()
}
#######################################################################
# implement model
#######################################################################
```

```
# specify 3 scalars, n.adapt, n.update, and n.iter
# n.adapt = number of iterations that JAGS will use to choose the sampler
  # and to assure optimum mixing of the MCMC chain
n.adapt = 1000
# n.update = number of iterations that will be discarded to allow the chain to
#   converge before iterations are stored (aka, burn-in)
n.update = 10000
# n.iter = number of iterations that will be stored in the
  # final chain as samples from the posterior distribution
n.iter = 10000
#######################
# Call to JAGS
#######################
jm = rjags::jags.model(
  file = "LynxJAGS.R"
  , data = data
  , inits = inits
  , n.chains = length(inits)
  , n.adapt = n.adapt
)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 18
##    Unobserved stochastic nodes: 42
##    Total graph size: 281
##
## Initializing model
```

```
stats::update(jm, n.iter = n.update, progress.bar = "none")
# save the coda object (more precisely, an mcmc.list object) to R as "zm"
zm = rjags::coda.samples(
  model = jm
  , variable.names = c(
      # parameters
      "phi"
      , "lambda"
      , "sigma.p"
      # process model
      , "N"
      # # test statistics
      , "mean_y"
      , "sd_y"
      , "fit_y"
      , "mean_y_sim"
      , "sd_y_sim"
      , "fit_y_sim"
      # # p-values
      , "p_val_mean"
      , "p_val_sd"
      , "p_val_fit"
```

```
      # derived quantities
      , "e"
      , "fam_grps"
    )
  , n.iter = n.iter
  , n.thin = 1
  , progress.bar = "none"
)
```

Summary of the marginal posterior distributions of the parameters

```
# summary
MCMCvis::MCMCsummary(zm, params = c(
    "phi"
    , "lambda"
    , "sigma.p"
    )
  )
```

```
##              mean         sd       2.5%       50%     97.5% Rhat n.eff
## phi     0.1647447 0.01229764 0.14138020 0.1643964 0.1896024    1   945
## lambda  1.0637611 0.04693322 0.97390355 1.0614679 1.1642356    1  9888
## sigma.p 0.1667144 0.05925085 0.07724555 0.1587550 0.3042247    1  2065
```

Summary of the marginal posterior distributions of the latent state

