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

Song S. Qian

6/3/2023

Introductory Examples

In this session, we will learn about the use of Bayesian inference with examples commonly encountered in ecological and environmental literature. These examples typically involve response variable distributions that are standard and covered in statistical textbooks. We will emphasize the flexibility of the Bayesian approach and explore how these models can be expanded to increase their realism compared to the data generation process.

The guiding principle for Bayesian applications in environmental and ecological statistics should adhere to the three criteria of an applied statistical model recommended by Cox (1995):

1. A probability distribution model for the response variable.
2. A parameter vector that defines the distribution model, where some parameters reflect features of the system under study.
3. The inclusion or representation of the data-generating process.

Of these criteria, the last one, the data-generating process, holds significant importance. Our aim is to develop models that are relevant to the problem at hand and reflect likely causal relationships rather than mere correlations. Furthermore, environmental and ecological data are often observational, meaning we collect the data without prior knowledge of the appropriate model or method for analysis. The statistics we learned in graduate school typically assume that we already know the correct model. We teach statistics chapter by chapter, assuming that we have the right model in mind. However, when working with data from environmental and ecological monitoring programs, we often start collecting data without a clear idea of the questions we want to answer. For instance, most of the data used in studying the effects of climate change are collected without specific hypotheses in mind. In my opinion, analyzing environmental/ecological data often begins with identifying the problem to be addressed. Since all models are inherently wrong (yet some are useful), our focus lies in determining how and when a flawed model can still be useful.

I have found the following process to be effective in analyzing environmental/ecological data:

1. Start with existing (simple) models that can be readily implemented in R.
2. Explore the model fit and identify its weaknesses using the concept of "posterior simulation." This involves using the fitted model to predict what we want to learn and compare it with what we already know.
3. Based on the identified weaknesses, reformulate the model to address those shortcomings.
4. Implement the model in Stan to further assess problems such as computational stability and identifiability (e.g., the snake fungal disease example with unknown error rates).
5. Repeat the posterior simulation to determine where and when the model is useful.

By following this iterative process, we can refine our models, understand their limitations, and identify their utility in analyzing environmental and ecological data.

Example 1 – The Effect of Gulf of Mexico Hypoxia on Benthic Communities

Section 4.2.2, Qian et al (2022).

```
benthic.data <- read.csv(paste(dataDIR, "BenthicData.csv", sep="/"),
                         header=T)
benthic.data$area2 <-
    ordered(benthic.data$area2,
            levels=levels(ordered(benthic.data$area2))[c(2,1,3)])
benthic.data$Area <-
    ordered(benthic.data$Area,
            levels=levels(ordered(benthic.data$Area))[c(2,1,3)])
head(benthic.data)
```

```
##   Area   area2 Station Replicates Richness Abundance log.abund. Depth
## 1    H hypoxic       2          1       13        68       1.84  6.10
## 2    H hypoxic       2          2       19       186       2.27  6.10
## 3    H hypoxic       2          3       15       132       2.12  6.10
## 4    H hypoxic      16          1       10        37       1.58 21.13
## 5    H hypoxic      16          2       15        88       1.95 21.13
## 6    H hypoxic      16          3       22       107       2.03 21.13
```

```
names(benthic.data)
```

```
## [1] "Area"       "area2"      "Station"    "Replicates" "Richness"
## [6] "Abundance"  "log.abund." "Depth"
```

```
## Station: core,
```

Initial analysis (ANOVA) was presented in Baustian et al (2009). It was published, but unsatisfactory nevertheless. Qian et al (2009) presented a hierarchical modeling alternative implemented in WinBUGS. The original study designed the sampling to mimic a randomized experiment. However, the treatment (benthic hypoxia) cannot be randomly assigned. Confounding factors cannot be ignored. Instead of a hypothesis testing problem (ANOVA), we address the problem as an estimation problem – estimating benthic community abundance and richness in three zones, hypoxic zone, inshore, and offshore "controls."

- MLE model – multilevel model with sampling cores nested in three zones.

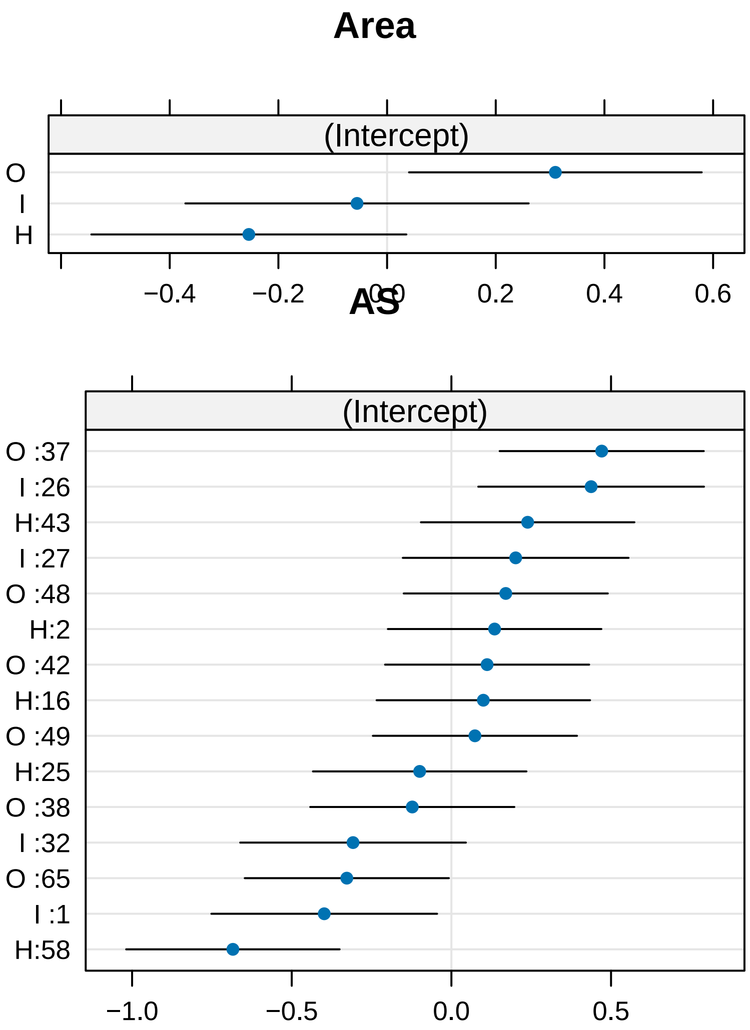
```
benthic.data$AS <- with(benthic.data, factor(paste(Area, Station, sep=":")))
benthicAb.Lme1 <- lmer(log(Abundance) ~ 1+(1|AS)+(1|Area), data=benthic.data)
benthicRn.Lme1 <- lmer(log(Richness) ~ 1+(1|AS)+(1|Area), data=benthic.data)
summary(benthicAb.Lme1)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: log(Abundance) ~ 1 + (1 | AS) + (1 | Area)
##    Data: benthic.data
##
## REML criterion at convergence: 68.3
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -2.19360 -0.56238  0.06669  0.61965  1.45372
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  AS       (Intercept) 0.16598  0.4074
##  Area     (Intercept) 0.08157  0.2856
##  Residual             0.15250  0.3905
## Number of obs: 45, groups:  AS, 15; Area, 3
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)   4.7055     0.2047   22.99
```

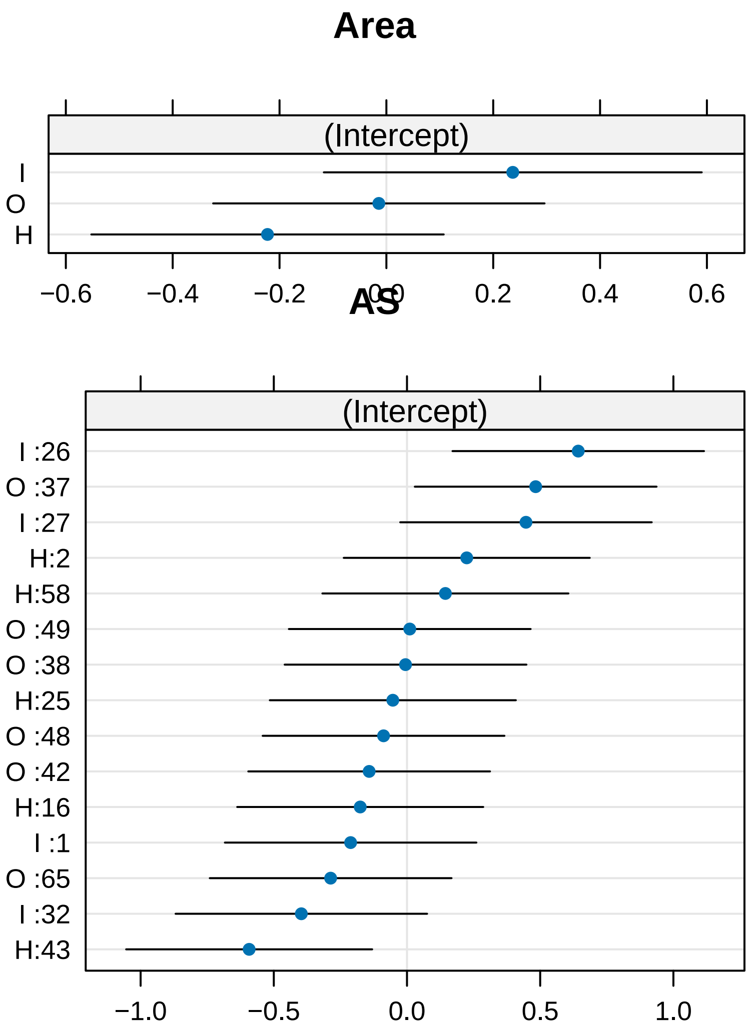
```
summary(benthicRn.Lme1)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: log(Richness) ~ 1 + (1 | AS) + (1 | Area)
##    Data: benthic.data
##
## REML criterion at convergence: 19.2
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -2.79130 -0.44849  0.02172  0.52833  2.12511
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  AS       (Intercept) 0.12665  0.3559
##  Area     (Intercept) 0.10414  0.3227
##  Residual             0.03539  0.1881
## Number of obs: 45, groups:  AS, 15; Area, 3
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)   2.8456     0.2101   13.54
```

```
dotLmeAb1 <- dotplot(ranef(benthicAb.Lme1, condVar=T))
dotLmeRn1 <- dotplot(ranef(benthicRn.Lme1, condVar=T))
print(dotLmeRn1[[1]], pos=c(0,0,1,0.75), more=T)
print(dotLmeRn1[[2]], pos=c(0,0.65, 1,1), more=F)
```



```
print(dotLmeAb1[[1]], pos=c(0,0,1,0.75), more=T)
print(dotLmeAb1[[2]], pos=c(0,0.65, 1,1), more=F)
```



Within zone variance is too high for the MLE method to properly estimate the among zone variance.

- Bayesian model Using a Bayesian hierarchical model, separating cores within each zone:

  $$y_{ijk} \sim N(\mu_{jk}, \sigma^2_{yk})$$

  where, ijk represents the ith observation from the jth core, in zone k, and

  $$\mu_{jk} \sim N(\theta_k, \sigma^2_z)$$

  and finally,

  $$\theta_k \sim N(\mu_{hyp}, \sigma^2_{hyp})$$

```
## Station: core,
## Area: zone
stan1_gom <- "
data{
  int K;  //total sample size
  int J;  //number of sediment cores
  int I;  //number of zones
  real y[K]; //observed response
  int core[K]; //core index
  int zone[K]; //zone index
  int core_zone[J]; //zone
}
parameters{
  real mu[J];
  real theta[I];
  real mu_hyp;
  real<lower=0> sigma_y[I];
  real<lower=0> sigma_i;
  real<lower=0> sigma_hyp;
}
model{
  sigma_hyp ~ normal(0,1); // for fast comuting
  for (i in 1:I){
    theta[i] ~ normal(mu_hyp, sigma_hyp);
  }
  for (j in 1:J){
    mu[j] ~ normal(theta[core_zone[j]], sigma_i);
  }
  for (k in 1:K){
    y[k] ~ normal(mu[core[k]], sigma_y[zone[k]]);
  }
}
generated quantities{
  real delta1;
  real delta2;
  real delta3;
  delta1 = theta[2]-theta[1];
  delta2 = theta[2]-theta[3];
  delta3 = theta[1]-theta[3];
}
"
stan.fit <- stan_model(model_code=stan1_gom)
```

We used a `generated quantities` code block to directly calculate the differences between two zones.

