SFS 2023 Short Course – Bayesian Applications in Environmental and Ecological Studies with R and Stan

Song S. Qian

6/3/2023

Example 3 — Grass Carp Population Estimation

- Statistical method – Data augmentation

In many cases, describing the response variable model can be highly difficult, as the posterior distribution is often challenging to express using simple distribution models. However, this difficulty can often be overcome by incorporating data from unobservable variable(s). In such situations, a class of statistical methods known as data augmentation methods is commonly employed.

Let's assume that the posterior distribution $\pi(\theta \mid y)$ is not easily manageable. However, we discover that the posterior distribution of θ conditional on another variable z ($\pi(\theta \mid z, y)$) is more amenable to analysis. By utilizing the conditional probability equation, we can express the joint posterior distribution of θ and z as follows:

$$\pi(\theta, z \mid y) = \pi(\theta \mid z, y)\pi(z \mid y)$$

From this joint posterior, we can derive the marginal distribution of θ by integrating out the variable z:

$$\pi(\theta \mid y) = \int_z \pi(\theta \mid z, y)\pi(z \mid y)dz$$

The unobservable variable z is commonly referred to as a latent variable. An example where this approach is applicable is in the modeling of grass carp populations.

- Background Grass carp, a popular aquaculture species in Asia and an invasive nuisance species in the Great Lakes region, has been causing concerns since its reproduction in the Sandusky River, a Lake Erie tributary, was confirmed in 2016. To address this issue, a regional task force was established to develop control methods. One approach implemented was the use of electrofishing to physically remove the fish. To estimate the grass carp population in the area, we utilized capture data from the Sandusky River and developed a basic model. This example demonstrates how a Bayesian approach can be employed to systematically improve an imperfect or flawed model to generate valuable information.

The primary objective of estimating the grass carp population is to assess the effectiveness of the removal efforts. However, we encountered several challenges in obtaining statistically meaningful data. Firstly, grass carp remains relatively scarce in the region. Despite removing over 100 fish annually from the Sandusky River, the majority of sampling trips failed to capture any grass carp. When successful, the number of captures was generally low, typically ranging from 1 to occasionally 2, with a single instance of 3 captures. Even with a well-designed random sampling plan, we do not possess an existing statistical model suitable for analyzing this type of data.

```r
## Importing data initial data set used in Gouveia et al (2023)
gc_data <- read.csv(paste(dataDIR, "1820_GC_disch.csv", sep = "/"), header = T)
### updated with 2021 data and calculated sampling distance from 2020-2021
gc_data2 <- read.csv(paste(dataDIR, "sandusky_model_data2.csv", sep = "/"), header = T)
### 2020-2021 efishing only (without nets)
```

```r
gc_data3 <- read.csv(paste(dataDIR, "sandusky_model_efishonly.csv", sep = "/"), header = T)
### Combined efishing and efishing + nets 2022
gc_data4 <- read.csv(paste(dataDIR, "sandusky_combo_2022.csv", sep = "/"), header = T)
### efishing only 2022
gc_data5 <- read.csv(paste(dataDIR, "sandusky_efishonly_2022.csv", sep = "/"), header = T)
### removing empty lines
gc_data2 <- gc_data2[!is.na(gc_data2$count), ]
gc_data3 <- gc_data3[!is.na(gc_data3$count), ]
# convert segment letters to numbers
# wsection.data$segment<-as.numeric(wsection.data$segment)
gc_data$segment[gc_data$segment == "M"] <- 1
gc_data$segment[gc_data$segment == "LM"] <- 2
gc_data$segment[gc_data$segment == "UM"] <- 3
gc_data$segment[gc_data$segment == "FM"] <- 4
gc_data$segment <- as.numeric(gc_data$segment)

gc_data2$segment[gc_data2$segment == "M"] <- 1
gc_data2$segment[gc_data2$segment == "LM"] <- 2
gc_data2$segment[gc_data2$segment == "UM"] <- 3
gc_data2$segment[gc_data2$segment == "FM"] <- 4
gc_data2$segment <- as.numeric(gc_data2$segment)
san_seg2 <- c("M", "LM", "UM", "FM")

gc_data3$segment[gc_data3$segment == "M"] <- 1
gc_data3$segment[gc_data3$segment == "LM"] <- 2
gc_data3$segment[gc_data3$segment == "UM"] <- 3
gc_data3$segment[gc_data3$segment == "FM"] <- 4
gc_data3$segment <- as.numeric(gc_data3$segment)

gc_data4$segment[gc_data4$segment == "M"] <- 1
gc_data4$segment[gc_data4$segment == "LM"] <- 2
gc_data4$segment[gc_data4$segment == "UM"] <- 3
gc_data4$segment[gc_data4$segment == "FM"] <- 4
gc_data4$segment <- as.numeric(gc_data4$segment)

gc_data5$segment[gc_data5$segment == "M"] <- 1
gc_data5$segment[gc_data5$segment == "LM"] <- 2
gc_data5$segment[gc_data5$segment == "UM"] <- 3
gc_data5$segment[gc_data5$segment == "FM"] <- 4
gc_data5$segment <- as.numeric(gc_data5$segment)
```

```
## Warning: NAs introduced by coercion
```

```r
san_seg23 <- paste(san_seg2, " (", table(gc_data2$segment), ",", table(gc_data3$segment),
    ")", sep = "")

season <- function(x) return(ifelse(x <= 6, 1, ifelse(x < 9, 2, 3)))

## creating R date and Season columns
gc_data$Rdate <- as.Date(paste(gc_data$month, gc_data$day, gc_data$year, sep = "-"),
    format = "%m-%d-%Y")
gc_data$Season <- season(gc_data$month)
### not using `as.Date(gc_data$cdate, format = '%d/%m/%Y')` cdata format not
### consistent : prior to 2021, date format was '%d/%m/%Y', since 2021, the
### format was '%m/%d/%Y'

gc_data2$Rdate <- as.Date(paste(gc_data2$month, gc_data2$day, gc_data2$year, sep = "-"),
    format = "%m-%d-%Y")
gc_data2$Season <- season(gc_data2$month)

gc_data3$Rdate <- as.Date(paste(gc_data3$month, gc_data3$day, gc_data3$year, sep = "-"),
    format = "%m-%d-%Y")
gc_data3$Season <- season(gc_data3$month)

gc_data4$Rdate <- as.Date(paste(gc_data4$month, gc_data4$day, gc_data4$year, sep = "-"),
    format = "%m-%d-%Y")
gc_data4$Season <- season(gc_data4$month)

gc_data5$Rdate <- as.Date(paste(gc_data5$month, gc_data5$day, gc_data5$year, sep = "-"),
    format = "%m-%d-%Y")
gc_data5$Season <- season(gc_data5$month)
```

The initial model adopted for estimating the grass carp population was based on the N-mixture model. This model was chosen because the data for the response variable (number of captures) were generated through two distinct processes: 1. Fish movement, which resulted in an unknown number of individuals being present at the sampling site. 2. The data collection method (electrofishing), which was imperfect.

To account for these processes, the N-mixture model employs a latent variable approach. Assuming that there are N individuals present at the sampling site when electrofishing begins, the number of captures follows a binomial distribution:

$$y_i \mid N_i \sim Bin(p, N_i)$$

where p is the "detection" probability, the probability of capturing a fish when it is present. The unknown number N is modeled as a Poisson random variable:

$$N_i \sim Pois(\lambda_i)$$

Because the sampling site is limited in size, we can define λ_i as the expected number of individuals in the location. For a river, we define the population size in terms of population density (λ) measured in number of fish per kilometer. Hence, $\lambda_i = d_i \lambda$ where d_i is the linear sampling distance along the river during the ith sampling event.

In the context of this mixture model problem, the standard statistical approach is to establish the joint distribution of y_i, p, N_i, and λ as follows:

$$\Pr(Y = y_i, p, N_i, \lambda) = \Pr(Y = y_i \mid p, N_i) \Pr(N_i \mid \lambda) \Pr(\lambda) \Pr(p)$$

Since N_i is an unknown (latent) variable, we integrate it out to derive the marginal posterior density of p and λ:

$$\Pr(p, \lambda \mid y_i) = \int \Pr(Y = y_i \mid p, N_i) \Pr(N_i \mid \lambda) \Pr(\lambda) \Pr(p) dN_i$$

As N_i is an integer, the integral is now a summation:

$$\Pr(p, \lambda \mid y_i) = \sum_{n=y_i}^{\infty} \Pr(Y = y_i \mid p, n) \Pr(n \mid \lambda) \Pr(\lambda) \Pr(p)$$

In this equation, the first term in the summation represents a binomial probability function, the second term corresponds to a Poisson probability function, and the last two terms are the priors for λ and p, respectively. By evaluating this summation, we can obtain the marginal posterior density of p and λ given the observed data y_i.