```
# summary
MCMCvis::MCMCsummary(zm, params = c("N"))
```

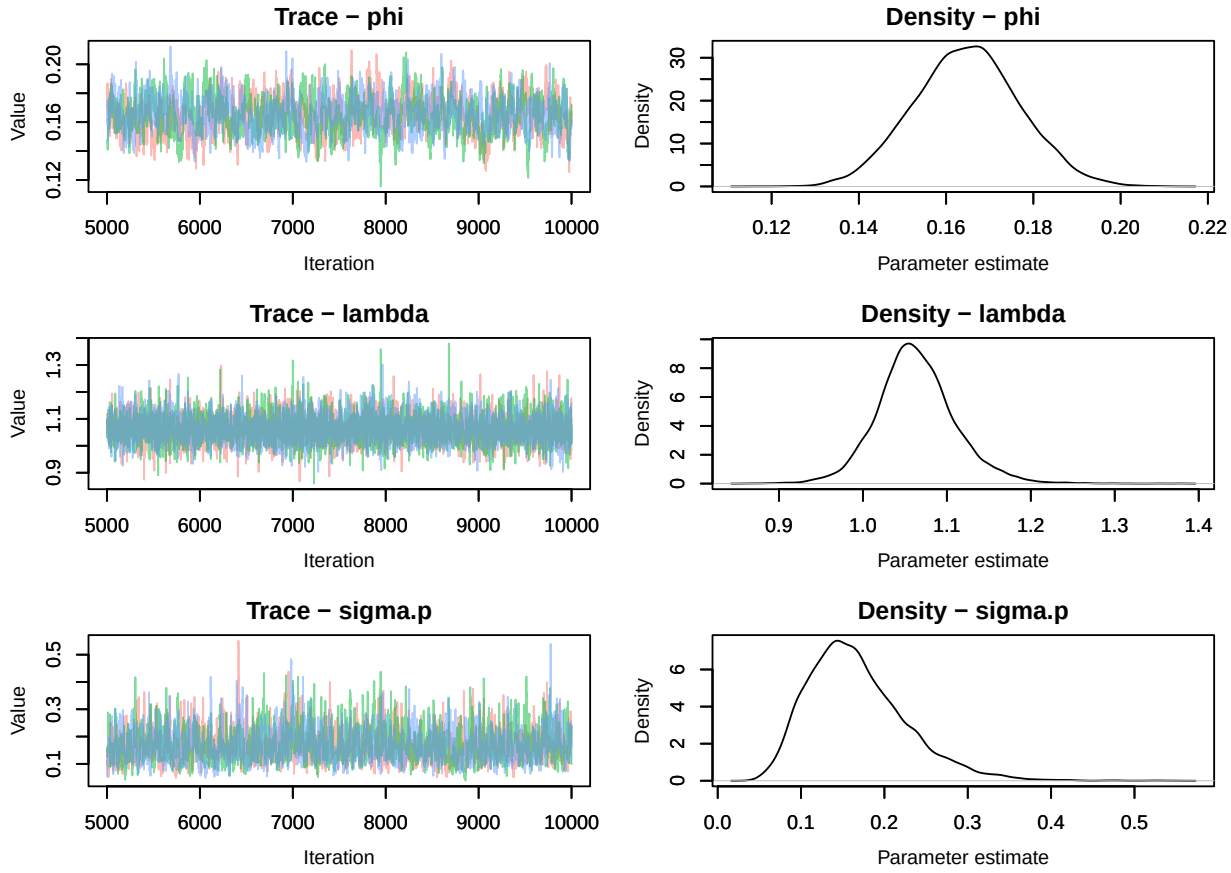
```
##           mean       sd      2.5%      50%    97.5% Rhat n.eff
## N[1]  496.7671 76.43734 368.28871 489.2131 671.9860    1  2559
## N[2]  469.2970 57.73169 368.85087 465.2175 594.3034    1  1797
## N[3]  389.6863 45.40879 309.02536 387.1064 487.7527    1  1843
## N[4]  313.4897 38.55823 244.11508 311.1846 395.6425    1  1876
## N[5]  272.3707 34.80090 209.94288 270.7097 345.8764    1  1753
## N[6]  242.6125 31.95979 184.15004 241.0622 310.7352    1  1879
## N[7]  228.4889 31.41910 170.90217 227.0574 293.2337    1  1813
## N[8]  247.3110 32.59720 189.29366 245.2813 317.6921    1  1859
## N[9]  266.0194 33.73012 205.12501 264.1973 337.9719    1  2040
## N[10] 274.2126 34.56559 211.57750 272.2495 346.9467    1  1984
## N[11] 303.5067 37.97527 