Now organizing input data and run the model

```
stan.in3 <- function(data = benthic.data, y.col=5,  ## richness
                     chains=nchains){ ## no slope
    n <- dim(data)[1]
    y <- log(data[,y.col])
    core <- as.numeric(ordered(data$Station))
    n.core <- max(core)
    zone <- as.numeric(ordered(data$Area))
    n.zone <- max(zone)
    oo <- order(core)
    ind <- cumsum(table(core[oo]))
    Core.zone <- zone[oo][ind] ## each core belongs to which zone
    stan.dat <- list(K=n, J=n.core, I=n.zone, y=y, core=core,
                     zone=zone, core_zone=Core.zone)
    inits <- list()
    for (i in 1:chains)
        inits[[i]] <- list( mu = rnorm(n.core), theta=rnorm(n.zone),
                           mu_hyp=rnorm(1), sigma_i=runif(1),
                           sigma_y=runif(n.zone), sigma_hyp=runif(1))
    parameters <- c("mu","theta","mu_hyp", "sigma_y", "sigma_i",
                     "sigma_hyp",  "delta1", "delta2","delta3")
    return(list(para=parameters, data=stan.dat, inits=inits,
                n.chains=chains))
}

input.to.stan <- stan.in3()  ## long-runs -- results without sigma_hyp prior saved
fit2keep <- sampling(stan.fit, data = input.to.stan$data,
                     init=input.to.stan$inits,
                     pars = input.to.stan$para,
                     iter=niters, thin=nthin,
                     chains=input.to.stan$n.chains,
                     control=list(adapt_delta=0.99, max_treedepth=20))
```

```
## Warning: There were 32 divergent transitions after warmup. See
## https://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.

## Warning: Examine the pairs() plot to diagnose sampling problems

## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail quanti
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#tail-ess
```

```
print(fit2keep)
```

```
## 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% n_eff Rhat
## mu[1]        2.38    0.00 0.09  2.21  2.32  2.37  2.43  2.56  2164    1
## mu[2]        2.72    0.00 0.14  2.44  2.64  2.72  2.81  3.01  1611    1
## mu[3]        2.69    0.00 0.14  2.42  2.60  2.69  2.77  2.96  2535    1
## mu[4]        2.49    0.00 0.13  2.23  2.41  2.49  2.58  2.75  2427    1
## mu[5]        3.24    0.00 0.09  3.05  3.19  3.25  3.30  3.41  1351    1
## mu[6]        3.00    0.00 0.09  2.83  2.95  3.00  3.06  3.17  2402    1
## mu[7]        2.47    0.00 0.09  2.30  2.42  2.47  2.52  2.66  2417    1
## mu[8]        3.61    0.00 0.13  3.35  3.53  3.62  3.70  3.86  2434    1
## mu[9]        3.04    0.00 0.13  2.78  2.95  3.04  3.12  3.30  2569    1
## mu[10]       3.27    0.00 0.13  3.01  3.18  3.27  3.35  3.52  2120    1
## mu[11]       2.82    0.00 0.14  2.53  2.74  2.82  2.90  3.07  2478    1
## mu[12]       3.32    0.00 0.13  3.06  3.24  3.32  3.40  3.57  2313    1
## mu[13]       3.23    0.00 0.13  2.98  3.14  3.22  3.31  3.50  2521    1
## mu[14]       1.93    0.00 0.15  1.65  1.84  1.92  2.02  2.24  1193    1
## mu[15]       2.84    0.00 0.13  2.58  2.75  2.84  2.92  3.11  2130    1
## theta[1]     2.79    0.00 0.19  2.41  2.66  2.78  2.91  3.17  2392    1
## theta[2]     2.60    0.00 0.20  2.22  2.47  2.60  2.72  3.00  2451    1
## theta[3]     3.14    0.00 0.19  2.77  3.02  3.15  3.27  3.50  1548    1
## mu_hyp       2.85    0.01 0.39  2.04  2.66  2.85  3.04  3.66  2305    1
## sigma_y[1]   0.15    0.00 0.05  0.08  0.11  0.14  0.17  0.26  2104    1
## sigma_y[2]   0.24    0.00 0.07  0.15  0.19  0.23  0.27  0.41  2131    1
## sigma_y[3]   0.23    0.00 0.05  0.16  0.19  0.22  0.26  0.36  2513    1
## sigma_i      0.41    0.00 0.11  0.25  0.33  0.39  0.46  0.68  2314    1
## sigma_hyp    0.54    0.01 0.37  0.07  0.28  0.45  0.71  1.48  2095    1
## delta1      -0.19    0.01 0.25 -0.70 -0.35 -0.18 -0.02  0.30  2507    1
## delta2     -0.54    0.01 0.29 -1.08 -0.74 -0.56 -0.35  0.02  1820    1
## delta3     -0.36    0.01 0.26 -0.88 -0.52 -0.36 -0.18  0.12  1985    1
## lp__       52.78    0.11 4.39 43.29 50.05 53.23 55.96 60.32  1669    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 23 11:58:00 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).
```

```
rich_stan <- rvsims(as.matrix(as.data.frame(extract(fit2keep))))

input.to.stan <- stan.in3(y.col = 6)  ## abundance
fit2keep <- sampling(stan.fit, data = input.to.stan$data,
                     init=input.to.stan$inits,
                     pars = input.to.stan$para,
                     iter=niters, thin=nthin,
                     chains=input.to.stan$n.chains,
                     control=list(adapt_delta=0.99, max_treedepth=20))
```

```
## Warning: There were 30 divergent transitions after warmup. See
## https://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

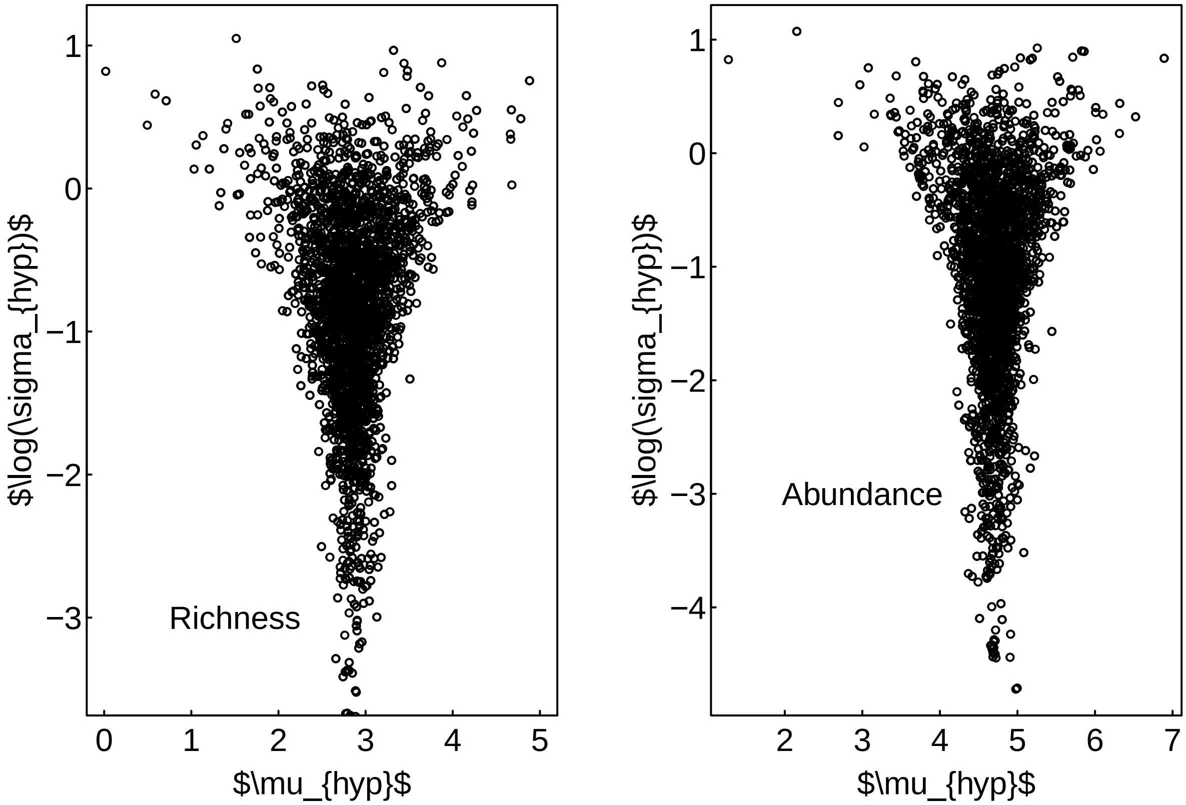
```
print(fit2keep)
```

```
## 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% n_eff Rhat
## mu[1]       4.76    0.01 0.26  4.25  4.60  4.76  4.92  5.27  2523    1
## mu[2]       4.68    0.01 0.25  4.19  4.51  4.68  4.84  5.15  2008    1
## mu[3]       4.33    0.00 0.23  3.88  4.17  4.33  4.48  4.78  2407    1
## mu[4]       4.43    0.00 0.24  3.98  4.28  4.42  4.58  4.93  2440    1
## mu[5]       5.48    0.01 0.32  4.78  5.28  5.51  5.71  6.01  2235    1
## mu[6]       5.31    0.01 0.28  4.72  5.14  5.32  5.49  5.81  2420    1
## mu[7]       4.63    0.01 0.28  4.12  4.44  4.62  4.80  5.20  2396    1
## mu[8]       5.15    0.00 0.22  4.71  5.01  5.16  5.31  5.56  2300    1
## mu[9]       4.68    0.00 0.20  4.27  4.55  4.68  4.80  5.08  2493    1
## mu[10]      4.56    0.00 0.20  4.16  4.42  4.56  4.69  4.97  1795    1
## mu[11]      3.98    0.01 0.28  3.46  3.79  3.97  4.16  4.54  2434    1
## mu[12]      4.60    0.01 0.20  4.21  4.47  4.60  4.74  5.00  1575    1
## mu[13]      4.70    0.00 0.20  4.31  4.57  4.70  4.84  5.09  2326    1
## mu[14]      4.60    0.01 0.24  4.10  4.45  4.60  4.76  5.08  1901    1
## mu[15]      4.42    0.00 0.20  4.02  4.28  4.41  4.55  4.81  2323    1
## theta[1]    4.94    0.01 0.25  4.47  4.77  4.92  5.11  5.45  2218    1
## theta[2]    4.48    0.00 0.23  4.04  4.33  4.48  4.63  4.93  2431    1
## theta[3]    4.69    0.00 0.18  4.32  4.58  4.69  4.80  5.04  2526    1
## mu_hyp      4.70    0.01 0.36  3.87  4.53  4.70  4.88  5.44  2097    1
## sigma_y[1]  0.52    0.00 0.18  0.29  0.39  0.48  0.60  0.97  2162    1
## sigma_y[2]  0.49    0.00 0.12  0.31  0.40  0.47  0.55  0.80  2632    1
## sigma_y[3]  0.39    0.00 0.09  0.26  0.32  0.37  0.43  0.60  2284    1
## sigma_i     0.42    0.00 0.15  0.17  0.32  0.41  0.50  0.76  1977    1
## sigma_hyp   0.49    0.01 0.39  0.04  0.22  0.38  0.65  1.49  2288    1
## delta1     -0.46    0.01 0.35 -1.17 -0.70 -0.44 -0.19  0.14  2104    1
## delta2     -0.20    0.01 0.27 -0.76 -0.37 -0.18 -0.02  0.30  2515    1
## delta3      0.25    0.01 0.29 -0.25  0.05  0.23  0.43  0.89  2263    1
## lp__       20.42    0.10 4.46 11.26 17.63 20.77 23.48 27.85  2059    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 23 11:58:06 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).
```

```
abun_stan <- rvsims(as.matrix(as.data.frame(extract(fit2keep))))

## Extract output
## zone means
tempA <- abun_stan[1:15]
names(tempA) <- paste("$\\mu_{", 1:15, "}$", sep="")
tempR <- rich_stan[1:15]
names(tempR) <- paste("$\\mu_{", 1:15, "}$", sep="")
## zone mean differences
delta0R <- rich_stan[25:27]
delta0A <- abun_stan[25:27]
names(delta0R) <- c("H-I","H-O","I-O")
names(delta0A) <- c("H-I","H-O","I-O")
```

We noticed warning messages about divergence transition. It is a sign of computational difficulties in sampling the posterior distribution.

```
par(mfrow=c(1,2),mar=c(3,3,1,1),
    mgp=c(1.25,0.125,0), las=1,tck=0.01)
plot(rich_stan$mu_hyp, log(rich_stan$sigma_hyp), cex=0.5,
     ylim=c(-3.5,log(3)), xlim=c(-0,5),
     xlab="$\\mu_{hyp}$", ylab="$\\log(\\sigma_{hyp})$")
text(1.5,-3, "Richness")
plot(abun_stan$mu_hyp, log(abun_stan$sigma_hyp), cex=0.5,
##     ylim=c(-3.5,log(150)), xlim=c(-100,100),
     xlab="$\\mu_{hyp}$", ylab="$\\log(\\sigma_{hyp})$")
text(3,-3, "Abundance")
```



An issue we encountered in this analysis is the presence of Neal's funnel, indicating a lack of information to adequately quantify both $\mu_{hyp}$ and σ^2_{hyp} simultaneously. This often results in highly correlated θ_k values. To address this, we require an informative prior for one of the two parameters. In this case, a strong prior was used for σ^2_{hyp}, specifically a half-normal distribution N(0,1). When using a noninformative prior, the computational performance of the program is significantly slower. To mitigate computational difficulties, we can reparameterize θ_k. Instead of directly modeling it as a normal random variable with parameters $\mu_{hyp}$ and σ^2_{hyp}, we introduce a new parameter $z_k \sim N(0, 1)$ and model θ_k as a transformed variable: $\theta_k = \mu_{hyp} + \sigma_{hyp} z_k$. Consequently, we no longer directly sample θ_k, which effectively reduces the occurrence of divergent transitions. However, the underlying model remains the same, and the presence of Neal's funnel persists.

The computational challenges encountered during this analysis prompted us to investigate the problem further. We realized that the available data do not provide sufficient information to simultaneously quantify θ_k, $\mu_{hyp}$, and σ^2_{hyp}, primarily due to the substantial within-zone variation among the cores. The multilevel model indicated that including zone as a random effect does not effectively explain the overall variance. As a result, we proposed two alternative models that avoid directly parameterizing the among-zone variation.

In the first alternative, we removed the zone as a hierarchical factor and treated the core means as exchangeable:

$$\mu_{jk} \sim N(\mu_{hyp}, \sigma^2_{hyp})$$

We estimated each zone mean as the average of the means of the cores within that zone. To accommodate relatively constrained priors for variance parameters (e.g., half-normal N(0,0.5)), the response variable was standardized.