First, it is important to note that the observed response variable data do not typically provide information about both p and λ. The parameter p is associated with the sampling method, while λ represents a characteristic of the target population. Therefore, it is appropriate to specify a meaningful prior distribution for p separately.

Secondly, the infinite upper bound in the summation is not computationally feasible. Realistically, it is highly unlikely to have more than 50 fish in the sampling site, which corresponds to a river segment of roughly 200-500 meters. Therefore, we can use 50 or 25 as reasonable upper bounds.

The log-likelihood function can be expressed as:

$$LL = \log\left(\sum_{y_i}^{n_{max}} \Pr(Y = y_i \mid p, n) \times \Pr(n \mid \lambda)\right)$$

Because

$$\Pr(Y = y_i \mid p, n) \times \Pr(n \mid \lambda) = e^{\log(\Pr(Y=y_i|p,n))+\log(\Pr(n|\lambda))}$$

we can use an efficient computation function such as `log_sum_exp` to write the log-likelihood function

```
model{
  int k;
  pd ~ beta(alpha, beta);
  for (i in 1:Nobs){
    vector[Nmax-y[i]+1] temp;
    for (j in y[i]:Nmax){
      k = j-y[i]+1;
      temp[k] = binomial_lpmf(y[i] | j, pd) +
                poisson_lpmf(j | lambda[i]);
    }
    target += log_sum_exp(temp);
  }
}
```

It is evident that the initial model assumption of constant fish density along the river is incorrect. To address this, we divided the river into four segments and assumed a constant density within each segment. While this assumption is still an approximation, it is considered "less wrong" given the available information.

In the implementation of the model with four river segments, the parameter λ is indexed by segments (λ_j, where j denotes the segment). We assume that the λ_j values are exchangeable, meaning they are different from each other, but we do not have prior knowledge about their specific differences. A natural Bayesian approach is to assign a single prior distribution for λ_j. Specifically, we assume $\log(\lambda_j) \sim N(\mu, \tau^2)$, where the common prior distribution $N(\mu, \tau^2)$ is referred to as the hyper-distribution, with hyperparameters μ and τ^2. The use of the log-transformed λ_j is due to it being the parameter of a Poisson distribution.

Similarly, when the model is implemented for multiple years, the parameter λ can be indexed by year, and the year-specific densities are assumed to be exchangeable. It is well-established in the statistics literature (e.g., Efron and Morris, 1977) that hierarchical modeling, which estimates segment- and year-specific parameters (λ_j), yields more accurate results compared to estimating parameters using data from individual units separately. While hierarchical modeling has been widely applied in various fields, its introduction to ecologists has been relatively recent.

The full N-mixture model is :

```
## N-mixture stan model
carp_BinPois <- "
data {
  int<lower=0> Nobs;
  int<lower=0> Nmax;
  int<lower=0> Nseg;
  int<lower=0,upper=Nseg> segments[Nobs];
  int y[Nobs];
  real alpha;
  real beta;
  real f;
}
parameters {
  real mu;
  vector[Nseg] z;
  real<lower=0,upper=10> sigma;
  real<lower=0,upper=1> pd;
}
transformed parameters {
  vector[Nseg] log_lambda;
  real lambda[Nobs];
  log_lambda = mu+sigma*z;
  for (i in 1:Nobs)
    lambda[i] = f*exp(log_lambda[segments[i]]);
}
model{
  int k;
  z ~ std_normal();
  mu ~ normal(0,5);
  sigma ~ normal(0,2.5);
  pd ~ beta(alpha, beta);
  for (i in 1:Nobs){
    vector[Nmax-y[i]+1] temp;
    for (j in y[i]:Nmax){
      k = j-y[i]+1;
      temp[k] = binomial_lpmf(y[i] | j, pd) +
                poisson_lpmf(j | lambda[i]);
    }
    target += log_sum_exp(temp);
  }
}
"
fit1 <- stan_model(model_code = carp_BinPois)
```

```
## Trying to compile a simple C file
```

```
## Running /usr/lib/R/bin/R CMD SHLIB foo.c
## using C compiler: ‘gcc (Ubuntu 11.3.0-1ubuntu1~22.04.1) 11.3.0’
## gcc -I"/usr/share/R/include" -DNDEBUG   -I"/usr/lib/R/site-library/Rcpp/include/"  -I"/usr/lib/R/site
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:88,
##                  from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/local/lib/R/site-library/StanHeaders/include/stan/math/prim/fun/Eigen.hpp
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:1: error: unknown type nar
##   628 | namespace Eigen {
##       | ^~~~~~~~~
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:17: error: expected ‘=’,
##   628 | namespace Eigen {
##       |                 ^
```

```
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/local/lib/R/site-library/StanHeaders/include/stan/math/prim/fun/Eigen.hpp
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:96:10: fatal error: complex: No such file or di
##    96 | #include <complex>
##       |          ^~~~~~~~~
## compilation terminated.
## make: *** [/usr/lib/R/etc/Makeconf:191: foo.o] Error 1
```

```
## prior parameters of the detection probability (beta(alp, bet))
alp <- 86.20832
bet <- 78.77657
```

The model is very slow.

```
#### N-mixture model--do not run time consuming

stan_in1 <- function(data = gc_data, a = alp, b = bet, f = 0.127/2, chains = nchains,
    Nmax = 25) {
    y <- data$count
    n <- length(y)
    seg <- as.numeric(ordered(data$segment))
    nseg <- max(seg)
    stan_data <- list(Nobs = n, Nmax = Nmax, Nseg = nseg, f = f, segments = seg,
        y = y, alpha = a, beta = b)
    stan_inits <- list()
    for (i in 1:chains) stan_inits[[i]] <- list(z = rnorm(nseg), mu = rnorm(1), sigma = runif(1),
        pd = runif(1))
    stan_pars <- c("log_lambda", "mu", "sigma", "pd")
    return(list(data = stan_data, inits = stan_inits, pars = stan_pars, n.chains = chains))
}

## full data --do not run time consuming
input.to.stan <- stan_in1(Nmax = 25)
## fit2keep <- sampling(fit1, data=input.to.stan$data,
## init=input.to.stan$inits, pars=input.to.stan$pars, iter=niters,thin=nthin,
## chains=input.to.stan$n.chains) control=list(max_treedepth=25))

## save(fit2keep, file='Stan_Nmixture.RData') print(fit2keep)
```

Since the majority of non-zero count values in the data are 1, there is limited information available to estimate the parameter of the Poisson model accurately. To address this issue, a simplified model proposed by Qian et al. (2022) transformed the count variable into a binary variable indicating whether a fish was captured or not. Specifically, for the electrofishing process, the capture of a grass carp can be modeled as a Bernoulli random variable:

$$y_i \sim Bern(p_d\theta)$$

where y_i represents the binary outcome indicating either a capture (1) or no capture (0). The detection probability of electronic fishing, denoted as p_d, represents the probability of detecting a fish if it is present. The parameter θ_i corresponds to the probability of at least one grass carp being present at the sampling site.

The number of grass carp present in the sampling site, N_i, can be modeled as a Poisson random variable with the mean equal to the product of the average fish density λ (measured in the number of fish per kilometer) and the size of the sampling site d (measured in kilometers). Therefore, we have $N_i \sim \text{Pois}(\lambda d)$. From this, we can derive the probability of at least one grass carp being present:

$$\theta = 1 - e^{-\lambda d}$$

The detection probability p_d was estimated based on an independent depletion study.