237.73284 300.5531 386.7946    1  1864
## N[12] 299.5619 36.47622 236.49676 296.5076 379.8717    1  1809
## N[13] 240.8727 29.97348 186.35808 239.2461 304.6253    1  2142
## N[14] 219.9535 27.90499 172.09469 217.6392 281.3253    1  1819
## N[15] 165.1271 24.30190 122.55966 163.5129 218.5067    1  1974
## N[16] 136.3102 21.85679  94.72411 135.6810 181.4745    1  1971
## N[17] 146.1303 22.40772 104.78011 145.0142 193.2133    1  2073
## N[18] 184.4152 26.27319 138.58921 182.2771 241.6332    1  2249
## N[19] 206.6711 33.67579 148.84143 203.9063 280.8852    1  2709
## N[20] 182.2055 52.40869 104.14111 174.2172 306.6800    1  5430
```

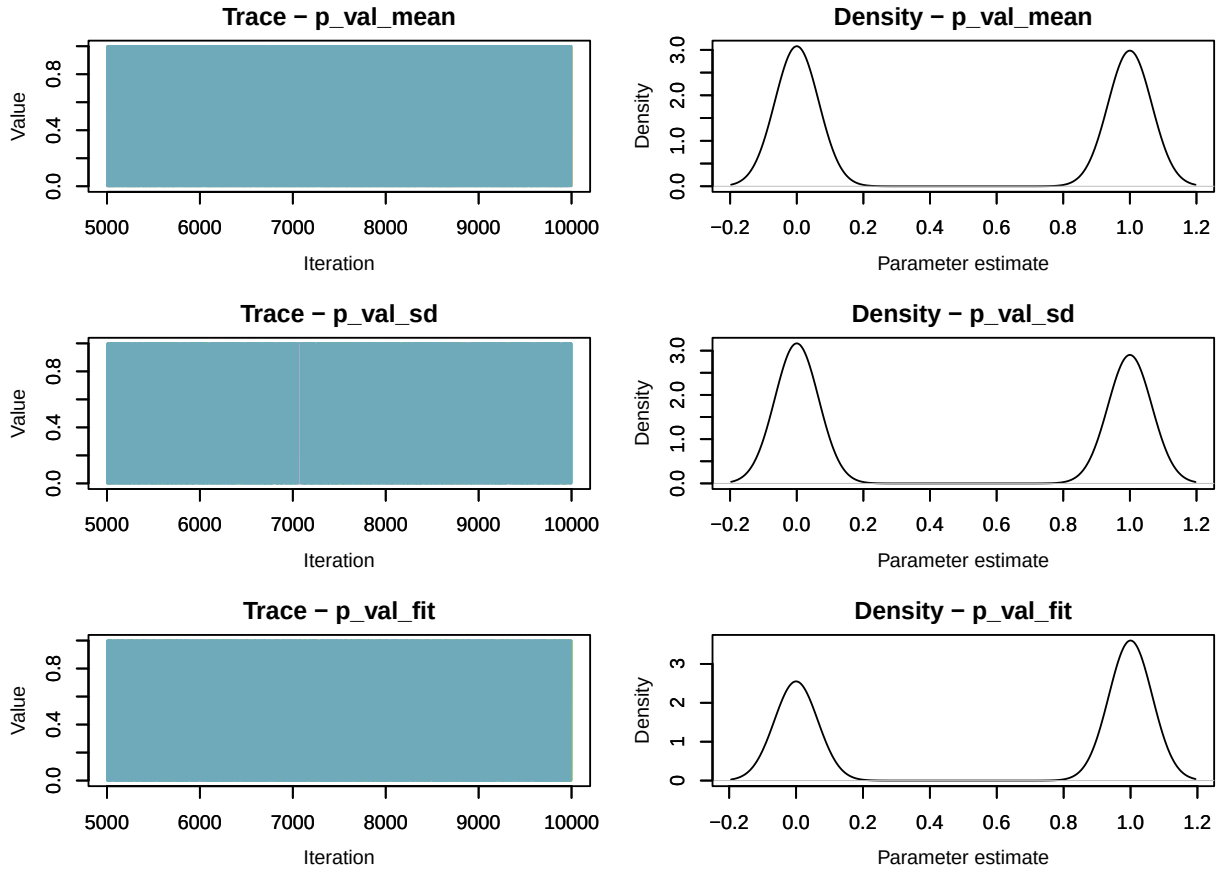
Question 2

Check MCMC chains for model parameters, process variance, and latent states for convergence. This will probably require using the `excl` option in `MCMCsummary`

Trace plots