```
###
### First alternative (One-way ANOVA)
###
stan2_gom <- "
data{
  int K;  //total sample size
  int J;  //number of sediment cores
  real y[K]; //observed response
  int core[K]; //core index
}
parameters{
  real mu[J];
  real mu_hyp;
  real<lower=0> sigma_y;
  real<lower=0> sigma_hyp;
}
model{
  sigma_hyp ~ normal(0,0.5);
  sigma_y ~ normal(0,0.5);
  for (j in 1:J){
    mu[j] ~ normal(mu_hyp, sigma_hyp);
  }
  for (k in 1:K){
    y[k] ~ normal(mu[core[k]], sigma_y);
  }
}
"
stan.fit2 <- stan_model(model_code=stan2_gom)

stan.in4 <- function(data = benthic.data, y.col=5,
                     chains=nchains){ ## no slope
  n <- dim(data)[1]
  y <- log(data[,y.col])
  y_ave <- mean(y)
  y_sd <- sd(y)

  y <- (y-y_ave)/y_sd
  core <- as.numeric(ordered(data$Station))
  n.core <- max(core)

  stan.dat <- list(K=n, J=n.core, y=y, core=core)
  inits <- list()
  for (i in 1:chains)
    inits[[i]] <- list( mu = rnorm(n.core),
                        mu_hyp=rnorm(1),
                        sigma_y=runif(1), sigma_hyp=runif(1))
  parameters <- c("mu","mu_hyp", "sigma_y", "sigma_hyp")
  return(list(para=parameters, data=stan.dat, inits=inits,
              n.chains=chains, y_cen=y_ave, y_spd=y_sd))
}

input.to.stan <- stan.in4()
fit2keep <- sampling(stan.fit2, data = input.to.stan$data,
                     init=input.to.stan$inits,
                     pars = input.to.stan$para,
                     iter=niters, thin=nthin,
                     chains=input.to.stan$n.chains)
print(fit2keep)
```

```
## 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% n_eff Rhat
## mu[1]       -0.99    0.00 0.23 -1.43 -1.14 -0.99 -0.85 -0.53  2455    1
## mu[2]       -0.26    0.00 0.22 -0.69 -0.40 -0.25 -0.11  0.18  2440    1
## mu[3]       -0.32    0.00 0.23 -0.76 -0.47 -0.32 -0.16  0.15  2620    1
## mu[4]       -0.75    0.00 0.23 -1.21 -0.90 -0.74 -0.60 -0.30  2423    1
## mu[5]        0.77    0.00 0.23  0.32  0.62  0.77  0.91  1.22  2431    1
## mu[6]        0.27    0.00 0.23 -0.19  0.13  0.27  0.42  0.72  2634    1
## mu[7]       -0.80    0.00 0.22 -1.24 -0.94 -0.80 -0.66 -0.36  2542    1
## mu[8]        1.55    0.00 0.23  1.09  1.40  1.55  1.71  2.01  2443    1
## mu[9]        0.29    0.00 0.22 -0.15  0.14  0.29  0.43  0.74  2428    1
## mu[10]       0.79    0.00 0.23  0.35  0.64  0.79  0.94  1.24  2497    1
## mu[11]      -0.04    0.00 0.23 -0.48 -0.19 -0.04  0.10  0.41  2550    1
## mu[12]       0.91    0.00 0.24  0.43  0.75  0.91  1.06  1.36  2562    1
## mu[13]       0.71    0.01 0.23  0.25  0.56  0.71  0.87  1.19  2007    1
## mu[14]      -1.99    0.00 0.23 -2.44 -2.15 -1.99 -1.84 -1.52  2217    1
## mu[15]      -0.14    0.00 0.23 -0.57 -0.29 -0.14  0.01  0.31  2445    1
## mu_hyp       0.00    0.00 0.24 -0.47 -0.17 -0.01  0.16  0.48  2546    1
## sigma_y      0.40    0.00 0.05  0.31  0.37  0.40  0.43  0.52  2400    1
## sigma_hyp    0.90    0.00 0.16  0.64  0.78  0.88  0.99  1.23  2392    1
## lp__         9.08    0.07 3.61  1.02  6.97  9.49 11.57 15.04  2385    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 23 11:58:58 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
rich_stan2 <- rvsims(as.matrix(as.data.frame(extract(fit2keep))))

## processing output
core <- as.numeric(ordered(benthic.data$Station))
n.core <- max(core)
zone <- as.numeric(ordered(benthic.data$Area))
n.zone <- max(zone)
oo <- order(core)
ind <- cumsum(table(core[oo]))
Core.zone <- zone[oo][ind] ## each core belongs to which zone
input_sd <- input.to.stan$y_spd
input_mu <- input.to.stan$y_cen
## return to the original scale
core1.musR <- input_mu + input_sd*rich_stan2[1:15]
zone11.Rmu <- mean(core1.musR[Core.zone==1])
zone12.Rmu <- mean(core1.musR[Core.zone==2])
zone13.Rmu <- mean(core1.musR[Core.zone==3])

deltaR11 = zone12.Rmu-zone11.Rmu
deltaR12 = zone12.Rmu-zone13.Rmu
deltaR13 = zone11.Rmu-zone13.Rmu

delta1R <- c(deltaR11, deltaR12, deltaR13)
names(delta1R) <- c("H-I","H-O","I-O")

## repeat for Abundance:
```

```
input.to.stan <- stan.in4(y.col=6)
fit2keep <- sampling(stan.fit2, data = input.to.stan$data,
                     init=input.to.stan$inits,
                     pars = input.to.stan$para,
                     iter=niters, thin=nthin,
                     chains=input.to.stan$n.chains)
```

```
## Warning: There were 1 divergent transitions after warmup. See
## https://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.

## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
print(fit2keep)
```

```
## 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% n_eff Rhat
## mu[1]       -0.03    0.01 0.34  -0.69  -0.25 -0.03  0.19  0.64  2556    1
## mu[2]        0.11    0.01 0.33  -0.55  -0.10  0.11  0.33  0.76  2431    1
## mu[3]       -0.57    0.01 0.34  -1.25  -0.79 -0.57 -0.35  0.10  2618    1
## mu[4]       -0.35    0.01 0.34  -1.01  -0.58 -0.35 -0.12  0.34  2335    1
## mu[5]        1.38    0.01 0.39   0.58   1.13  1.39  1.65  2.12  2394    1
## mu[6]        1.06    0.01 0.38   0.30   0.80  1.06  1.33  1.79  2265    1
## mu[7]       -0.34    0.01 0.34  -1.00  -0.56 -0.34 -0.11  0.35  2607    1
## mu[8]        0.80    0.01 0.36   0.12   0.56  0.81  1.04  1.50  2549    1
## mu[9]        0.01    0.01 0.34  -0.64  -0.22  0.01  0.25  0.67  2676    1
## mu[10]      -0.23    0.01 0.35  -0.95  -0.45 -0.23 -0.01  0.44  2306    1
## mu[11]      -1.24    0.01 0.38  -1.93  -1.50 -1.25 -0.98 -0.48  2481    1
## mu[12]      -0.15    0.01 0.33  -0.79  -0.38 -0.15  0.08  0.50  2123    1
## mu[13]       0.02    0.01 0.33  -0.66  -0.20  0.02  0.24  0.67  2021    1
## mu[14]      -0.02    0.01 0.33  -0.67  -0.24 -0.02  0.20  0.64  2329    1
## mu[15]      -0.48    0.01 0.34  -1.14  -0.71 -0.48 -0.25  0.20  2557    1
## mu_hyp       0.00    0.00 0.22  -0.42  -0.14  0.00  0.14  0.47  2375    1
## sigma_y      0.66    0.00 0.09   0.52   0.60  0.65  0.71  0.86  2619    1
## sigma_hyp    0.73    0.00 0.16   0.44   0.62  0.72  0.83  1.09  2461    1
## lp__       -10.13    0.07 3.36 -17.62 -12.26 -9.74 -7.68 -4.75  2445    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 23 11:59:02 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).
```

```
abun_stan2 <- rvsims(as.matrix(as.data.frame(extract(fit2keep))))

## processing output
core <- as.numeric(ordered(benthic.data$Station))
n.core <- max(core)
zone <- as.numeric(ordered(benthic.data$Area))
n.zone <- max(zone)
oo <- order(core)
ind <- cumsum(table(core[oo]))
Core.zone <- zone[oo][ind] ## each core belongs to which zone
```

```r
input_sd <- input.to.stan$y_spd
input_mu <- input.to.stan$y_cen
## return to the original scale
core1.musA <- input_mu + input_sd*abun_stan2[1:15]
zone11.Amu <- mean(core1.musA[Core.zone==1])
zone12.Amu <- mean(core1.musA[Core.zone==2])
zone13.Amu <- mean(core1.musA[Core.zone==3])

deltaA11 = zone12.Amu-zone11.Amu
deltaA12 = zone12.Rmu-zone13.Amu
deltaA13 = zone11.Rmu-zone13.Amu

delta1A <- c(deltaA11, deltaA12, deltaA13)
names(delta1A) <- c("H-I","H-O","I-O")
```

The second alternative is to mimic the multilevel model:

$$\begin{array}{rcl} y_i & \sim & N(\mu_i, \sigma_y^2) \\ \mu_i & = & \mu_0 + \alpha_{k[i]} + \beta_{j[i]} \\ \alpha_k & \sim & N(0, \sigma_z^2) \\ \beta_j & \sim & N(0, \sigma_c^2) \end{array}$$

where $k[i]$ and $j[i]$ represent that the ith observation is in kth zone and jth core.

```r
stan3_gom <- "
data{
  int K;  //total sample size
  int J;  //number of sediment cores
  int I;  //number of zones
  real y[K]; //observed response
  int core[K]; //core index
  int zone[K]; //zone index
}
parameters{
  real muK[J];
  real muZ[I];
  real mu0;
  real<lower=0> sigmaY;
  real<lower=0> sigmaK;
  real<lower=0> sigmaZ;
}
model{
  sigmaK ~ normal(0,0.5);
  sigmaZ ~ normal(0,0.5);
  sigmaY ~ normal(0,0.5);
  for (i in 1:I){
    muZ[i] ~ normal(0, sigmaZ);
  }
  for (j in 1:J){
    muK[j] ~ normal(0, sigmaK);
  }
  for (k in 1:K){
    y[k] ~ normal(mu0+muK[core[k]]+muZ[zone[k]], sigmaY);
  }
}
```

```
generated quantities{
  real delta1;
  real delta2;
  real delta3;
  delta1 = muZ[2]-muZ[1];
  delta2 = muZ[2]-muZ[3];
  delta3 = muZ[1]-muZ[3];
}
"

stan.fit3 <- stan_model(model_code=stan3_gom)

stan.in5 <- function(data = benthic.data, y.col=5, chains=nchains){
  n <- dim(data)[1]
  y <- log(data[,y.col])
  y_ave <- mean(y)
  y_sd <- sd(y)
  y <- (y-y_ave)/y_sd
  core <- as.numeric(ordered(data$Station))
  n.core <- max(core)
  zone <- as.numeric(ordered(data$Area))
  n.zone <- max(zone)

  stan.dat <- list(K=n, J=n.core, I=n.zone, y=y, core=core, zone=zone)
  inits <- list()
  for (i in 1:chains)
    inits[[i]] <- list( muK = rnorm(n.core),
                        muZ=rnorm(n.zone),
                        mu0=rnorm(1),
                       sigmaY=runif(1), sigmaK=runif(1),
                       sigmaZ=runif(1))
  parameters <- c("mu0","muK", "muZ", "sigmaY", "sigmaK",
                  "sigmaZ", "delta1", "delta2", "delta3")
  return(list(para=parameters, data=stan.dat, inits=inits,
              n.chains=chains, y_cen=y_ave, y_spd=y_sd))
}

input.to.stan <- stan.in5()
fit2keep <- sampling(stan.fit3, data = input.to.stan$data,
                     init=input.to.stan$inits,
                     pars = input.to.stan$para,
                     iter=niters, thin=nthin,
                     chains=input.to.stan$n.chains,
                     control=list(adapt_delta=0.99, max_treedepth=15))
```

```
## Warning: There were 2 divergent transitions after warmup. See
## https://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.

## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
print(fit2keep)
```

```
## 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% n_eff Rhat
## mu0     -0.03    0.01 0.41 -0.89 -0.27 -0.03  0.20  0.80  2457    1
## muK[1]  -0.84    0.01 0.38 -1.61 -1.08 -0.83 -0.59 -0.08  2346    1
## muK[2]   0.18    0.01 0.38 -0.57 -0.07  0.19  0.45  0.91  2256    1
## muK[3]   0.11    0.01 0.38 -0.64 -0.15  0.11  0.37  0.85  2176    1
## muK[4]  -0.29    0.01 0.39 -1.05 -0.55 -0.29 -0.03  0.46  2345    1
## muK[5]   0.88    0.01 0.38  0.15  0.64  0.87  1.13  1.63  2336    1
## muK[6]   0.39    0.01 0.38 -0.31  0.13  0.39  0.64  1.16  2332    1
## muK[7]  -0.65    0.01 0.37 -1.38 -0.90 -0.65 -0.40  0.07  2358    1
## muK[8]   1.07    0.01 0.39  0.32  0.81  1.06  1.33  1.87  2251    1
## muK[9]  -0.15    0.01 0.37 -0.84 -0.39 -0.16  0.10  0.59  2195    1
## muK[10]  0.33    0.01 0.38 -0.41  0.08  0.33  0.57  1.09  2232    1
## muK[11]  0.40    0.01 0.38 -0.32  0.14  0.40  0.65  1.15  2159    1
## muK[12]  0.45    0.01 0.38 -0.26  0.19  0.43  0.69  1.23  2196    1
## muK[13]  0.25    0.01 0.39 -0.50 -0.01  0.24  0.51  1.01  2072    1
## muK[14] -1.50    0.01 0.40 -2.28 -1.77 -1.50 -1.22 -0.74  2296    1
## muK[15] -0.55    0.01 0.37 -1.26 -0.80 -0.56 -0.31  0.17  2237    1
## muZ[1]  -0.11    0.01 0.42 -1.00 -0.34 -0.09  0.13  0.74  2247    1
## muZ[2]  -0.45    0.01 0.45 -1.41 -0.70 -0.40 -0.14  0.33  2408    1
## muZ[3]   0.51    0.01 0.46 -0.29  0.21  0.48  0.78  1.51  2088    1
## sigmaY   0.40    0.00 0.05  0.32  0.37  0.40  0.43  0.52  2574    1
## sigmaK   0.75    0.00 0.15  0.49  0.64  0.73  0.84  1.11  2360    1
## sigmaZ   0.54    0.01 0.26  0.09  0.36  0.52  0.70  1.12  1975    1
## delta1  -0.33    0.01 0.42 -1.20 -0.60 -0.31 -0.04  0.45  2208    1
## delta2  -0.96    0.01 0.51 -1.93 -1.31 -0.98 -0.60  0.00  1941    1
## delta3  -0.63    0.01 0.45 -1.53 -0.92 -0.61 -0.32  0.19  2150    1
## lp__    11.67    0.08 3.79  3.32  9.38 11.92 14.35 18.28  2388    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 23 12:00:01 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).
```

```
rich_stan3 <- rvsims(as.matrix(as.data.frame(extract(fit2keep))))
input_sd <- input.to.stan$y_spd
input_mu <- input.to.stan$y_cen
zone2.musR <- input_sd*rich_stan3[17:19]
core2.musR <- input_mu+input_sd*rich_stan3[2:16]

zone21.Rmu <- mean(core2.musR[Core.zone==1]) + zone2.musR[1]
zone22.Rmu <- mean(core2.musR[Core.zone==2]) + zone2.musR[2]
zone23.Rmu <- mean(core2.musR[Core.zone==3]) + zone2.musR[3]

deltaR21 = zone22.Rmu-zone21.Rmu
deltaR22 = zone22.Rmu-zone23.Rmu
deltaR23 = zone21.Rmu-zone23.Rmu

delta2R <- c(deltaR21, deltaR22, deltaR23)
names(delta2R) <- c("H-I","H-O","I-O")

input.to.stan <- stan.in5(y.col=6)
fit2keep <- sampling(stan.fit3, data = input.to.stan$data,
                     init=input.to.stan$inits,
                     pars = input.to.stan$para,
                     iter=niters, thin=nthin,
                     chains=input.to.stan$n.chains,
                     control=list(adapt_delta=0.99, max_treedepth=15))
```

```
## Warning: There were 7 divergent transitions after warmup. See
## https://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.

## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
print(fit2keep)
```

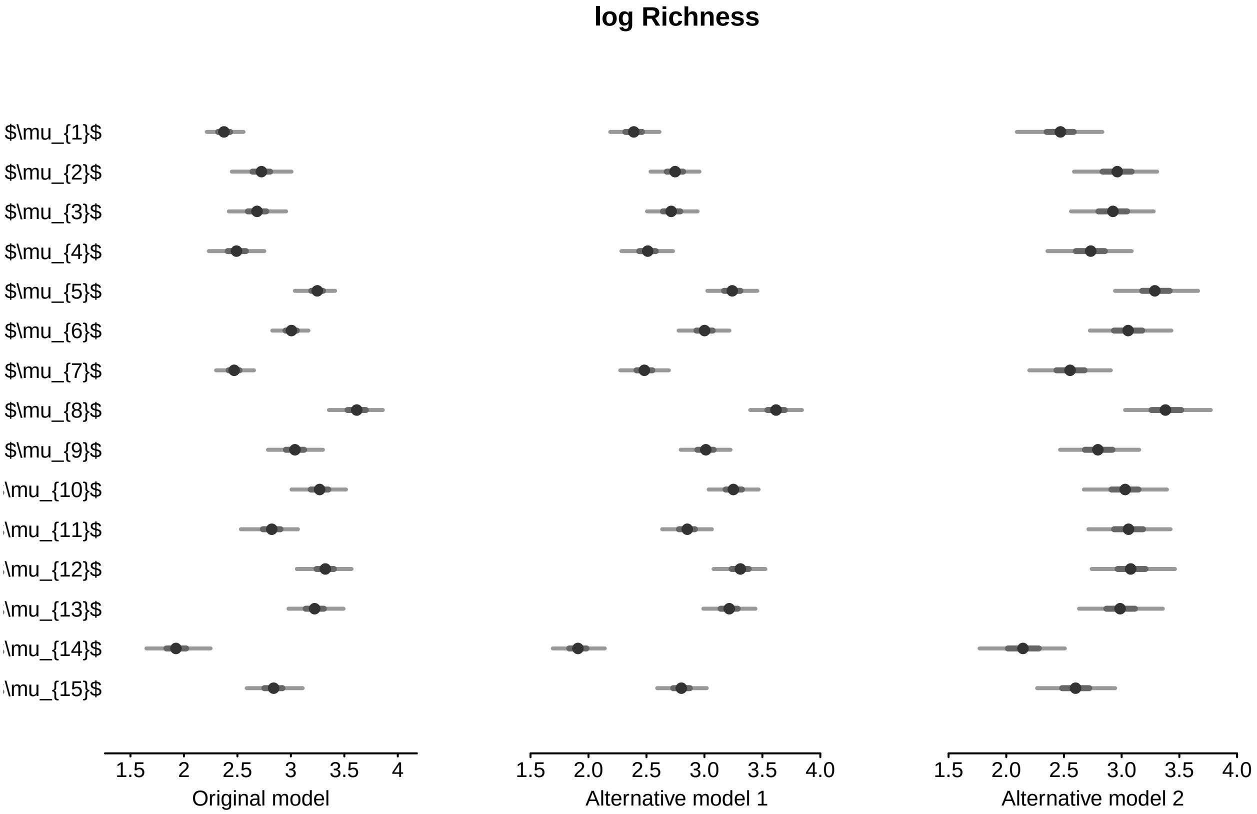
```
## Inference for Stan model: anon_model.
## 8 chains, each with iter=5000; warmup=2500; thin=8;
## post-warmup draws per chain=313, total post-warmup draws=2504.
##
##          mean se_mean   sd   2.5%    25%   50%   75% 97.5% n_eff Rhat
## mu0      0.01    0.01 0.37  -0.73  -0.19  0.01  0.23  0.74  2400    1
## muK[1]  -0.27    0.01 0.43  -1.15  -0.56 -0.26  0.01  0.55  2391    1
## muK[2]   0.31    0.01 0.41  -0.48   0.02  0.32  0.57  1.12  2382    1
## muK[3]  -0.32    0.01 0.42  -1.16  -0.60 -0.30 -0.04  0.50  2502    1
## muK[4]  -0.13    0.01 0.40  -0.91  -0.40 -0.12  0.14  0.68  2451    1
## muK[5]   1.07    0.01 0.48   0.14   0.73  1.07  1.40  2.02  2252    1
## muK[6]   0.77    0.01 0.45  -0.08   0.45  0.75  1.09  1.69  2406    1
## muK[7]  -0.56    0.01 0.42  -1.41  -0.83 -0.55 -0.27  0.24  2301    1
## muK[8]   0.77    0.01 0.41  -0.01   0.49  0.77  1.04  1.59  2539    1
## muK[9]   0.01    0.01 0.38  -0.77  -0.24  0.01  0.25  0.75  2321    1
## muK[10] -0.21    0.01 0.39  -0.97  -0.47 -0.21  0.04  0.54  2447    1
## muK[11] -0.98    0.01 0.46  -1.90  -1.29 -0.97 -0.67 -0.09  2469    1
## muK[12] -0.14    0.01 0.38  -0.89  -0.39 -0.14  0.11  0.63  2042    1
## muK[13]  0.02    0.01 0.39  -0.75  -0.23  0.02  0.27  0.78  2444    1
## muK[14]  0.16    0.01 0.41  -0.63  -0.10  0.17  0.43  0.94  2457    1
## muK[15] -0.44    0.01 0.40  -1.24  -0.70 -0.44 -0.17  0.32  2300    1
## muZ[1]   0.33    0.01 0.41  -0.31   0.04  0.26  0.56  1.29  2396    1
## muZ[2]  -0.29    0.01 0.40  -1.20  -0.51 -0.24 -0.03  0.40  2248    1
## muZ[3]  -0.02    0.01 0.36  -0.76  -0.20 -0.01  0.16  0.76  2418    1
## sigmaY   0.67    0.00 0.09   0.52   0.60  0.66  0.72  0.86  2442    1
## sigmaK   0.66    0.00 0.17   0.36   0.55  0.65  0.77  1.04  2380    1
## sigmaZ   0.44    0.01 0.27   0.04   0.24  0.41  0.59  1.05  2124    1
## delta1  -0.63    0.01 0.50  -1.68  -0.98 -0.60 -0.22  0.16  2199    1
## delta2  -0.28    0.01 0.38  -1.12  -0.52 -0.24  0.00  0.37  2020    1
## delta3   0.35    0.01 0.40  -0.31   0.05  0.30  0.61  1.22  2358    1
## lp__    -8.13    0.08 3.94 -16.89 -10.49 -7.75 -5.37 -1.38  2155    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 23 12:00:08 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).
```

```
abun_stan3 <- rvsims(as.matrix(as.data.frame(extract(fit2keep))))

input_sd <- input.to.stan$y_spd
input_mu <- input.to.stan$y_cen
```

```
zone2.musA <- input_sd*abun_stan3[17:19]
core2.musA <- input_mu+input_sd*abun_stan3[2:16]

zone21.Amu <- mean(core2.musA[Core.zone==1]) + zone2.musA[1]
zone22.Amu <- mean(core2.musA[Core.zone==2]) + zone2.musA[2]
zone23.Amu <- mean(core2.musA[Core.zone==3]) + zone2.musA[3]

deltaA21 = zone22.Amu-zone21.Amu
deltaA22 = zone22.Amu-zone23.Amu
deltaA23 = zone21.Amu-zone23.Amu

delta2A <- c(deltaA21, deltaA22, deltaA23)
names(delta2A) <- c("H-I","H-O","I-O")
```

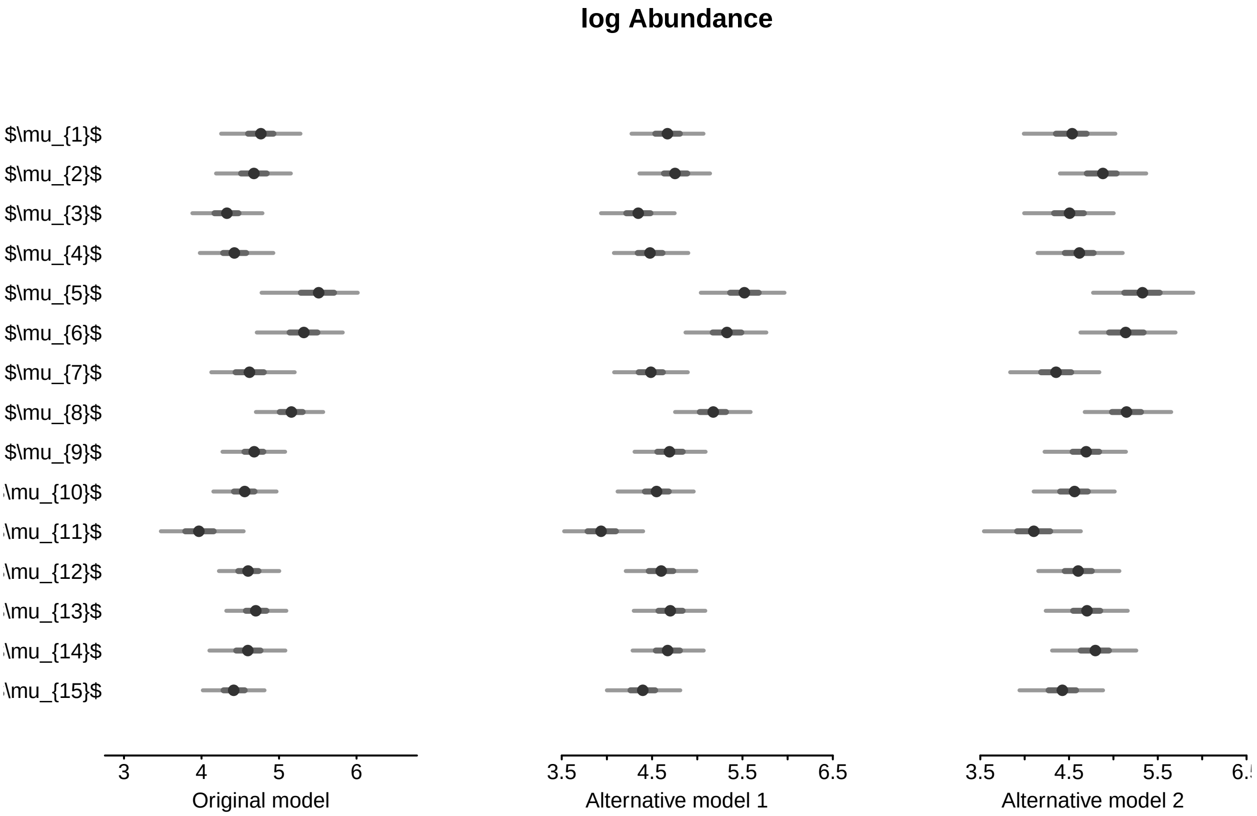
Now we make some comparisons of the three alternative models