Using a latent variable method, we can use z_i to represent where at least one fish was present ($z_i = 1$) or not ($z_i = 0$). The likelihood function is the probability of observing a specific y_i. The joint probability of y_i and $z - i$ is

$$\pi(y_i, z_i \mid \lambda, p_d) = \pi(y_i \mid z_i, p_d)\pi(z_i \mid \lambda).$$

Here,

$$\pi(y_i \mid z_i, p_d) = \begin{cases} p_d & \text{if } z_i = 1 \\ 0 & \text{if } z_i = 0 \end{cases}$$

and

$$\pi(z_i \mid \lambda) = \begin{cases} 1 - e^{-\lambda d} & \text{if } z_i = 1 \\ e^{-\lambda d} & \text{if } z_i = 0 \end{cases}.$$

Because the latent variable is binary, the marginalization is

$$\begin{array}{rl} \pi(y_i \mid \lambda, p_d) = & \pi(y_i, z_i = 1 \mid \lambda, p_d) + \pi(y_i, z_i = 0 \mid \lambda, p_d) \\ = & \pi(y_i \mid z_i = 0, p_d)\pi(z_i = 0 \mid \lambda) + \\ & \pi(y_i \mid z_i = 1, p_d)\pi(z_i = 1 \mid \lambda) \\ = & \begin{cases} 0 \times e^{-\lambda d_i} + p_d \times (1 - e^{-\lambda d_i}) & \text{if } y_i = 1 \\ 1 \times e^{-\lambda d_i} + (1 - p_d)(1 - e^{-\lambda d_i}) & \text{if } y_i = 0 \end{cases} \end{array}.$$

```
### Binary formulation--do not run
carp_BernBern <- "
data {
  int<lower=0> Nobs;
  int<lower=0> Nseg;
  int<lower=0,upper=Nseg> segments[Nobs];
  int y[Nobs];
  real a;
  real b;
  real f;
}
parameters {
  vector[Nseg] z;
  real<lower=0,upper=1> pd;
  real mu;
  real<lower=0,upper=10> sigma;
}
transformed parameters{
  vector[Nseg] log_lambda;
  real lambda[Nobs];
  log_lambda = mu + z * sigma;
  for (i in 1:Nobs)
    lambda[i] = f*exp(log_lambda[segments[i]]);
}
model {
  real temp2[2];
  pd ~ beta(a, b);
  z ~ std_normal();
  mu ~ normal(0,5);
  sigma ~ normal(0,2.5);
  for (i in 1:Nobs){
    temp2[1] = -lambda[i];
    temp2[2] = log1m_exp(-lambda[i])+log(1-pd);
    target += y[i]*(log1m_exp(-lambda[i]) + log(pd)) +
              (1-y[i])*log_sum_exp(temp2);
  }
}
"
fit2 <- stan_model(model_code = carp_BernBern)
```

```
## Trying to compile a simple C file

## Running /usr/lib/R/bin/R CMD SHLIB foo.c
## using C compiler: ‘gcc (Ubuntu 11.3.0-1ubuntu1~22.04.1) 11.3.0’
## gcc -I"/usr/share/R/include" -DNDEBUG   -I"/usr/lib/R/site-library/Rcpp/include/"  -I"/usr/lib/R/site
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:88,
##                  from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/local/lib/R/site-library/StanHeaders/include/stan/math/prim/fun/Eigen.hpp
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:1: error: unknown type nam
##   628 | namespace Eigen {
##       | ^~~~~~~~~
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:17: error: expected ‘=’,
##   628 | namespace Eigen {
##       |                 ^
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/local/lib/R/site-library/StanHeaders/include/stan/math/prim/fun/Eigen.hpp
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:96:10: fatal error: complex: No such file or di
##    96 | #include <complex>
##       |          ^~~~~~~~~
## compilation terminated.
## make: *** [/usr/lib/R/etc/Makeconf:191: foo.o] Error 1
```

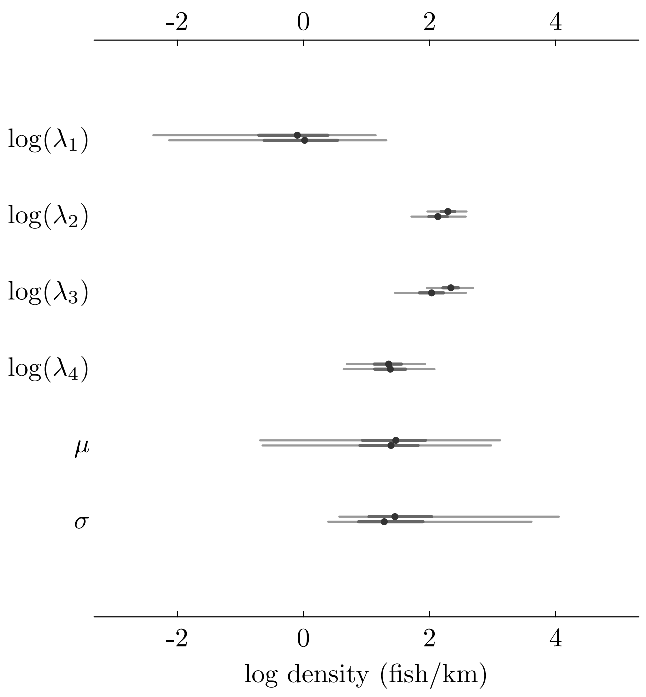
```
stan_in2 <- function(data = gc_data, a = alp, b = bet, f = 0.127/2, chains = nchains) {
    y <- as.numeric(data$count > 0)
    n <- length(y)
    seg <- as.numeric(ordered(data$segment))
    nseg <- max(seg)
    stan_data <- list(Nobs = n, Nseg = nseg, f = f, segments = seg, y = y, a = a,
        b = b)
    stan_inits <- list()
    for (i in 1:chains) stan_inits[[i]] <- list(z = rnorm(nseg), mu = rnorm(1, -2),
        sigma = runif(1))
    stan_pars <- c("log_lambda", "mu", "sigma", "pd")
    return(list(data = stan_data, inits = stan_inits, pars = stan_pars, n.chains = chains))
}

###--do not run
input.to.stan <- stan_in2()
## fit2keepBin <- sampling(fit2, data=input.to.stan$data,
## init=input.to.stan$inits, pars=input.to.stan$pars, iter=niters,thin=nthin,
## chains=input.to.stan$n.chains) control=list(max_treedepth=25))
## print(fit2keepBin)

## save(fit2keep, fit2keepBin, file='sandusky_compare.RData')
load("sandusky_compare.RData")
```

```
## model comparison to justify the use of the simplified model
```

Run-time difference is two orders of magnitude (1500 versus 30).

To justify the use of the simplified model, we made a simple comparison of the comparable model parameters:

```
NM_coef <- rvsims(as.matrix(as.data.frame(rstan::extract(fit2keep, pars = c("log_lambda",
    "mu", "sigma")))))
NMbin_coef <- rvsims(as.matrix(as.data.frame(rstan::extract(fit2keepBin, pars = c("log_lambda",
    "mu", "sigma")))))

Mixture_coef <- cbind(NM_coef, NMbin_coef)

rownames(Mixture_coef) <- c("$\\log(\\lambda_1)$", "$\\log(\\lambda_2)$", "$\\log(\\lambda_3)$",
    "$\\log(\\lambda_4)$", "$\\mu$", "$\\sigma$")

par(mar = c(3, 3, 1, 1), mgp = c(1.25, 0.125, 0), tck = 0.01)
mlplot((Mixture_coef), xlim = c(-3, 5), xlab = "log density (fish/km)", cex = 0.5)
```



The proposed model satisfies Cox's three criteria for an applied statistical model. The response variable is modeled using the Bernoulli distribution, with the parameter p_d representing the effectiveness of the sampling method, and the parameter λ describing the average fish density, which is of primary interest for estimating the fish population. The division of the river segments, although based on data availability rather than ecological considerations, allows for the evaluation of temporal population trends through sequential updating.

The concept of updating upon new data is a fundamental aspect of Bayesian analysis. In this case, we treat the posterior distribution of model parameters at one time as the priors for the next time step. Given the systematic removal pressure on the grass carp population, it is necessary to assume that the population changes every year. When multiple years of data are available, a Bayesian hierarchical model can effectively capture the uncertainty associated with annual population differences.

The hierarchical model is structured as a hyper-distribution of segment-specific densities. Specifically, we assume that the logarithm of the segment means, $\log(\lambda_j)$, follows a normal distribution with mean μ_λ and variance τ^2_λ. The hyper-parameter μ_λ represents the log-mean of segment means, while τ^2_λ represents the among-segment variance. To account for temporal changes in population, the model is extended to incorporate the division of data into segments (j) and years (t):The hierarchical model can be summarized as a hyper-distribution of segment-specific densities. That is:

$$\log(\lambda_{jt}) \sim N(\mu_{\lambda,t}, \tau^2_{\lambda,t})$$

To reflect our uncertainty about how the hyper-parameters vary by year, we introduce a second layer of hierarchical modeling:

$$\left(\mu_{\lambda,t}, \tau^2_{\lambda,t}\right) \sim \text{Normal-Iverse-Gamma}(\mu_0, n_0, \alpha, \beta)$$

Here, we impose a common prior for the annual hyper-parameters $\mu_{\lambda,t}$ and $\tau^2_{\lambda,t}$ using the normal-inverse-gamma distribution, which is a conjugate family of priors. In a Bayesian hierarchical model, vague or flat priors are typically used for the hyper-parameters, and the available data (in this case, multiple years of data) is used to estimate the parameters.