```
# trace plot
MCMCvis::MCMCtrace(zm, params = c(
    # parameters
    "phi"
    , "lambda"
    , "sigma.p"
    # p-values
    , "p_val_mean"
    , "p_val_sd"
    , "p_val_fit"
  )
  , pdf = FALSE
)
```





Question 4

Conduct posterior predictive checks by simulating a new dataset for family groups (f_t) at every MCMC iteration. Calculate a Bayesian p value using the sums of squared discrepancy between the observed and the predicted number of family groups based on observed and simulated data,

$$T^{observed} = \sum_{t=1}^{n} \left(f_t^{observed} - N_t\phi\right)^2$$

$$T^{model} = \sum_{t=1}^{n} \left(f_t^{simulated} - N_t\phi\right)^2$$

The Bayesian p value is the proportion of MCMC iterations for which $T^{model} > T^{obs}$

Posterior predictive check - Test Statistics

```
# summary
MCMCvis::MCMCsummary(zm, params = c(
```

```
      # test statistics
      "mean_y"
      , "mean_y_sim"
      , "sd_y"
      , "sd_y_sim"
      , "fit_y"
      , "fit_y_sim"
    )
    , n.eff = FALSE
  )
```

```
##                 mean         sd      2.5%       50%      97.5% Rhat
## mean_y      44.05263   0.000000  44.05263  44.05263   44.05263  NaN
## mean_y_sim  44.00696   2.201593  39.78947  43.94737   48.42105    1
## sd_y        17.75911   0.000000  17.75911  17.75911   17.75911  NaN
## sd_y_sim    17.78184   2.710145  13.05252  17.58804   23.64416    1
## fit_y      766.80379 306.642985 367.71554 719.00265 1437.42054    1
## fit_y_sim  835.41820 291.364736 376.70440 797.94691 1509.83470    1
```

Posterior predictive check - p-values

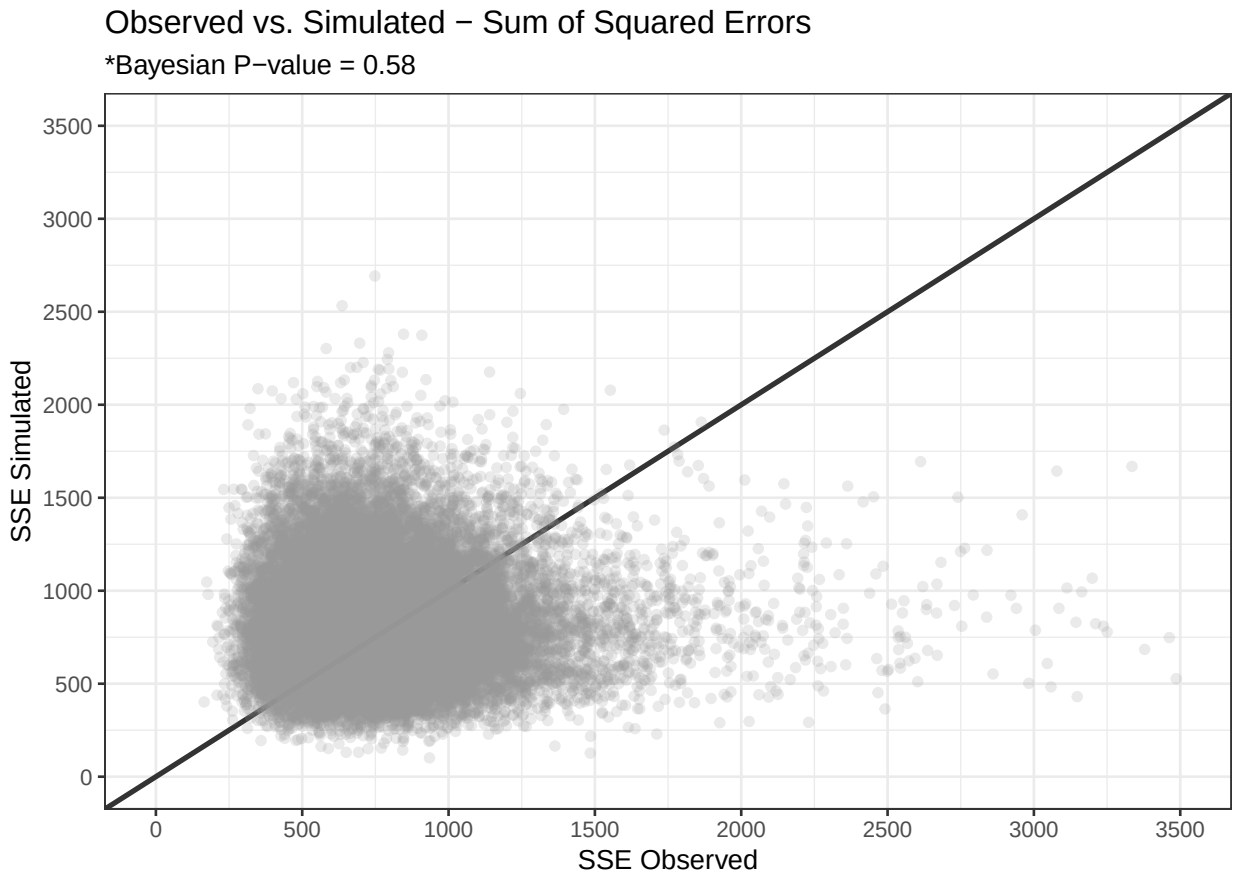
```
# summary
MCMCvis::MCMCsummary(zm, params = c(
      # p-values
      "p_val_mean"
      , "p_val_sd"
      , "p_val_fit"
    )
  )
```