```
## 1. Estimated core means
par(mfrow=c(1,3), mar=c(3,1,2,0.1), mgp=c(1.25,0.125,0), las=1, tck=-0.01)
mlplot(tempR, xlab="Original model", top.axis=F)
mlplot(core1.musR, xlab="Alternative model 1",
       main="log Richness", axes=F)
axis(1)
mlplot(core2.musR, xlab="Alternative model 2", axes=F)
axis(1)
```



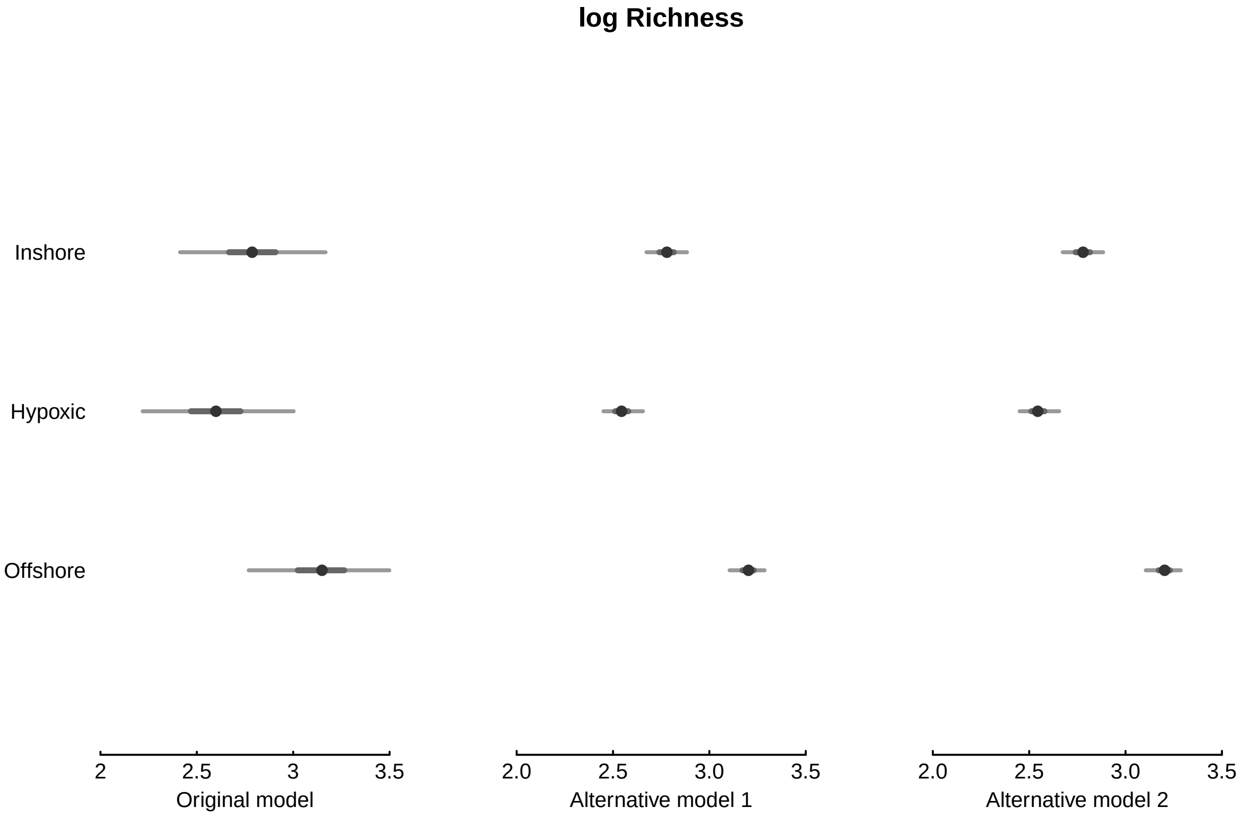
```
par(mfrow=c(1,3), mar=c(3,1,2,0.1), mgp=c(1.25,0.125,0), las=1, tck=-0.01)
mlplot(tempA, xlab="Original model", top.axis=F)
mlplot(core1.musA, xlab="Alternative model 1",
       main="log Abundance", axes=F)
```

```
axis(1)
mlplot(core2.musA, xlab="Alternative model 2", axes=F)
axis(1)
```

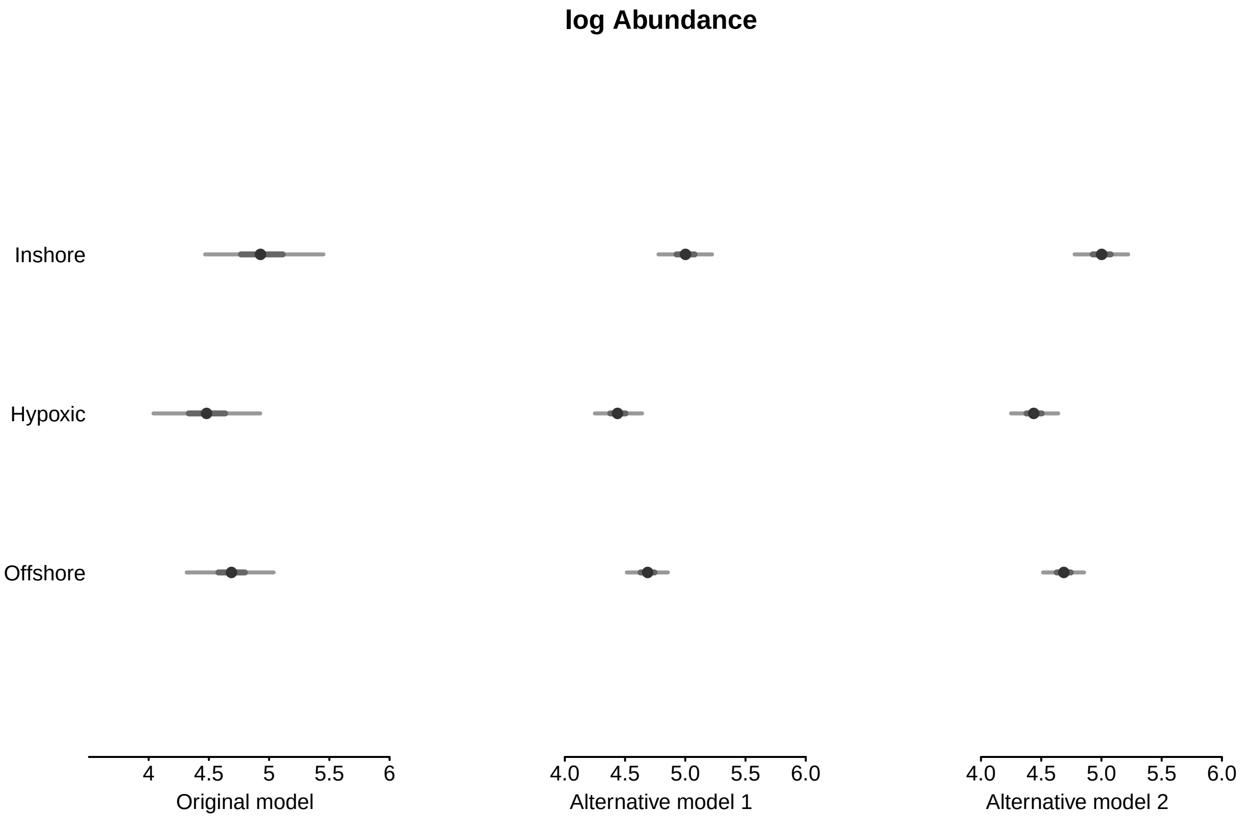


```
## 2. Estimated zone means
#### extract zone means and mean differences
zone0R.thetas <- rich_stan[16:18]
names(zone0R.thetas) <- c("Inshore", "Hypoxic","Offshore")
zone0A.thetas <- abun_stan[16:18]
names(zone0A.thetas) <- c("Inshore", "Hypoxic","Offshore")

zone1R.thetas <- c(zone11.Rmu, zone12.Rmu, zone13.Rmu)
names(zone1R.thetas) <- c("Inshore", "Hypoxic","Offshore")

zone1A.thetas <- c(zone11.Amu, zone12.Amu, zone13.Amu)
names(zone1A.thetas) <- c("Inshore", "Hypoxic","Offshore")

zone2R.thetas <- c(zone21.Rmu, zone22.Rmu, zone23.Rmu)
names(zone2R.thetas) <- c("Inshore", "Hypoxic","Offshore")

zone2A.thetas <- c(zone21.Amu, zone22.Amu, zone23.Amu)
names(zone2A.thetas) <- c("Inshore", "Hypoxic","Offshore")

par(mfrow=c(1,3), mar=c(3,1,2,0.1), mgp=c(1.25,0.125,0), las=1, tck=0.01)
mlplot(zone0R.thetas, xlab="Original model", top.axis=F, xlim=c(2,3.5))
abline(v=0)
mlplot(zone1R.thetas, xlab="Alternative model 1", main="log Richness",
       axes=F, xlim=c(2,3.5))
```

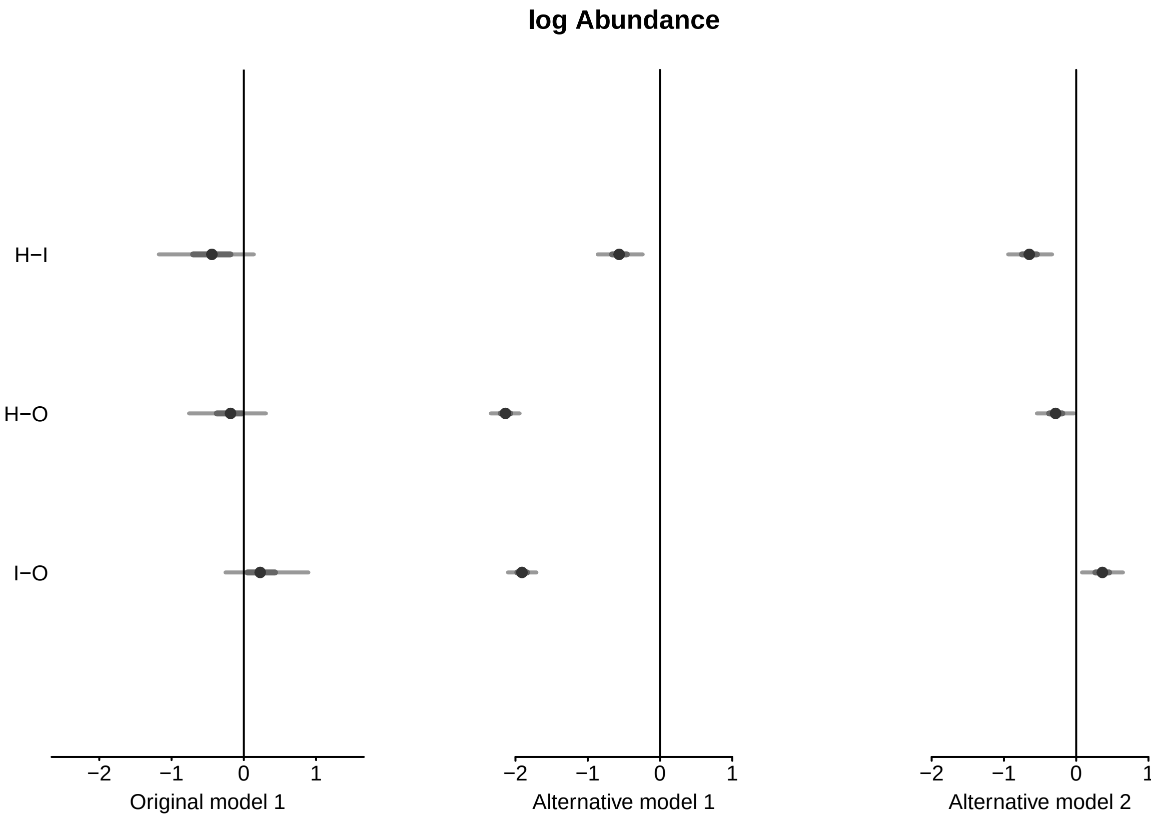
```
axis(1)
abline(v=0)
mlplot(zone1R.thetas, xlab="Alternative model 2", axes=F, xlim=c(2,3.5))
axis(1)
abline(v=0)
```



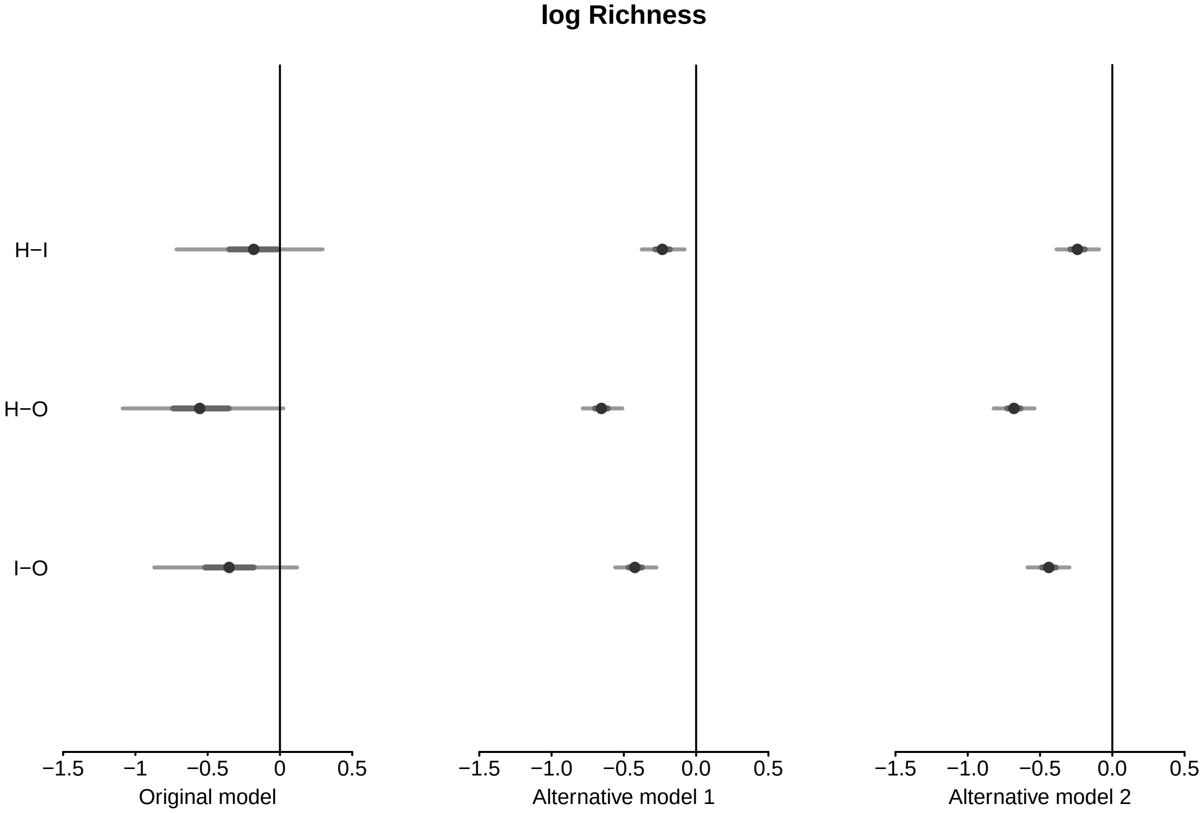
```
par(mfrow=c(1,3), mar=c(3,1,2,0.1), mgp=c(1.25,0.125,0), las=1, tck=-0.01)
mlplot(zone0A.thetas, xlab="Original model", top.axis=F, xlim=c(3.6,6))
mlplot(zone1A.thetas, xlab="Alternative model 1", main="log Abundance",
       axes=F, xlim=c(3.6,6))
axis(1)
mlplot(zone1A.thetas, xlab="Alternative model 2", axes=F, xlim=c(3.6,6))
axis(1)
```