Qian et al. (2022) proposed a sequential updating approach for hierarchical models, eliminating the need to wait for data from multiple years. The estimated hyper-parameters based on the available data are used to estimate the prior parameters (i.e., $\mu_0, n_0, \alpha, \beta$), and with data from each additional year, the Bayesian estimator is applied directly (i.e., $\log(\lambda_{jt}) \sim N(\mu_{\lambda,t}, \nu^2_{\lambda,t})$), updating the hyper-parameters and deriving the prior parameters for the next year.

```
### Sequential updating 1. priors for hyper-parameters

## prior parameter estimation using method of moments be careful of the
## normal-gamma specification (in terms of sig2, not sig) the inputs are MCMC
## samples of mu and sig2
prior_pars_NIG <- function(mus, sigs2) {
    Ex <- mean(mus)
    Vx <- sd(mus)^2
    Esig2 <- mean(sigs2)
    Vsig2 <- sd(sigs2)^2
    return(list(mu0 = Ex, beta = Esig2 * (1 + Esig2^2/Vsig2), alpha = 2 + Esig2^2/Vsig2,
        lambda = Esig2/Vx))
}

prior_pars_IG <- function(sig2) {
    Esig2 <- mean(sig2)
    Vsig2 <- sd(sig2)^2
    return(list(alpha = 2 + Esig2^2/Vsig2, beta = Esig2 * (1 + Esig2^2/Vsig2)))
}
```

In a way, we summarize previous years information in the prior and update them one year at a time. Each time, not only we have the year-specific estimate but also updates the cumulative information.

```
##### Single detection probability
BernBern_seq <- "
data {
  int<lower=0> Nobs;
  int<lower=0> Nseg;
  int<lower=0,upper=Nseg> segments[Nobs];
  int<lower=0,upper=1> y[Nobs];
  real<lower=0> a;
  real<lower=0> b;
  real<lower=0> f[Nobs];
  real<lower=0> hyp_alpha;
  real<lower=0> hyp_beta;
  real hyp_mu;
  real<lower=0> hyp_n0;
}
parameters {
  vector[Nseg] z;
  real<lower=0,upper=1> pd;
  real mu;
  real<lower=0,upper=10> sigma2;
}
transformed parameters{
  vector[Nseg] log_lambda;
  real lambda[Nobs];
  log_lambda = mu + z * sqrt(sigma2);
  for (i in 1:Nobs)
    lambda[i] = f[i]*exp(log_lambda[segments[i]]);
}
model {
  real temp2[2];
  pd ~ beta(a, b);
  z ~ std_normal();
  sigma2 ~ inv_gamma(hyp_alpha,hyp_beta);
  mu ~ normal(hyp_mu,sqrt(sigma2/hyp_n0));
  for (i in 1:Nobs){
    temp2[1] = -lambda[i];
    temp2[2] = log1m_exp(-lambda[i])+log(1-pd);
    target += y[i]*(log1m_exp(-lambda[i]) + log(pd)) +
              (1-y[i])*log_sum_exp(temp2);
  }
}
"
fit3 <- stan_model(model_code = BernBern_seq)
```

```
## Trying to compile a simple C file

## Running /usr/lib/R/bin/R CMD SHLIB foo.c
## using C compiler: ‘gcc (Ubuntu 11.3.0-1ubuntu1~22.04.1) 11.3.0’
## gcc -I"/usr/share/R/include" -DNDEBUG   -I"/usr/lib/R/site-library/Rcpp/include/"  -I"/usr/lib/R/site
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:88,
##                  from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/local/lib/R/site-library/StanHeaders/include/stan/math/prim/fun/Eigen.hpp
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:1: error: unknown type nam
##   628 | namespace Eigen {
##       | ^~~~~~~~~
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:17: error: expected ‘=’,
##   628 | namespace Eigen {
##       |                 ^
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/local/lib/R/site-library/StanHeaders/include/stan/math/prim/fun/Eigen.hpp
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:96:10: fatal error: complex: No such file or di
##    96 | #include <complex>
##       |          ^~~~~~~~~
## compilation terminated.
## make: *** [/usr/lib/R/etc/Makeconf:191: foo.o] Error 1
```

Now let's implement the sequential updating

```r
## organizing input data and initial values
seqinput1 <- function(data = gc_data2, a = alp, b = bet, priors = hyp_prior, chains = nchains,
    Seg = "segment", sub = NULL, f = 0.127/2, varyingF = T) {
    if (!is.null(sub))
        data <- data[sub, ]
    y <- as.numeric(data$count > 0)
    n <- length(y)
    seg <- as.numeric(ordered(data[, Seg]))
    nseg <- max(seg)
    if (varyingF) {
        f <- data$efish_distance/2000
        f[is.na(f)] <- mean(f, na.rm = T)
    } else {
        f <- rep(f, n)
    }
    stan_data <- list(Nobs = n, Nseg = nseg, f = f, segments = seg, y = y, a = a,
        b = b, hyp_mu = priors$mu0, hyp_n0 = priors$lambda, hyp_alpha = priors$alpha,
        hyp_beta = priors$beta)
    stan_inits <- list()
    for (i in 1:chains) stan_inits[[i]] <- list(z = rnorm(nseg), mu = rnorm(1, -2),
        sigma2 = runif(1))
    stan_pars <- c("log_lambda", "mu", "sigma2", "pd")
    return(list(data = stan_data, inits = stan_inits, pars = stan_pars, n.chains = chains))
}
```