```
##                 mean        sd 2.5% 50% 97.5% Rhat n.eff
## p_val_mean 0.4894333 0.4998967    0   0     1    1 14094
## p_val_sd   0.4744667 0.4993559    0   0     1    1 11538
## p_val_fit  0.5805333 0.4934800    0   1     1    1 17834
```

Posterior predictive check - Sum of Squared Errors Plot

```
# PLOT
MCMCvis::MCMCchains(zm, params = c("fit_y", "fit_y_sim")) %>%
  data.frame() %>%
ggplot(data = .) +
  geom_abline(intercept = 0, slope = 1, lwd = 1, color = "gray20") +
  geom_point(
    mapping  = aes(x = fit_y, y = fit_y_sim)
    , color = "gray60"
    , alpha = 0.2
  ) +
  scale_y_continuous(
    limits = c(
```

```
      0
    , 3500
  )
  , breaks = scales::extended_breaks(n=10)
) +
scale_x_continuous(
  limits = c(
    0
    , 3500
  )
  , breaks = scales::extended_breaks(n=10)
) +
xlab("SSE Observed") +
ylab("SSE Simulated") +
labs(
  title = "Observed vs. Simulated - Sum of Squared Errors"
  , subtitle = paste0(
      "*Bayesian P-value = "
      , MCMCvis::MCMCchains(zm, params = c("p_val_fit")) %>% mean() %>% scales::comma(accuracy = 0.01)
    )
) +
theme_bw()
```



The Bayesian p-value of 0.58 indicates good fit for the lynx population.

Assure yourself that the process model adequately accounts for temporal autocorrelation in the residuals—allowing the assumption that they are independent and identically distributed. To do this, include a derived quantity

$$e_t = y_t - N_t\phi$$

in your JAGS code and coda object. Use the following code or something like it to examine how autocorrelation in the residuals changes with time lag.