```
## between zone differences
par(mfrow=c(1,3), mar=c(3,1,2,0.1), mgp=c(1.25,0.125,0), las=1, tck= -0.01)
mlplot(delta0A, xlab="Original model 1", top.axis=F, xlim=c(-2.5,1.5))
abline(v=0)
mlplot(delta1A, xlab="Alternative model 1",
       main="log Abundance", axes=F,
       xlim=c(-2.5,1.5))
axis(1)
abline(v=0)
mlplot(delta2A, xlab="Alternative model 2", axes=F, xlim=c(-2.5,1.5))
axis(1)
abline(v=0)
```



```
par(mfrow=c(1,3), mar=c(3,1,2,0.1), mgp=c(1.25,0.125,0), las=1, tck=-0.01)
mlplot(delta0R, xlab="Original model", top.axis=F, xlim=c(-1.5,0.5))
abline(v=0)
mlplot(delta1R, xlab="Alternative model 1",
       main="log Richness", axes=F, xlim=c(-1.5,0.5))
abline(v=0)
axis(1)
mlplot(delta2R, xlab="Alternative model 2", axes=F, xlim=c(-1.5,0.5))
axis(1)
abline(v=0)
```



### Example 2 – Zero-Inflation or Not? In this example, we introduce the concept of Bayesian posterior simulation for model evaluation.

Incidental Catch in Fisheries (Hilborn and Mangel, 1997. *Ecological Detective*) aimed to determine the minimum sample size of "statistically meaningful data" required to estimate the mean bycatch rate with a limited level of uncertainty. The original study (Bartle, 1991) used Central Limit Theorem-based confidence intervals to define an acceptable level of uncertainty. The goal was to estimate the mean and variance of the bycatch numbers. Hilborn and Mangel (1997) utilized the negative binomial distribution to describe the bycatch data, where the response variable is the bycatch count, a count variable taking only non-negative integer values.

The Poisson distribution is perhaps the most commonly used probability distribution for count data. Fisher et al. (1943) suggested that the Poisson distribution is one of three types of distributions for biological measurements.

The Poisson distribution has one parameter (λ), representing the mean. The probability function is given by

$$\pi(Y = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

When observing independent observations $y_1, \cdots, y_n$, the log-likelihood function is

$$LL = \log\left(\prod_{i=1}^{n} \frac{\lambda^k e^{-\lambda}}{k!}\right) \propto \log(\lambda)\sum_{i=1}^{n} y_i - n\lambda$$

In a Bayesian statistics, we normally use the gamma distribution as the prior for λ, because the posterior distribution of λ is also a gamma distribution. If the prior is $\lambda \sim \text{gamma}(\alpha, \beta)$, the posterior is $\lambda \mid y_1, \cdots, y_n \sim \text{gamma}(\alpha + \sum_{i=1}^{y}, \beta + n)$. The gamma distribution is the same as the χ^2 distribution, which Fisher et al (1943) used to derive the negative binomial distribution.

We fit the bycatch data using both Poisson and binomial distributions to show why the negative binomial model is more useful.

```r
zipdata <- data.frame(Capture=0:17,
                      numb=c(807,37,27,8,4,4,1,3,1,0,0,2,1,1,0,0,0,1))
zippos <- rep(zipdata[-1,1], zipdata$numb[-1])
zip0 <- zipdata$numb[1]

##############################################
## Simple models (without zero inflation) ####
##############################################

## the Poisson model
Pois_bycatch <- "
data{
      int<lower=1> n0; //number of 0's
      int<lower=1> np; //number of non-zero counts
      int<lower=1> yp[np];
   }
   parameters{
      real<lower=0> lambda;
   }
   model{
     target += n0*poisson_log_lpmf(0|log(lambda));
     target += poisson_log_lpmf(yp|log(lambda));
   }
"
fitPois <- stan_model(model_code=Pois_bycatch)

## Neg_binomial
NB_bycatch <- "
data{
      int<lower=1> n0; //number of 0's
      int<lower=1> np; //number of non-zero counts
      int<lower=1> yp[np];
   }
   parameters{
      real<lower=0> mu;
      real<lower=0> r;
   }
   model{
     mu ~ normal(0,2);
     r ~ normal(0,2);
     target += n0*neg_binomial_2_log_lpmf(0 | log(mu), r);
     target += neg_binomial_2_log_lpmf(yp | log(mu), r);
   }
"
fitNB <- stan_model(model_code=NB_bycatch)
```

We separated 0s from non-zero observations to make the code more comparable to the zero-inflated alternatives.

```r
## one input function for both models
bycatch_input <- function(yp=zippos, n0=zip0, model="Pois", n.chains=nchains){
    N <- length(yp)+n0
    np <- length(yp)
    data <- list(np=np, n0=n0, yp=yp)
    inits <- list()
    for (i in 1:n.chains){
        if (model=="Pois")
            inits[[i]] <- list(lambda=runif(1))
        else
            inits[[i]] <- list(mu=runif(1), r=runif(1))
    }
    if (model=="Pois")
        pars <- c("lambda")
    else
        pars <- c("mu","r")
  return(list(data=data, init=inits, nchains=n.chains, para=pars, model=model))
}
```

Fitting the model

```
## the Poisson model
input.to.stan <- bycatch_input()
keep_Pois <- sampling(fitPois, data=input.to.stan$data,
                 init=input.to.stan$init,
                 pars=input.to.stan$para,
                 iter=niters,thin=nthin,
                 chains=input.to.stan$nchains)##,
##                 control=list(max_treedepth=20))
print(keep_Pois)
```

```
## 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% n_eff Rhat
## lambda    0.28    0.00 0.02    0.25    0.27    0.28    0.29    0.31  2661    1
## lp__   -789.86    0.01 0.67 -791.68 -790.02 -789.60 -789.44 -789.39  2241    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 23 12:01:56 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).
```

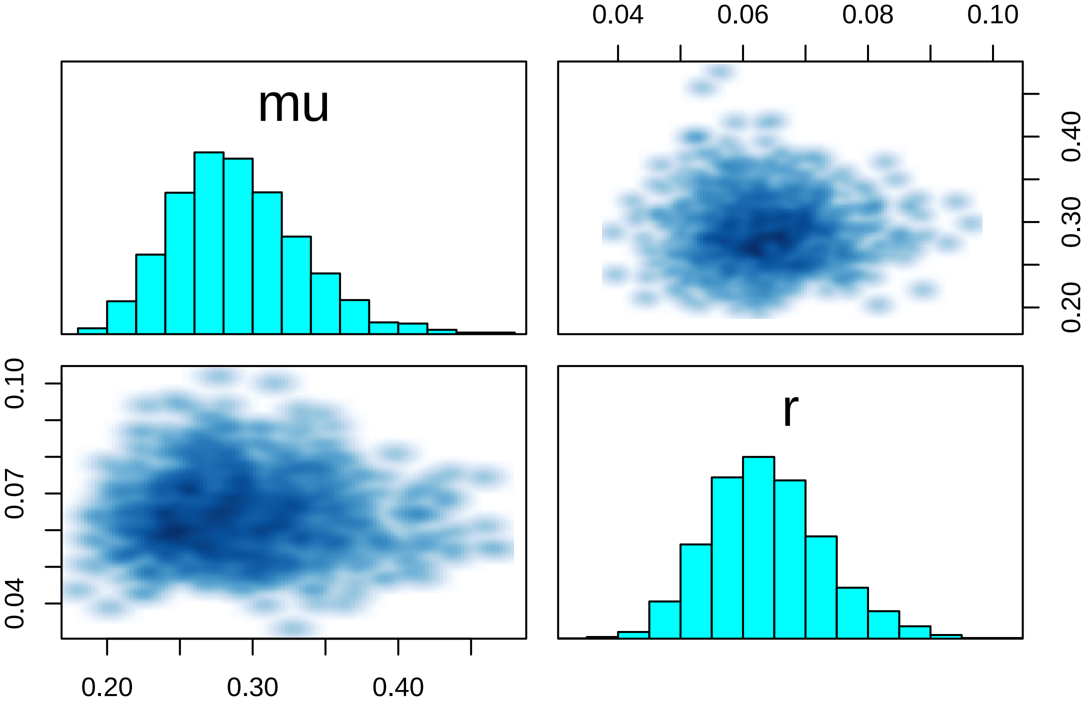
```
Pois_stanout <- extract(keep_Pois, pars="lambda")
Pois_coef <- rvsims(as.matrix(as.data.frame(Pois_stanout)))

## negative bin:
input.to.stan <- bycatch_input(model="NB")
keep_NB <- sampling(fitNB, data=input.to.stan$data,
                 init=input.to.stan$init,
                 pars=input.to.stan$para,
                 iter=niters,thin=nthin,
                 chains=input.to.stan$nchains)##,
##                 control=list(max_treedepth=20))
print(keep_NB)
```

```
## 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.
##
```

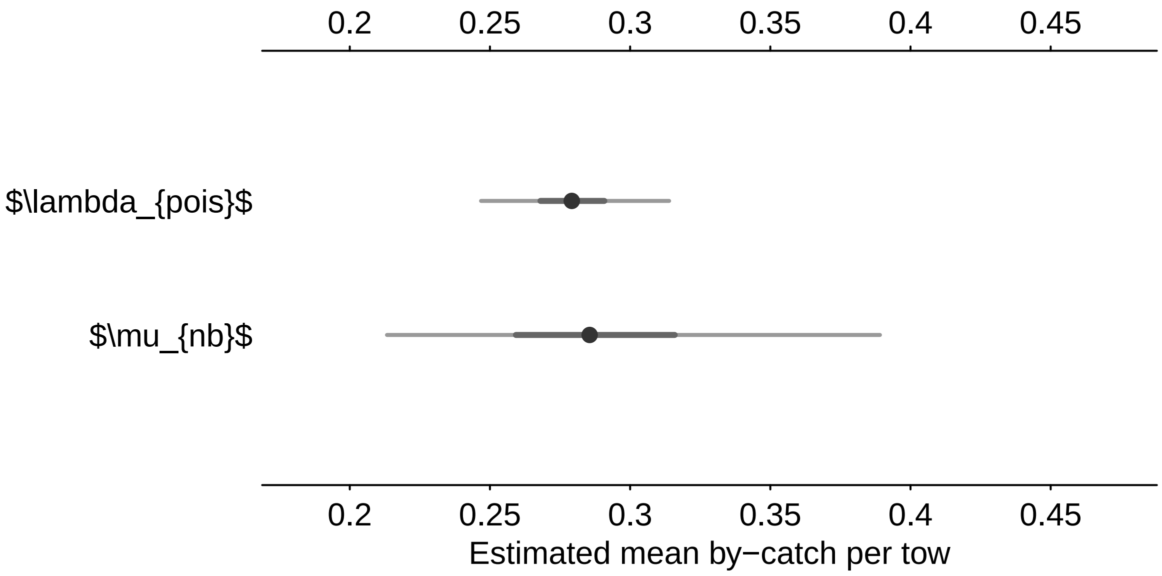
```
##          mean se_mean   sd    2.5%     25%     50%     75%   97.5% n_eff Rhat
## mu       0.29    0.00 0.04    0.21    0.26    0.28    0.32    0.39  2306    1
## r        0.06    0.00 0.01    0.05    0.06    0.06    0.07    0.08  2581    1
## lp__ -457.15    0.02 0.97 -459.72 -457.54 -456.84 -456.45 -456.19  2470    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 23 12:02:00 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).
```