Sequential updating:

```r
### 2020 by Segment
hyppars <- rstan::extract(fit2keepBin, pars = c("mu", "sigma"))
hyp_prior <- prior_pars_NIG(mus = hyppars$mu, sigs2 = hyppars$sigma^2)
input.to.stan <- seqinput1(data = gc_data2, sub = gc_data2$year == 2020)
fit2keepSeg_2020 <- sampling(fit3, data = input.to.stan$data, init = input.to.stan$inits,
    pars = input.to.stan$pars, iter = niters, thin = nthin, chains = input.to.stan$n.chains)
## control=list(max_treedepth=25))
print(fit2keepSeg_2020)
```

```
## Inference for Stan model: anon_model.
## 8 chains, each with iter=5000; warmup=2500; thin=8;
## post-warmup draws per chain=313, total post-warmup draws=2504.
##
##                  mean se_mean   sd    2.5%     25%     50%     75%   97.5%
## log_lambda[1]   -0.84    0.02 0.73   -2.44   -1.27   -0.76   -0.34    0.38
## log_lambda[2]    1.30    0.01 0.38    0.56    1.05    1.29    1.54    2.06
## log_lambda[3]    1.29    0.01 0.30    0.72    1.09    1.29    1.48    1.90
## log_lambda[4]    0.96    0.01 0.43    0.11    0.68    0.96    1.24    1.82
## mu               1.00    0.01 0.52   -0.05    0.69    1.00    1.32    2.00
## sigma2           1.97    0.02 1.25    0.68    1.17    1.63    2.37    5.34
## pd               0.51    0.00 0.04    0.44    0.48    0.51    0.54    0.59
## lp__          -227.56    0.04 1.99 -232.51 -228.61 -227.21 -226.11 -224.76
##               n_eff Rhat
## log_lambda[1]  2318    1
## log_lambda[2]  2468    1
## log_lambda[3]  2470    1
## log_lambda[4]  2531    1
## mu              2360    1
## sigma2          2681    1
## pd              2419    1
## lp__            2676    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 30 12:47:18 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
### 2021 by Segment
hyppars <- rstan::extract(fit2keepSeg_2020, pars = c("mu", "sigma2"))
hyp_prior <- prior_pars_NIG(mus = hyppars$mu, sigs2 = hyppars$sigma2)
input.to.stan <- seqinput1(data = gc_data2, sub = gc_data2$year == 2021)
fit2keepSeg_2021 <- sampling(fit3, data = input.to.stan$data, init = input.to.stan$inits,
    pars = input.to.stan$pars, iter = niters, thin = nthin, chains = input.to.stan$n.chains)
## control=list(max_treedepth=25))
print(fit2keepSeg_2021)
```

```
## Inference for Stan model: anon_model.
## 8 chains, each with iter=5000; warmup=2500; thin=8;
## post-warmup draws per chain=313, total post-warmup draws=2504.
##
##                  mean se_mean   sd    2.5%     25%     50%     75%   97.5%
## log_lambda[1]   -0.87    0.01 0.59   -2.20   -1.23   -0.82   -0.44    0.17
## log_lambda[2]   -0.57    0.01 0.41   -1.41   -0.84   -0.55   -0.29    0.15
## log_lambda[3]    0.03    0.01 0.27   -0.54   -0.15    0.04    0.21    0.52
## log_lambda[4]    0.17    0.01 0.33   -0.51   -0.04    0.17    0.40    0.80
## mu               0.54    0.01 0.40   -0.26    0.27    0.54    0.79    1.31
## sigma2           1.77    0.02 0.84    0.75    1.19    1.58    2.11    4.00
## pd               0.51    0.00 0.04    0.43    0.49    0.51    0.54    0.59
## lp__          -247.67    0.04 1.87 -252.07 -248.72 -247.39 -246.30 -244.96
##               n_eff Rhat
## log_lambda[1]  2061    1
## log_lambda[2]  2558    1
## log_lambda[3]  2229    1
## log_lambda[4]  2625    1
## mu             2387    1
## sigma2         2452    1
## pd             2374    1
## lp__           2537    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 30 12:47:51 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
### 2022 by Segment
hyppars <- rstan::extract(fit2keepSeg_2021, pars = c("mu", "sigma2"))
hyp_prior <- prior_pars_NIG(mus = hyppars$mu, sigs2 = hyppars$sigma2)
input.to.stan <- seqinput1(data = gc_data4)
fit2keepSeg_2022 <- sampling(fit3, data = input.to.stan$data, init = input.to.stan$inits,
    pars = input.to.stan$pars, iter = niters, thin = nthin, chains = input.to.stan$n.chains)
## control=list(max_treedepth=25))
print(fit2keepSeg_2022)
```

```
## Inference for Stan model: anon_model.
## 8 chains, each with iter=5000; warmup=2500; thin=8;
## post-warmup draws per chain=313, total post-warmup draws=2504.
##
##                    mean se_mean   sd    2.5%     25%     50%     75%   97.5%
## log_lambda[1]   -0.71    0.01 0.55   -1.88   -1.05   -0.68   -0.33    0.28
## log_lambda[2]   -0.67    0.01 0.45   -1.65   -0.95   -0.64   -0.35    0.12
## log_lambda[3]   -1.34    0.01 0.64   -2.79   -1.73   -1.28   -0.90   -0.25
## log_lambda[4]   -1.24    0.01 0.74   -2.87   -1.68   -1.18   -0.72    0.03
## mu               0.14    0.01 0.36   -0.56   -0.10    0.14    0.38    0.83
## sigma2           1.87    0.02 0.78    0.87    1.33    1.71    2.22    3.91
## pd               0.51    0.00 0.04    0.43    0.48    0.51    0.54    0.58
## lp__          -176.20    0.04 1.93 -180.90 -177.23 -175.82 -174.83 -173.44
##               n_eff Rhat
## log_lambda[1]  2575    1
## log_lambda[2]  2395    1
## log_lambda[3]  2544    1
## log_lambda[4]  2509    1
## mu             2508    1
## sigma2         2554    1
## pd             2411    1
## lp__           2472    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 30 12:48:08 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

We can change segment to season to study the seasonal pattern of population sizes as a way to understand the fish's movement within a year. All we need to do is to enter `Seg="Season"` when running the function `seqinput1`.

Instead of running the model separately by segments and by season, we can also make the model more complicated by including season as a nested factor under segments. This means that each of the 4 segment means are split into three seasonal components, leading to 12 segment-season means. Let j be the index of segments and s be the index of season. The 12 means can be denoted as $\log(\lambda_{js})$. The nested model can be expressed as:

$$\log(\lambda_{js}) \sim N(\mu_j, \sigma_2^2)$$

where, μ_j is the segment mean and σ_2^2 is the among season variance. The segment means are than modeled as random variables from the hyper distribution $\mu_j \sim N(\mu_{hyp}, \tau_{hyp}^2)$. I used two indicators in the Stan model to make the program more efficient.

```
BernBern_seq2 <- "
data {
  int<lower=0> Nobs;
  int<lower=1> Nseg;
  int<lower=1> Nsn;
  int<lower=1> Nsegsn;
  int<lower=1,upper=Nsegsn> indij[Nobs];
  int<lower=1,upper=Nseg> indj[Nsegsn];
  int<lower=0,upper=1> y[Nobs];
  real<lower=0> a;
  real<lower=0> b;
  real<lower=0> f[Nobs];
  real<lower=0> hyp_alpha;
  real<lower=0> hyp_beta;
  real hyp_mu;
  real<lower=0> hyp_n0;
}
parameters {
  vector[Nsegsn] zij;
  vector[Nseg] zi;
  real<lower=0,upper=1> pd;
  real m;
  real<lower=0,upper=10> sigma2;
  real<lower=0,upper=10> tau2;
}
transformed parameters{
  vector[Nsegsn] log_lambda;
  real lambda[Nobs];
  vector[Nseg] mu;
  mu = m+zi*sqrt(tau2);
  for (i in 1:Nsegsn)
    log_lambda[i] = mu[indj[i]] + zij[i] * sqrt(sigma2);
  for (i in 1:Nobs)
    lambda[i] = f[i]*exp(log_lambda[indij[i]]);
}
model {
  real temp2[2];
  pd ~ beta(a, b);
  zij ~ std_normal();
  zi ~ std_normal();
  tau2 ~ inv_gamma(hyp_alpha,hyp_beta);
  m ~ normal(hyp_mu, sqrt(tau2/hyp_n0));
  for (i in 1:Nobs){
    temp2[1] = -lambda[i];
    temp2[2] = log1m_exp(-lambda[i])+log(1-pd);
    target += y[i]*(log1m_exp(-lambda[i]) + log(pd)) +
              (1-y[i])*log_sum_exp(temp2);
  }
}
"

fit4 <- stan_model(model_code = BernBern_seq2)
```

```
## Trying to compile a simple C file

## Running /usr/lib/R/bin/R CMD SHLIB foo.c
## using C compiler: ‘gcc (Ubuntu 11.3.0-1ubuntu1~22.04.1) 11.3.0’
## gcc -I"/usr/share/R/include" -DNDEBUG   -I"/usr/lib/R/site-library/Rcpp/include/"  -I"/usr/lib/R/sit
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:88,
##                  from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/local/lib/R/site-library/StanHeaders/include/stan/math/prim/fun/Eigen.hpp
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:1: error: unknown type na
##   628 | namespace Eigen {
##       | ^~~~~~~~~
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:17: error: expected ‘=’,
##   628 | namespace Eigen {
```

```
##       |                  ^
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/local/lib/R/site-library/StanHeaders/include/stan/math/prim/fun/Eigen.hpp
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:96:10: fatal error: complex: No such file or di
##    96 | #include <complex>
##       |          ^~~~~~~~~
## compilation terminated.
## make: *** [/usr/lib/R/etc/Makeconf:191: foo.o] Error 1
```

One is indij, matching each observation to the 12 segment-season combinations, and the other is indj matching the 12 segment-season combinations to segments. With the index, I can define $\log(\lambda_{js})$ as a (one-dimension) vector instead of a (two-dimension) matrix. Some care must be taken in organizing input data:

```r
seqinput2 <- function(data = gc_data4, a = alp, b = bet, priors = hyp_prior, chains = nchains,
    sub = NULL) {
    if (!is.null(sub))
        data <- data[sub, ]
    y <- as.numeric(data$count > 0)
    n <- length(y)
    segCh <- data$segment
    seg <- as.numeric(ordered(segCh))
    snCh <- data$Season
    sn <- as.numeric(ordered(snCh))
    temp <- paste(segCh, snCh)
    indij1 <- as.numeric(ordered(temp))
    indijtemp <- sort(unique(temp))
    indij2 <- as.numeric(ordered(indijtemp))
    indj <- as.numeric(ordered(substring(indijtemp, 1, 1)))
    nsegsn <- max(indij2)
    nseg <- max(seg)
    nsn <- max(sn)
    f <- data$efish_distance
    f[is.na(f)] <- mean(f, na.rm = T)
    stan_data <- list(Nobs = n, Nseg = nseg, Nsn = nsn, Nsegsn = nsegsn, indij = indij1,
        indj = indj, y = y, a = a, b = b, f = f/2000, hyp_mu = priors$mu0, hyp_n0 = priors$lambda,
        hyp_alpha = priors$alpha, hyp_beta = priors$beta)
    stan_inits <- list()
    for (i in 1:chains) stan_inits[[i]] <- list(zij = rnorm(nsegsn), zi = rnorm(nseg),
        pd = runif(1), m = rnorm(1, -2), tau2 = runif(1), sigma2 = runif(1))
    stan_pars <- c("log_lambda", "mu", "m", "sigma2", "tau2", "pd")
    return(list(data = stan_data, inits = stan_inits, pars = stan_pars, n.chains = chains))
}