```
acf(unlist(MCMCpstr(zm, param = "e", func = mean)), main = "", lwd = 3, ci = 0)
```

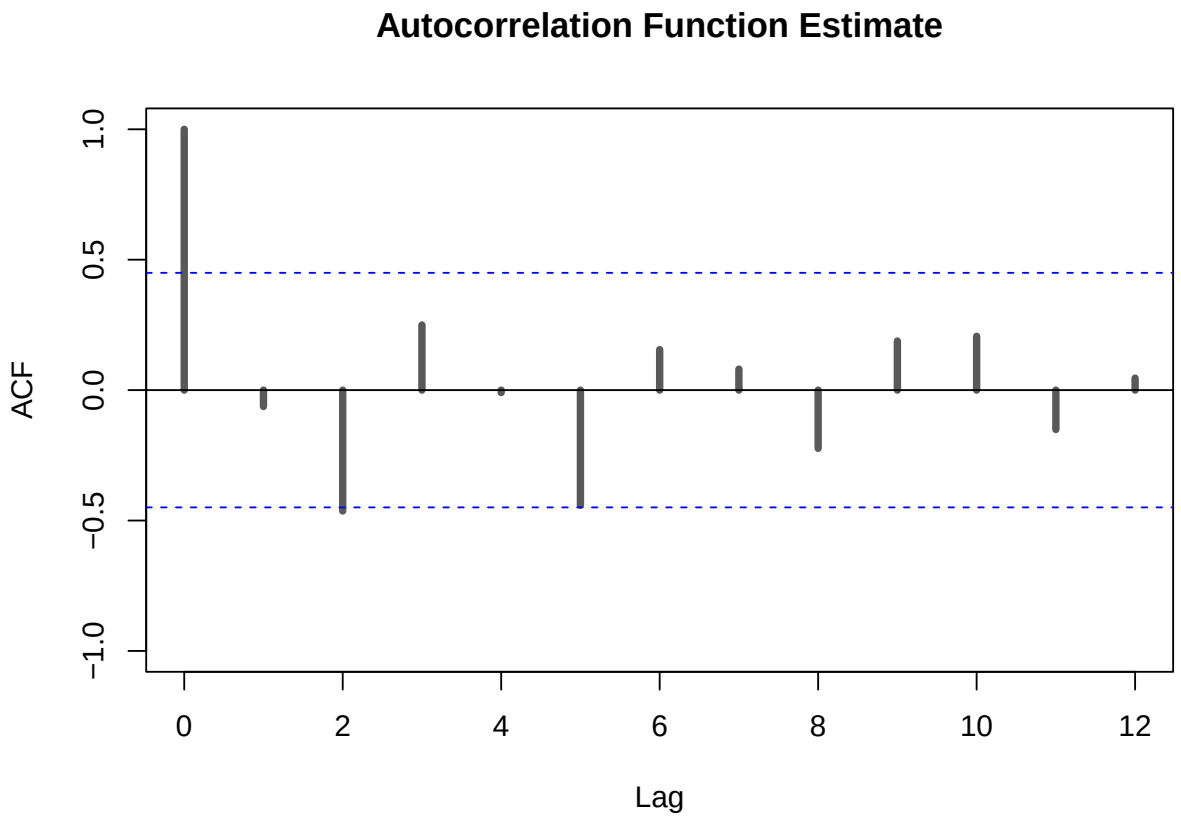
Summary of the marginal posterior distributions of the model error

```
# summary
MCMCvis::MCMCsummary(zm, params = c("e"))
```

```
##              mean        sd       2.5%        50%     97.5% Rhat n.eff
## e[1]    0.6289148 10.769199 -24.237223  1.7049226 19.213948    1  9397
## e[2]    7.1343037  7.275449  -8.462388  7.6159966 19.914442    1  6968
## e[3]   -0.8428119  5.684539 -12.594393 -0.6236300  9.907187    1 12377
## e[4]   -2.3557274  4.960343 -12.451593 -2.2660983  7.159820    1 10577
## e[5]   -0.6165049  4.544530  -9.803276 -0.4949165  7.987968    1  9224
## e[6]   -0.7420126  4.245331  -9.258295 -0.6979701  7.368418    1  7497
## e[7]   -5.4282067  4.251750 -13.889650 -5.4089441  2.785827    1  5195
## e[8]    1.4954980  4.251061  -7.255990  1.6396519  9.513253    1 10296
## e[9]    0.4179074  4.448930  -8.804225  0.5409618  8.744562    1 11442
## e[10] -2.9288154  4.598694 -12.329252 -2.8058928  5.705724    1 11124
## e[11]  3.2777623  4.945458  -7.405626  3.6051419 12.068832    1  9941
## e[12]  3.9117617  4.803784  -6.594982  4.2266054 12.417122    1  8660
## e[13] -4.4744787  4.036513 -12.798863 -4.4127953  3.178736    1  9554
## e[14]  2.9608563  3.709412  -5.104279  3.2134618  9.511705    1  8211
## e[15]  3.9626900  3.284408  -3.040939  4.1512959  9.929013    1  7288
## e[16] -8.3322194  3.162672 -14.485767 -8.3581371 -2.065336    1  4102
## e[17] -3.9399953  3.186114 -10.301021 -3.8913497  2.214272    1  5293
## e[18]  2.7859003  3.665896  -5.073396  3.0183546  9.389276    1  7212
## e[19]  4.1387515  4.893452  -6.637413  4.5026757 12.721993    1  6312
```

Autocorrelation Function Estimate Plot