```
pairs(keep_NB, pars=c("mu","r"))
```

```
## Warning in par(usr): argument 1 does not name a graphical parameter

## Warning in par(usr): argument 1 does not name a graphical parameter
```

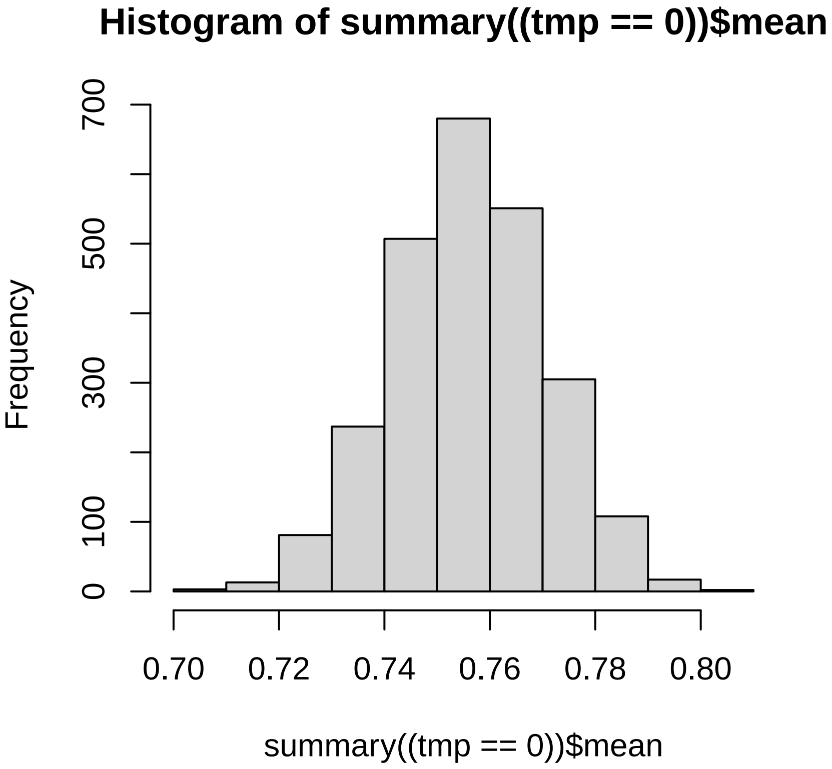


```
NB_stanout <- extract(keep_NB, pars=c("mu","r"))
NB_coef <- rvsims(as.matrix(as.data.frame(NB_stanout)))

## Comparison
means <- c(Pois_coef[1], NB_coef[1])
names(means) <- c("$\\lambda_{pois}$", "$\\mu_{nb}$")

par(mgp=c(1.25,0.25,0), tck=-0.01)
mlplot(means, xlab="Estimated mean by-catch per tow")
```

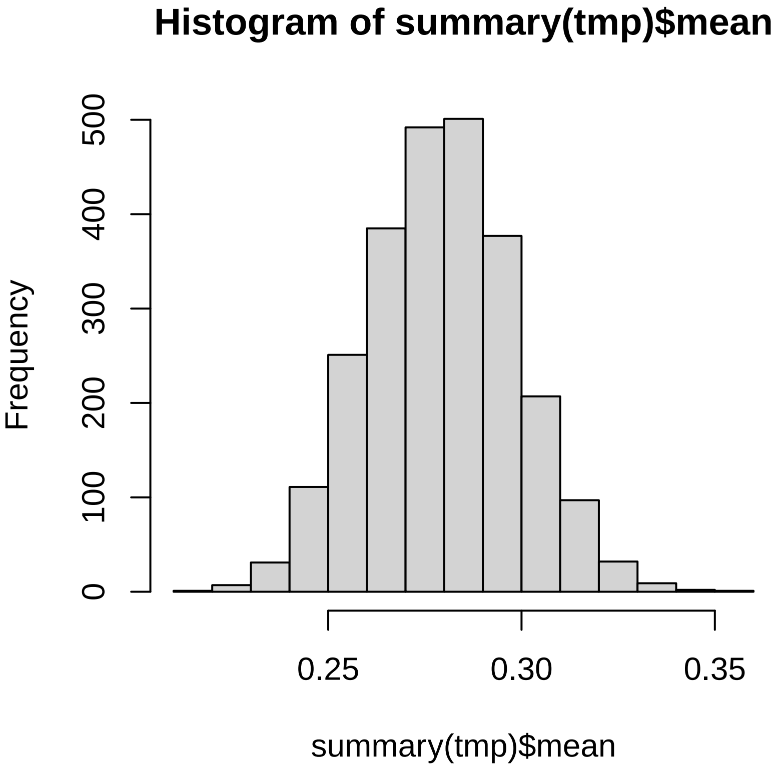


Posterior simulation – using the fitted model to replicate the data repeatedly

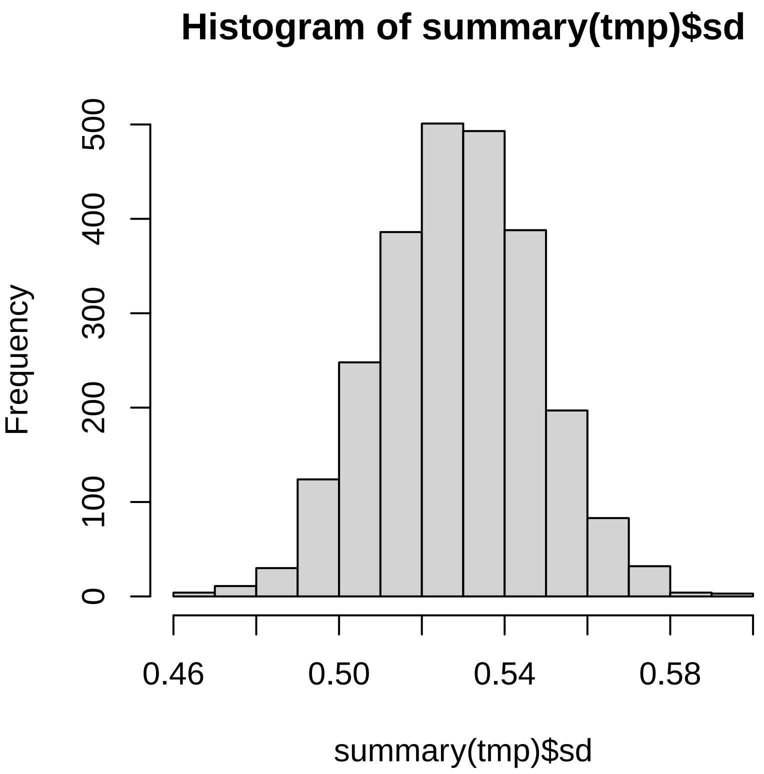
```
## simulations
nsims <- length(Pois_stanout$lambda)
n <- length(zippos)+zip0

### Poisson model
n0_pois <- mu_pois <- sig_pois <- NULL
for (i in 1:nsims){
    tmp <- rpois(n, lambda=Pois_stanout$lambda[i])
    n0_pois <- c(n0_pois, mean(tmp==0))
    mu_pois <- c(mu_pois, mean(tmp))
    sig_pois <- c(sig_pois, sd(tmp))
}
### here is why I like `rv`
tmp <- rvpois(1, Pois_stanout$lambda)
hist(summary((tmp==0))$mean)
```



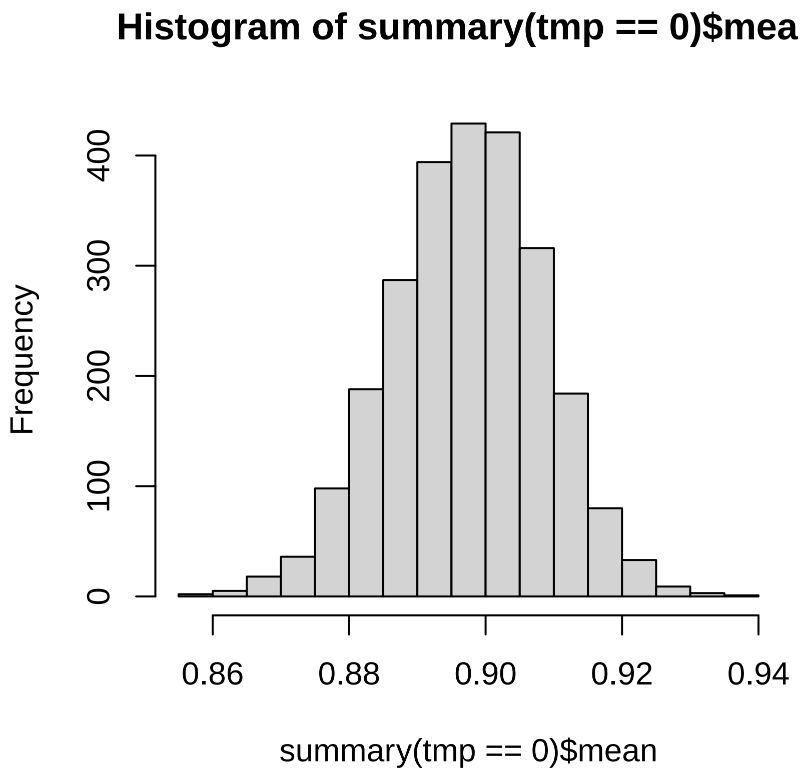
```
hist(summary(tmp)$mean)
```



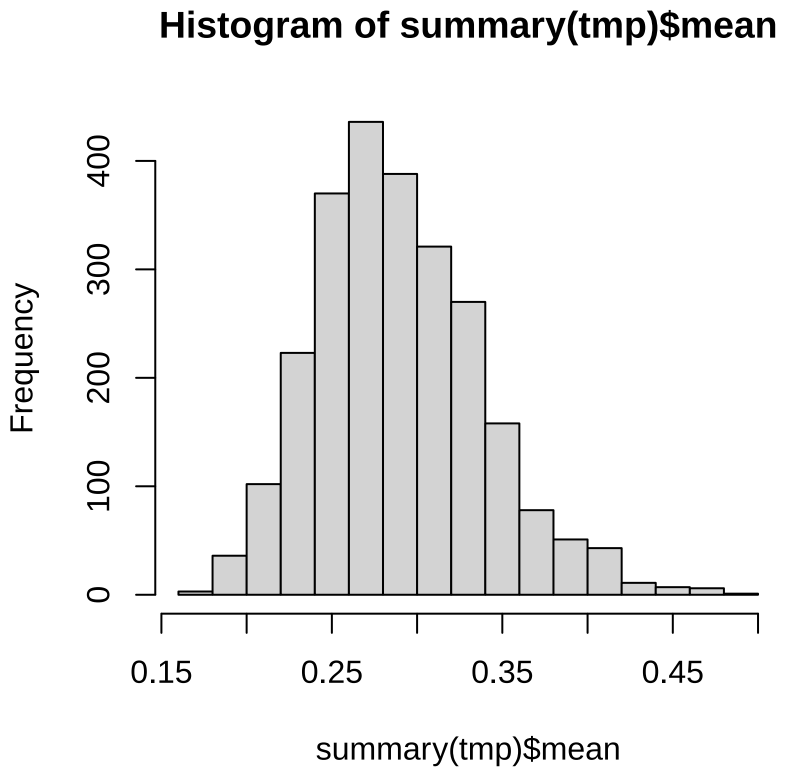
```
hist(summary(tmp)$sd)
```



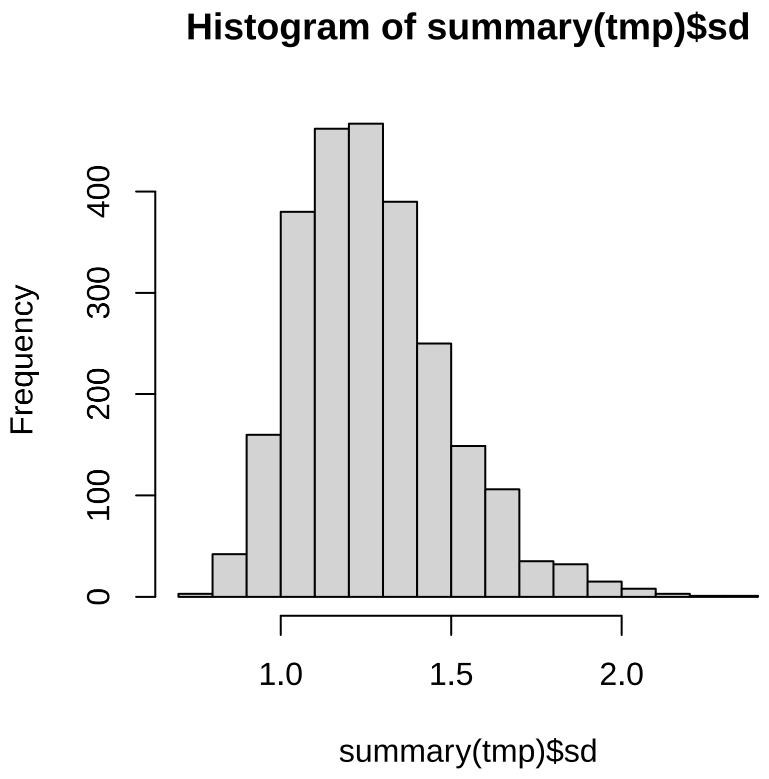
```
### negative Bin:
n0_NB <- mu_NB <- sig_NB <- NULL
for (i in 1:nsims){
    tmp <- rnbinom(n, mu=NB_stanout$mu[i], size=NB_stanout$r[i])
    n0_NB <- c(n0_NB, mean(tmp==0))
    mu_NB <- c(mu_NB, mean(tmp))
    sig_NB <- c(sig_NB, sd(tmp))
}
## or use rv (generating from gamma-poisson conjugate):
tmp1 <- rvgamma(1, NB_stanout$r, NB_stanout$r/NB_stanout$mu)
tmp <- rvpois(1, tmp1)
hist(summary(tmp==0)$mean)
```