### 2020 nested
hyppars <- rstan::extract(fit2keepBin, pars = c("mu", "sigma"))
hyp_prior <- prior_pars_NIG(mus = hyppars$mu, sigs2 = hyppars$sigma^2)
input.to.stan <- seqinput2(data = gc_data2, sub = gc_data2$year == 2020)
fit2keepNested2020 <- sampling(fit4, data = input.to.stan$data, init = input.to.stan$inits,
    pars = input.to.stan$pars, iter = niters, thin = nthin, chains = input.to.stan$n.chains)
## control=list(max_treedepth=25))
print(fit2keepNested2020)
```

```
## Inference for Stan model: anon_model.
```

```
## 8 chains, each with iter=5000; warmup=2500; thin=8;
## post-warmup draws per chain=313, total post-warmup draws=2504.
##
##                    mean se_mean   sd    2.5%     25%     50%     75%   97.5%
## log_lambda[1]     -1.10    0.03 1.32   -4.18   -1.77   -0.98   -0.23    1.11
## log_lambda[2]     -0.87    0.02 0.91   -2.98   -1.40   -0.79   -0.26    0.68
## log_lambda[3]     -1.38    0.03 1.24   -4.31   -2.00   -1.21   -0.54    0.51
## log_lambda[4]      0.82    0.03 1.32   -2.14    0.20    0.99    1.59    3.06
## log_lambda[5]      1.37    0.02 0.71    0.13    0.93    1.33    1.73    2.91
## log_lambda[6]      1.42    0.01 0.57    0.52    1.07    1.39    1.69    2.50
## log_lambda[7]      1.04    0.03 1.42   -2.22    0.43    1.15    1.73    3.84
## log_lambda[8]      1.30    0.01 0.39    0.57    1.03    1.30    1.57    2.08
## log_lambda[9]      1.33    0.01 0.49    0.46    1.02    1.32    1.62    2.28
## log_lambda[10]     0.49    0.03 1.35   -2.77   -0.08    0.64    1.23    2.91
## log_lambda[11]     0.41    0.01 0.68   -1.14    0.00    0.47    0.88    1.60
## log_lambda[12]     1.64    0.02 0.88    0.44    1.09    1.50    1.99    3.92
## mu[1]             -0.68    0.02 0.97   -2.74   -1.29   -0.64   -0.05    1.17
## mu[2]              1.19    0.01 0.72   -0.33    0.77    1.20    1.63    2.61
## mu[3]              1.20    0.01 0.71   -0.30    0.79    1.22    1.60    2.63
## mu[4]              0.87    0.01 0.75   -0.66    0.43    0.87    1.31    2.41
## m                  1.00    0.01 0.55   -0.11    0.65    1.00    1.35    2.08
## sigma2             1.50    0.04 1.87    0.02    0.28    0.77    1.93    7.19
## tau2               2.06    0.03 1.34    0.68    1.18    1.70    2.48    5.76
## pd                 0.50    0.00 0.04    0.42    0.48    0.50    0.53    0.59
## lp__            -233.06    0.08 3.72 -241.27 -235.36 -232.63 -230.44 -226.82
##                 n_eff Rhat
## log_lambda[1]    2609    1
## log_lambda[2]    2348    1
## log_lambda[3]    2443    1
## log_lambda[4]    2420    1
## log_lambda[5]    2240    1
## log_lambda[6]    2471    1
## log_lambda[7]    2387    1
## log_lambda[8]    2267    1
## log_lambda[9]    2395    1
## log_lambda[10]   2483    1
## log_lambda[11]   2555    1
## log_lambda[12]   2581    1
## mu[1]            2295    1
## mu[2]            2404    1
## mu[3]            2511    1
## mu[4]            2605    1
## m                2624    1
## sigma2           2311    1
## tau2             2411    1
## pd               2363    1
## lp__             2107    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 30 12:49:31 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```r
### 2021 nested
hyppars <- rstan::extract(fit2keepNested2020, pars = c("mu", "sigma2"))
hyp_prior <- prior_pars_NIG(mus = hyppars$mu, sigs2 = hyppars$sigma2)
input.to.stan <- seqinput2(data = gc_data2, sub = gc_data2$year == 2021)
fit2keepNested2021 <- sampling(fit4, data = input.to.stan$data, init = input.to.stan$inits,
    pars = input.to.stan$pars, iter = niters, thin = nthin, chains = input.to.stan$n.chains)
## control=list(max_treedepth=25))
print(fit2keepNested2021)
```

```
## Inference for Stan model: anon_model.
## 8 chains, each with iter=5000; warmup=2500; thin=8;
## post-warmup draws per chain=313, total post-warmup draws=2504.
##
##                    mean se_mean   sd    2.5%     25%     50%     75%   97.5%
## log_lambda[1]     -1.81    0.03 1.30   -4.82   -2.52   -1.64   -0.90    0.22
## log_lambda[2]     -1.88    0.03 1.27   -4.79   -2.55   -1.72   -1.04    0.19
## log_lambda[3]     -0.54    0.01 0.74   -2.11   -1.03   -0.50   -0.04    0.81
## log_lambda[4]     -0.49    0.01 0.73   -2.09   -0.94   -0.43    0.03    0.76
## log_lambda[5]     -2.03    0.02 1.19   -5.03   -2.66   -1.85   -1.20   -0.26
## log_lambda[6]     -0.28    0.01 0.55   -1.47   -0.62   -0.25    0.09    0.74
## log_lambda[7]      0.94    0.01 0.45    0.09    0.64    0.93    1.22    1.84
## log_lambda[8]     -0.84    0.01 0.56   -2.07   -1.18   -0.78   -0.46    0.13
## log_lambda[9]     -0.33    0.01 0.51   -1.42   -0.65   -0.29    0.03    0.58
## log_lambda[10]     0.92    0.01 0.59   -0.13    0.54    0.90    1.23    2.02
## log_lambda[11]    -0.94    0.02 0.84   -2.76   -1.42   -0.83   -0.36    0.42
## log_lambda[12]    -0.17    0.01 0.58   -1.36   -0.55   -0.12    0.24    0.86
## mu[1]             -0.90    0.02 0.84   -2.74   -1.40   -0.87   -0.34    0.64
## mu[2]             -0.62    0.01 0.74   -2.18   -1.08   -0.58   -0.15    0.75
## mu[3]             -0.10    0.01 0.68   -1.46   -0.52   -0.09    0.32    1.24
## mu[4]             -0.09    0.02 0.70   -1.57   -0.53   -0.07    0.37    1.25
## m                 -0.18    0.01 0.56   -1.36   -0.53   -0.17    0.19    0.87
## sigma2             2.15    0.04 1.81    0.27    0.90    1.60    2.71    7.44
## tau2               1.11    0.02 0.75    0.36    0.65    0.90    1.30    3.10
## pd                 0.51    0.00 0.04    0.43    0.49    0.51    0.54    0.59
## lp__            -240.56    0.08 3.85 -249.12 -242.97 -240.24 -237.82 -234.14
##                n_eff Rhat
## log_lambda[1]   2465    1
## log_lambda[2]   2409    1
## log_lambda[3]   2506    1
## log_lambda[4]   2490    1
## log_lambda[5]   2372    1
## log_lambda[6]   2579    1
## log_lambda[7]   2553    1
## log_lambda[8]   2413    1
## log_lambda[9]   2494    1
## log_lambda[10]  2140    1
## log_lambda[11]  2368    1
## log_lambda[12]  2449    1
## mu[1]           2587    1
## mu[2]           2558    1
## mu[3]           2682    1
## mu[4]           2120    1
## m               2470    1
## sigma2          2355    1
## tau2            2449    1
## pd              2401    1
## lp__            2354    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 30 12:50:29 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
### 2022 nested
hyppars <- rstan::extract(fit2keepNested2021, pars = c("mu", "sigma2"))
hyp_prior <- prior_pars_NIG(mus = hyppars$mu, sigs2 = hyppars$sigma2)
input.to.stan <- seqinput2(data = gc_data4)
fit2keepNested2022 <- sampling(fit4, data = input.to.stan$data, init = input.to.stan$inits,
    pars = input.to.stan$pars, iter = niters, thin = nthin, chains = input.to.stan$n.chains)
## control=list(max_treedepth=25))
print(fit2keepNested2022)
```