```
MCMCvis::MCMCpstr(zm, param = "e", func = mean) %>%
  unlist() %>%
  stats::acf(., main = "Autocorrelation Function Estimate"
             , lwd = 4
             , col = "gray35"
             # , ci = 0
             , type = "correlation"
             , ylim = c(-1,1)
  )
```



Question 5

Write a paragraph describing how to interpret the plot produced by this function.

Response

The autocorrelation function (ACF) estimate shows the autocorrelation of the model error over time. It is an estimate of the temporal dependence of the model errors and is calculated as a correlation coefficient (ρ) where $-1 \leq \rho \leq 1$. Ideally, the plot of the ACF would show no pattern (e.g. decreasing or increasing) over time (a-axis) with correlation values (ρ) not near -1 or 1. The plot of the ACF above reveals that this chain is not highly autocorrelated, which means that the assumption of independent errors does hold for these data.

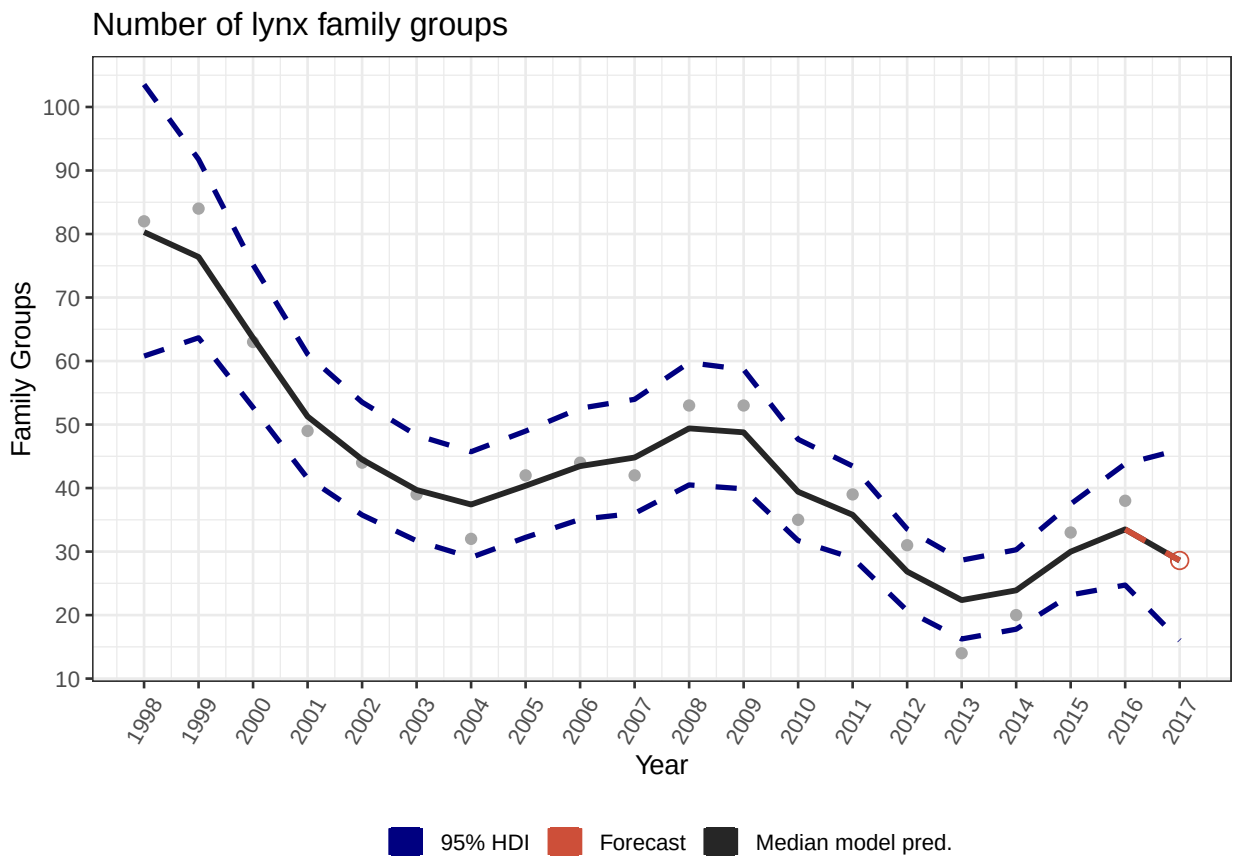
Question 6

Plot the median of the marginal posterior distribution of the number of lynx family groups over time (1998-2016) including a highest posterior density interval. Include your forecast for 2017 (the predictive process distribution) in this plot.

Median model prediction of lynx family groups plot

```
# data
dta_temp <- dplyr::bind_cols(
  t = c(y$year, max(y$year)+1)
  , census = c(y$census, NA)
  , median_fam_grps = MCMCvis::MCMCpstr(zm, params = "fam_grps", func = median) %>% unlist()
  , MCMCvis::MCMCpstr(zm, params = "fam_grps", func = function(x) HDInterval::hdi(x, credMass = 0.95)) %
      as.data.frame()
) %>%
  dplyr::mutate(
    census_fcast = ifelse(dplyr::row_number()==dplyr::n(), median_fam_grps, NA)
    , census_fcast2 = ifelse(dplyr::row_number()>=dplyr::n()-1, median_fam_grps, NA)
  )
# plot
ggplot(data = dta_temp, mapping = aes(x = t)) +
  geom_point(
    mapping = aes(y = census)
    , color = "gray65"
    , shape = 16
    , size = 2
  ) +
  geom_point(
    mapping = aes(y = census_fcast, color = "Forecast")
    # , color = "firebrick"
    , shape = 1
    , size = 3
  ) +
  geom_line(
    mapping = aes(y = median_fam_grps, color = "Median model pred.")
    # , color = "black"
    , lwd = 1.1
  ) +
  geom_line(
    mapping = aes(y = census_fcast2, color = "Forecast")
    , lwd = 1.1
    , linetype = "dashed"
  ) +
  geom_line(
    mapping = aes(y = fam_grps.upper, color = "95% HDI")
    # , color = "royalblue"
    , lwd = 1
    , linetype = "dashed"
  ) +
  geom_line(
    mapping = aes(y = fam_grps.lower, color = "95% HDI")
    , lwd = 1
    , linetype = "dashed"
  ) +
  scale_y_continuous(breaks = scales::extended_breaks(n=10)) +
  scale_x_continuous(breaks = scales::extended_breaks(n=15)) +
  scale_color_manual(values = c("navy", "tomato3", "gray15")) +
  xlab("Year") +
```

```
ylab("Family Groups") +
labs(
  title = "Number of lynx family groups"
) +
theme_bw() +
theme(
  legend.position = "bottom"
  , legend.direction = "horizontal"
  , legend.title = element_blank()
  , axis.text.x = element_text(angle = 60, vjust = 0.5, hjust = 0.5)
) +
guides(color = guide_legend(override.aes = list(shape = 15,size = 5)))
```



Question 6

Optional, but strongly recommended for those who seek to support policy and management with models Due to licensing constraints related to the time that it takes to properly issue hunting permits/licenses, Lynx harvest decisions are made before the population is censused, even though harvest actually occurs shortly after the census. Make a forecast of the number of family groups in 2018 assuming five alternative levels for 2017 harvest (0, 10, 25, 50, and 75 animals). Environmentalists and hunters have agreed on a acceptable range for lynx abundance in the unit, 26 - 32 family groups. Compute the probability that the post-harvest number of family groups will be below, within, and above this range during 2018. Tabulate these values. Hint: Set up a "model experiment" in your JAGS code where you forecast the number

of lynx family groups during 2018 under the specified levels of harvest. Extract the MCMC chains for the forecasted family groups(e.g., `fg.hat`) using `MCMCchains` Use the `ecdf` function on the R side to compute the probabilities that the forecasted number groups will be below, within, or above the acceptable range.

```
# try it
```