```
## Inference for Stan model: anon_model.
## 8 chains, each with iter=5000; warmup=2500; thin=8;
## post-warmup draws per chain=313, total post-warmup draws=2504.
##
##                    mean se_mean   sd    2.5%     25%     50%     75%   97.5%
## log_lambda[1]     -0.40    0.02 0.80   -2.08   -0.88   -0.39    0.14    1.12
## log_lambda[2]     -1.78    0.02 1.16   -4.64   -2.35   -1.59   -0.98   -0.11
## log_lambda[3]     -0.70    0.02 0.90   -2.53   -1.23   -0.66   -0.10    0.89
## log_lambda[4]     -0.60    0.01 0.71   -2.07   -1.04   -0.59   -0.12    0.69
## log_lambda[5]     -1.00    0.01 0.64   -2.42   -1.39   -0.96   -0.55    0.09
## log_lambda[6]     -1.02    0.02 0.80   -2.74   -1.48   -0.96   -0.49    0.40
## log_lambda[7]     -1.36    0.02 0.87   -3.25   -1.88   -1.30   -0.77    0.17
## log_lambda[8]     -1.82    0.02 0.84   -3.70   -2.31   -1.74   -1.23   -0.42
## log_lambda[9]     -1.86    0.02 1.18   -4.57   -2.51   -1.71   -1.06    0.09
## log_lambda[10]    -1.91    0.02 1.25   -4.87   -2.59   -1.77   -1.06    0.12
## log_lambda[11]    -1.49    0.02 0.90   -3.49   -2.03   -1.42   -0.85    0.02
## log_lambda[12]    -1.77    0.03 1.30   -4.81   -2.50   -1.63   -0.91    0.46
## mu[1]             -0.92    0.01 0.76   -2.57   -1.35   -0.88   -0.43    0.44
## mu[2]             -0.87    0.01 0.68   -2.27   -1.29   -0.83   -0.44    0.46
## mu[3]             -1.49    0.02 0.79   -3.15   -1.99   -1.46   -0.96   -0.05
## mu[4]             -1.53    0.02 0.88   -3.41   -2.05   -1.48   -0.94    0.11
## m                 -0.87    0.01 0.53   -1.96   -1.18   -0.85   -0.52    0.12
## sigma2             1.35    0.03 1.58    0.03    0.31    0.78    1.76    5.93
## tau2               1.56    0.02 0.86    0.58    0.98    1.34    1.88    3.87
## pd                 0.52    0.00 0.04    0.44    0.49    0.52    0.54    0.59
## lp__            -175.82    0.08 3.69 -184.04 -178.08 -175.53 -173.17 -169.62
##                n_eff Rhat
## log_lambda[1]   2604    1
## log_lambda[2]   2713    1
## log_lambda[3]   2461    1
## log_lambda[4]   2514    1
## log_lambda[5]   2680    1
## log_lambda[6]   2327    1
## log_lambda[7]   2589    1
## log_lambda[8]   2462    1
## log_lambda[9]   2373    1
## log_lambda[10]  2525    1
## log_lambda[11]  2467    1
## log_lambda[12]  2569    1
## mu[1]           2611    1
## mu[2]           2457    1
## mu[3]           2180    1
## mu[4]           2499    1
## m               2528    1
## sigma2          2565    1
## tau2            2363    1
## pd              2557    1
## lp__            2297    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 30 12:51:00 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

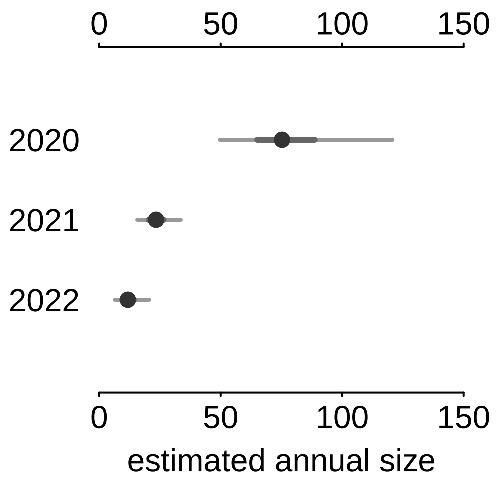
Running the model is not the most challenging task, communicating the results to ecologists is. I tried some plots but am not entirely happy with them.

```
load("seqUpdate2020_2022.RData")
coef2020seg <- extract(fit2keepSeg_2020, pars = c("log_lambda", "mu"))
segCoefrv20 <- rvsims(as.matrix(as.data.frame(coef2020seg)))
coef2020sn <- extract(fit2keepSn_2020, pars = c("log_lambda", "mu"))
snCoefrv20 <- rvsims(as.matrix(as.data.frame(coef2020sn)))
coef2020nested <- extract(fit2keepNested2020, pars = c("log_lambda", "mu", "m"))
nestCoefrv20 <- rvsims(as.matrix(as.data.frame(coef2020nested)))
coef2021seg <- extract(fit2keepSeg_2021, pars = c("log_lambda", "mu"))
segCoefrv21 <- rvsims(as.matrix(as.data.frame(coef2021seg)))
coef2021sn <- extract(fit2keepSn_2021, pars = c("log_lambda", "mu"))
snCoefrv21 <- rvsims(as.matrix(as.data.frame(coef2021sn)))
coef2021nested <- extract(fit2keepNested2021, pars = c("log_lambda", "mu", "m"))
nestCoefrv21 <- rvsims(as.matrix(as.data.frame(coef2021nested)))
coef2022seg <- extract(fit2keepSeg_2022, pars = c("log_lambda", "mu"))
segCoefrv22 <- rvsims(as.matrix(as.data.frame(coef2022seg)))
coef2022sn <- extract(fit2keepSn_2022, pars = c("log_lambda", "mu"))
snCoefrv22 <- rvsims(as.matrix(as.data.frame(coef2022sn)))
coef2022nested <- extract(fit2keepNested2022, pars = c("log_lambda", "mu", "m"))
nestCoefrv22 <- rvsims(as.matrix(as.data.frame(coef2022nested)))

logLambdaSeg <- rvmatrix(c(segCoefrv20[1:4], segCoefrv21[1:4], segCoefrv22[1:4]),
    ncol = 3)
musSeg <- c(segCoefrv20[5], segCoefrv21[5], segCoefrv22[5])
tikz(file = "segbyyear.tex", height = 5, width = 4.5, standAlone = F)
par(mar = c(3, 1, 1, 2), mgp = c(1.25, 0.125, 0), tck = -0.015)
mlplot(exp(logLambdaSeg), xlab = "Segments by Year", col = 1:3)
dev.off()
```

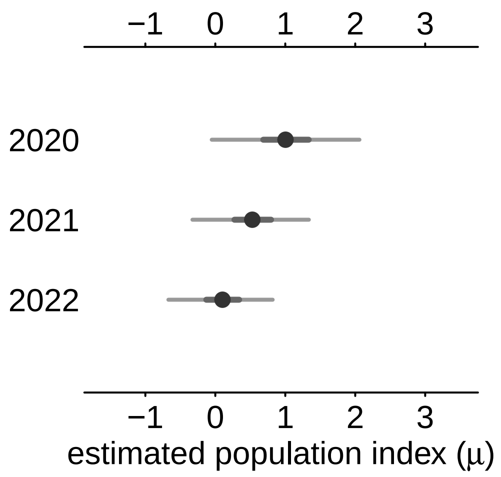
```
## pdf
##   2
```

```
names(musSeg) <- c("2020", "2021", "2022")
## abline(v=summary(musSeg)$mean, lty=1:3) legend(x='topleft',lty=1:3,
## legend=c('2020','2020-2021','2020-2022'), bty='n')

### calculating number of fish
```

```
seg_size <- c(6.44, 6.44, 8.05, 6.44)
seg_num <- exp(logLambdaSeg[, ]) * seg_size

seg_num20 <- simapply(seg_num[, 1], sum)
seg_num21 <- simapply(seg_num[, 2], sum)
seg_num22 <- simapply(seg_num[, 3], sum)

seg_numYr <- c(seg_num20, seg_num21, seg_num22)
names(seg_numYr) <- c("2020", "2021", "2022")
par(mar = c(3, 2, 1, 1), mgp = c(1.25, 0.125, 0), tck = -0.01)
mlplot(seg_numYr, xlab = "estimated annual size", xlim = c(0, 150))
```

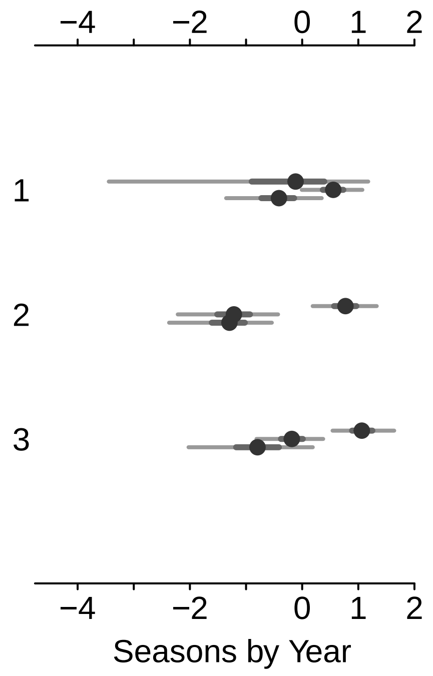


```
par(mar = c(3, 2, 1, 1), mgp = c(1.25, 0.125, 0), tck = -0.01)
mlplot(musSeg, xlab = expression(paste("estimated population index (", mu, ")", sep = "")))
```



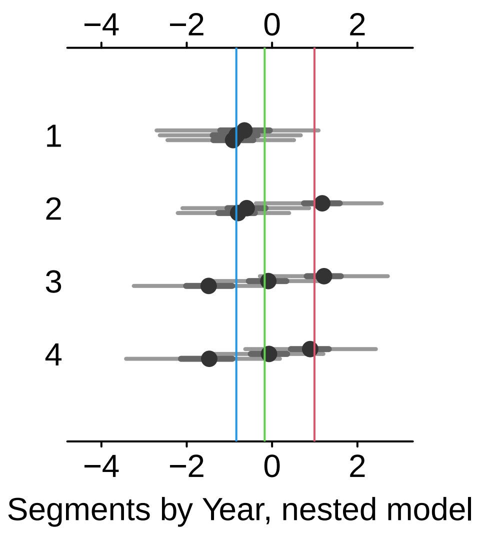
```
### end of annual numbers
```

```
logLambdaSn <- rvmatrix(c(snCoefrv20[1:3], snCoefrv21[1:3], snCoefrv22[1:3]), ncol=3)
musSn <- c(snCoefrv20[4], snCoefrv21[4], snCoefrv22[4])
par(mar=c(3, 1, 1, 2), mgp=c(1.25,0.125,0), tck=-0.015)
mlplot(logLambdaSn, xlab="Seasons by Year", xlim=c(-4.5, 2))
```

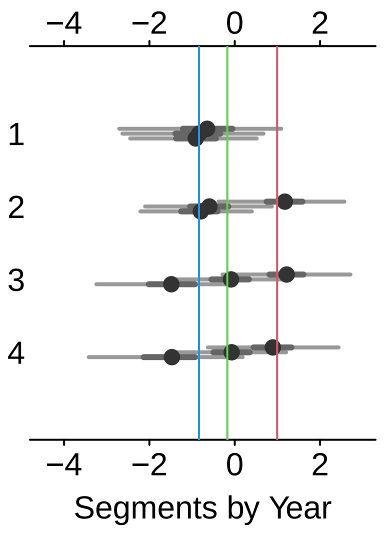


```
##abline(v=summary(musSn)$mean, lty=1:3)
##legend(x="topleft",lty=1:3, legend=c("2020","2020-2021","2020-2022"), bty="n")
```

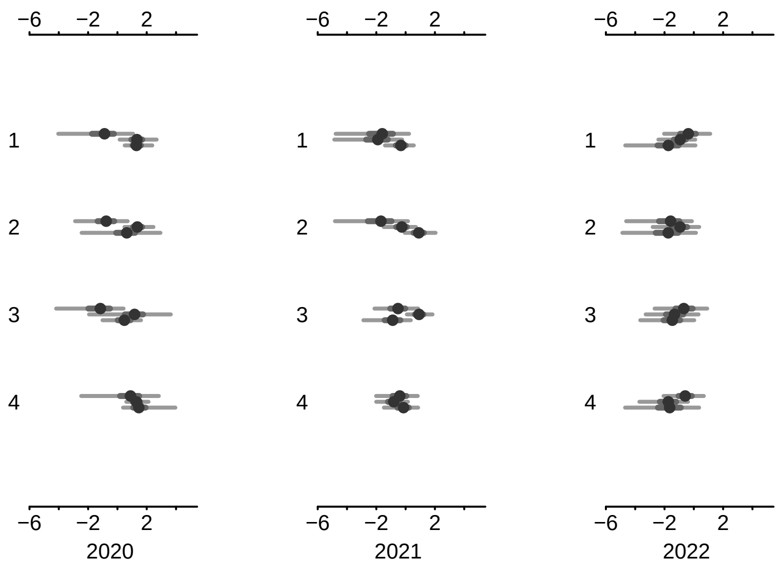
```
logLambdaNst  <- list(rvmatrix(nestCoefrv20[1:12], ncol=3),
                      rvmatrix(nestCoefrv21[1:12], ncol=3),
                      rvmatrix(nestCoefrv22[1:12], ncol=3))
musNst <- rvmatrix(c(nestCoefrv20[13:16], nestCoefrv21[13:16], nestCoefrv22[13:16]), ncol=3)
mNst <- c(nestCoefrv20[17], nestCoefrv21[17], nestCoefrv22[17])

par(mar=c(3, 1, 1, 2), mgp=c(1.25,0.125,0), tck=-0.015)
mlplot(musNst, xlab="Segments by Year, nested model", xlim=c(-4.5, 3))
##abline(v=summary(mNst)$mean, col=2:4)

par(mar=c(3, 1, 1, 2), mgp=c(1.25,0.125,0), tck=-0.015)
mlplot(musNst, xlab="Segments by Year, nested model", xlim=c(-4.5, 3))
abline(v=summary(mNst)$mean, col=2:4)
```



```r
par(mar=c(3, 1, 1, 2), mgp=c(1.25,0.125,0), tck=-0.015)
mlplot(musNst, xlab="Segments by Year", xlim=c(-4.5, 3))
abline(v=summary(mNst)$mean, col=2:4)
```



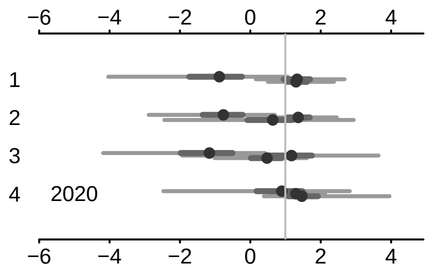
```r
par(mfrow=c(1,3), mar=c(3, 1, 1, 0.5), mgp=c(1.25,0.125,0), tck=-0.015)
mlplot(logLambdaNst[[1]], xlab="2020", xlim=c(-6,5))
##abline(v=summary(musNst[,1])$mean, col=2:5)
mlplot(logLambdaNst[[2]], xlab="2021", xlim=c(-6,5))
##abline(v=summary(musNst[,2])$mean, col=2:5)
mlplot(logLambdaNst[[3]], xlab="2022", xlim=c(-6,5))
```

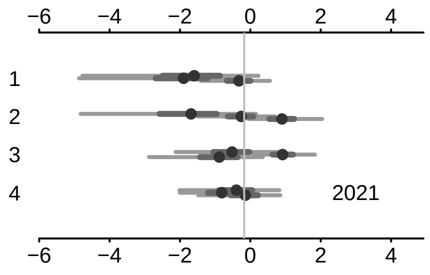


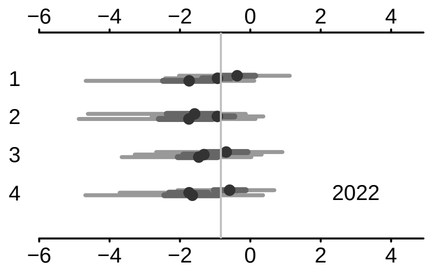
```r
##abline(v=summary(musNst[,3])$mean, col=2:5)
```

```r
par(mfrow=c(3,1), mar=c(1.5, 2, 1, 2), mgp=c(1.25,0.15,0), tck=-0.015)
mlplot(logLambdaNst[[1]], xlab="", xlim=c(-6,4.5))
abline(v=summary(mNst[1])$mean, col="gray")
text(x=-5,y=4, "2020")
mlplot(logLambdaNst[[2]], xlab="", xlim=c(-6,4.5))
abline(v=summary(mNst[2])$mean, col="gray")
```

```
text(x=3,y=4, "2021")
mlplot(logLambdaNst[[3]], xlab="", xlim=c(-6,4.5))
abline(v=summary(mNst[3])$mean, col="gray")
text(x=3,y=4, "2022")
```


−6
−4
−2
0
2
4
1
2
3
4
2020


−6
−4
−2
0
2
4
1
2
3
4
2021


−6
−4
−2
0
2
4
1
2
3
4
2022