```
hist(summary(tmp)$mean)
```

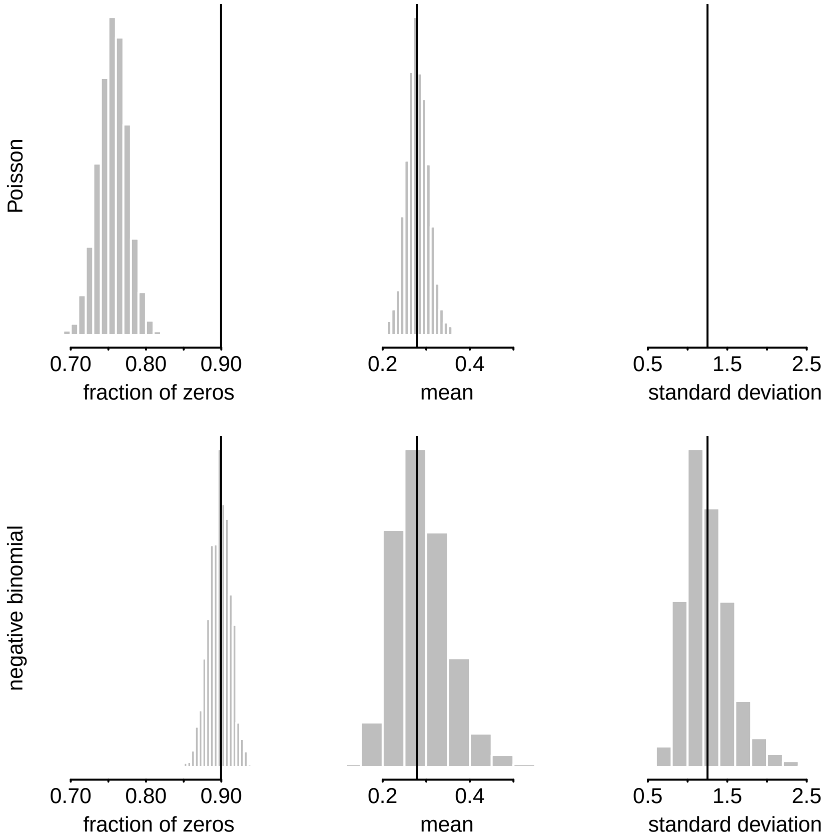


```
hist(summary(tmp)$sd)
```



```
par(mfrow=c(2,3), mar=c(2.5,2.5,1,1), mgp=c(1.25,0.125,0), tck=-0.01)
hist(n0_pois, xlim=range(c(n0_pois, n0_NB, zip0/n)),
     xlab="fraction of zeros", ylab="Poisson", main="",
     border="white", density=-1, col="gray", yaxt="n")
abline(v=zip0/n)
hist(mu_pois,
     xlim=range(c(mu_pois, mu_NB, mean(c(rep(0,zip0),zippos)))),
     xlab="mean", ylab="", yaxt="n", main="",
     border="white", density=-1, col="gray")
abline(v=mean(c(rep(0,zip0),zippos)))
#hist(sig_pois,
#     xlim=range(c(sig_pois, sig_NB, sd(c(rep(0,zip0),zippos)))),
#     xlab="standard deviation", ylab="", yaxt="n", main="",
#     border="white", density=-1, col="gray")
hist(sig_pois,
     xlim=range(c(sig_pois, sig_NB,sd(c(rep(0,zip0),zippos)))),
     xlab="standard deviation", ylab="", main="",yaxt="n",
     border="white", density=-1, col=1)
abline(v=sd(c(rep(0,zip0),zippos)))

hist(n0_NB, xlim=range(c(n0_pois, n0_NB,zip0/n)),yaxt="n",
     xlab="fraction of zeros", ylab="negative binomial", main="",
     border="white", density=-1, col="gray")
abline(v=zip0/n)
hist(mu_NB,
     xlim=range(c(mu_pois, mu_NB,mean(c(rep(0,zip0),zippos)))),
     xlab="mean", ylab="", main="",yaxt="n",
     border="white", density=-1, col="gray")
abline(v=mean(c(rep(0,zip0),zippos)))
```

```
hist(sig_NB,
     xlim=range(c(sig_pois, sig_NB,sd(c(rep(0,zip0),zippos)))),
     xlab="standard deviation", ylab="", main="",yaxt="n",
     border="white", density=-1, col="gray")
abline(v=sd(c(rep(0,zip0),zippos)))
```



The Poisson model cannot replicate the variance and the fraction of zeros of the data. The data are zero inflated if the Poisson model is used.

Zero-Inflated count data models

The zero-inflated model is designed to handle count data that exhibits excess zeros. Under certain conditions, the observations are always 0 (true zero). For example, there were no sea birds present. Without knowing the conditions, we must treat observed 0s as a combination of true zeros and 0s resulted from, for example, the imperfect sampling method (i.e., bycatch happened but failed to observe). Assuming that the chance of sampling a true 0 is θ (and a probability of $1-\theta$ sampling a Poisson counts). The probability of observing any outcome y_i is defined as follows: - The probability of observing 0 is given by $\pi(y_i) = \theta + (1-\theta) \cdot Pois(0|\lambda)$, where $Pois(0|\lambda)$ represents the probability of observing a zero count in a Poisson distribution with mean λ. - The probability of observing $y_i > 0$ is given by $\pi(y_i) = (1-\theta) \cdot Pois(y_i|\lambda)$, representing the probability of observing a non-zero count in the Poisson distribution.

$$\pi(y_i) = \begin{cases} \theta + (1-\theta)Pois(0 \mid \lambda) & \text{if } y_i = 0 \\ (1-\theta)Pois(y_i \mid \lambda) & \text{if } y_i > 0. \end{cases}$$

The likelihood function is

$$L = [\theta + (1-\theta)e^{-\lambda}]^{n_0} \times \prod_{i=1}^{n_p} (1-\theta)\frac{\lambda^{y_i}e^{-\lambda}}{y_i!}$$

The log-likelihood function is

$$LL = n_0 \log[\theta + (1-\theta)e^{-\lambda}] + \left(n_p \log(1-\theta) + \sum_{i=1}^{n} \log\left(\frac{\lambda^{y_i} e^{-\lambda}}{y_i!}\right)\right)$$

Writing the log-likelihood in Stan:

```
model{
  target += n0*log_sum_exp(log(theta), log1m(theta)-\lambda);
  target += np*log1m(theta) + poisson_lpmf(yp| lambda);
}
```

Because $\log(\theta)$ is the log of probability of observing a true 0, we can also directly use the Stan function `bernoulli_logit_lpmf` to write the code:

```
model{
  target += n0*log_sum_exp(bernoulli_logit_lpmf(1|zi),
bernoulli_logit_lpmf(0|zi)+poisson_log_lpmf(0|eta));
  target += np*bernoulli_logit_lpmf(0|zi) +
            poisson_log_lpmf(yp | eta);
}
```

where `zi` is the logit of the probability of a true zero ($z_i = \text{logit}(\theta)$) and `eta` is the log of the Poisson parameter ($\eta = \log(\lambda)$).

Lambert (1992) introduced a generalized linear model for zero-inflated Poisson process. In the model, the Poisson model parameter λ and the binomial model parameter θ are modeled as linear functions of predictors through the respective link function:

$$\begin{aligned} \log(\lambda) &= \mathbf{X}\beta \\ \text{logit}(\theta) &= \mathbf{Z}\alpha \end{aligned}$$

The R package `pscl` implements the MLE of the Lambert ZIP model.

Fitting the models:

```
zip_bycatch <-"
data{
  int<lower=1> n0;
  int<lower=1> np;
  int<lower=1> yp[np];
}
parameters{
  real<lower=0,upper=1> theta;
  real<lower=0> lambda;
}
transformed parameters{
  real eta;
  real zi;
  eta = log(lambda);
  zi = logit(theta);
}
model{
  theta ~ beta(1,1);
  lambda ~ normal(0,5);
  target += n0*log_sum_exp(bernoulli_logit_lpmf(1|zi),
bernoulli_logit_lpmf(0|zi)+poisson_log_lpmf(0|eta));
  target += np*bernoulli_logit_lpmf(0|zi) +
            poisson_log_lpmf(yp | eta);
}
"

zinb_bycatch <-"
data{
  int<lower=1> n0;
  int<lower=1> np;
  int<lower=1> yp[np];
}
parameters{
  real<lower=0,upper=1> theta;
  real<lower=0> mu;
  real<lower=0> r;
}
transformed parameters{
  real eta;
  eta=log(mu);
}
model{
  theta ~ beta(1,1);
  mu ~ normal(0,5);
  r ~ normal(0,5);
  target += n0*log_sum_exp(log(theta),
                           log1m(theta)+neg_binomial_2_log_lpmf(0|eta,r));
  target += np*log1m(theta) +
            neg_binomial_2_log_lpmf(yp | eta, r);
}
"


##############################################
fit_zip <- stan_model(model_code = zip_bycatch)
fit_zinb <- stan_model(model_code = zinb_bycatch)
##############################################
```

Processing data and initial values

```
##############################################
## The data and initial values
##############################################
ZI_bycatch <- function(yp=zippos, n0=zip0, model="zip", n.chains=nchains){
    np <- length(yp)
    N <- np+n0
    data <- list(np=np, n0=n0, yp=yp)
    inits <- list()
    for (i in 1:n.chains){
        if (model == "zip")
            inits[[i]] <- list(lambda=runif(1), theta=runif(1))
        else
            inits[[i]] <- list(mu=runif(1), r=runif(1), theta=runif(1))
    }
    if (model == "zip")
        paras <- c("lambda","theta")
    else
```

```
        paras <- c("theta","mu","r")
  return(list(data=data, init=inits, nchains=n.chains, para=paras, model=model))
}

input.to.stan <- ZI_bycatch()
keep_zip <- sampling(fit_zip, data=input.to.stan$data,
                 init=input.to.stan$init,
                 pars=input.to.stan$para,
                 iter=niters,thin=nthin,
                 chains=input.to.stan$nchains)##,
##                   control=list(max_treedepth=20))
print(keep_zip)
```

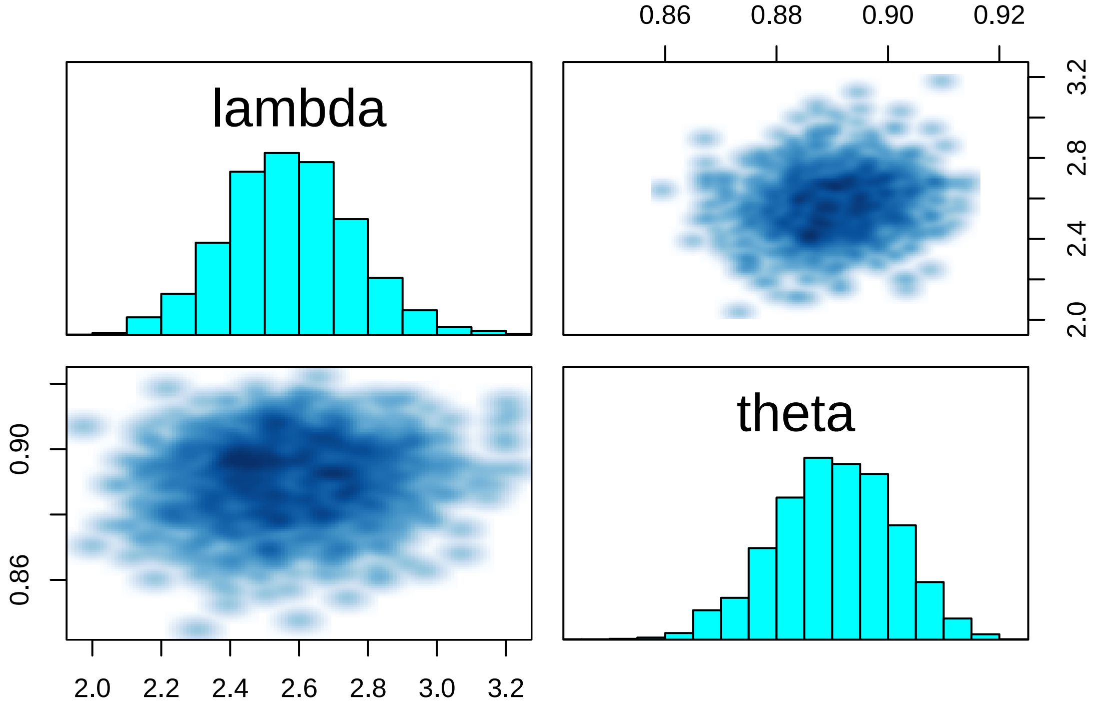
```
## Inference for Stan model: anon_model.
## 8 chains, each with iter=5000; warmup=2500; thin=8;
## post-warmup draws per chain=313, total post-warmup draws=2504.
##
##           mean se_mean   sd    2.5%     25%     50%     75%   97.5% n_eff Rhat
## lambda    2.57    0.00 0.19    2.21    2.44    2.56    2.69    2.94  2660    1
## theta     0.89    0.00 0.01    0.87    0.88    0.89    0.90    0.91  2472    1
## lp__   -501.65    0.02 1.00 -504.30 -502.05 -501.34 -500.95 -500.69  2498    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 23 12:04:00 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).
```

```
input.to.stan <- ZI_bycatch(model="zinb")
keep_zinb <- sampling(fit_zinb, data=input.to.stan$data,
                 init=input.to.stan$init,
                 pars=input.to.stan$para,
                 iter=niters,thin=nthin,
                 chains=input.to.stan$nchains)##,
##                   control=list(max_treedepth=20))
print(keep_zinb)
```

```
## 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% n_eff Rhat
## theta    0.62    0.00 0.20    0.11    0.53    0.68    0.77    0.84  1904    1
## mu       0.94    0.01 0.41    0.31    0.61    0.91    1.22    1.81  2133    1
## r        0.30    0.00 0.20    0.07    0.15    0.26    0.40    0.80  2082    1
## lp__  -456.92    0.03 1.33 -460.32 -457.56 -456.59 -455.94 -455.36  2159    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 23 12:04:05 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).
```

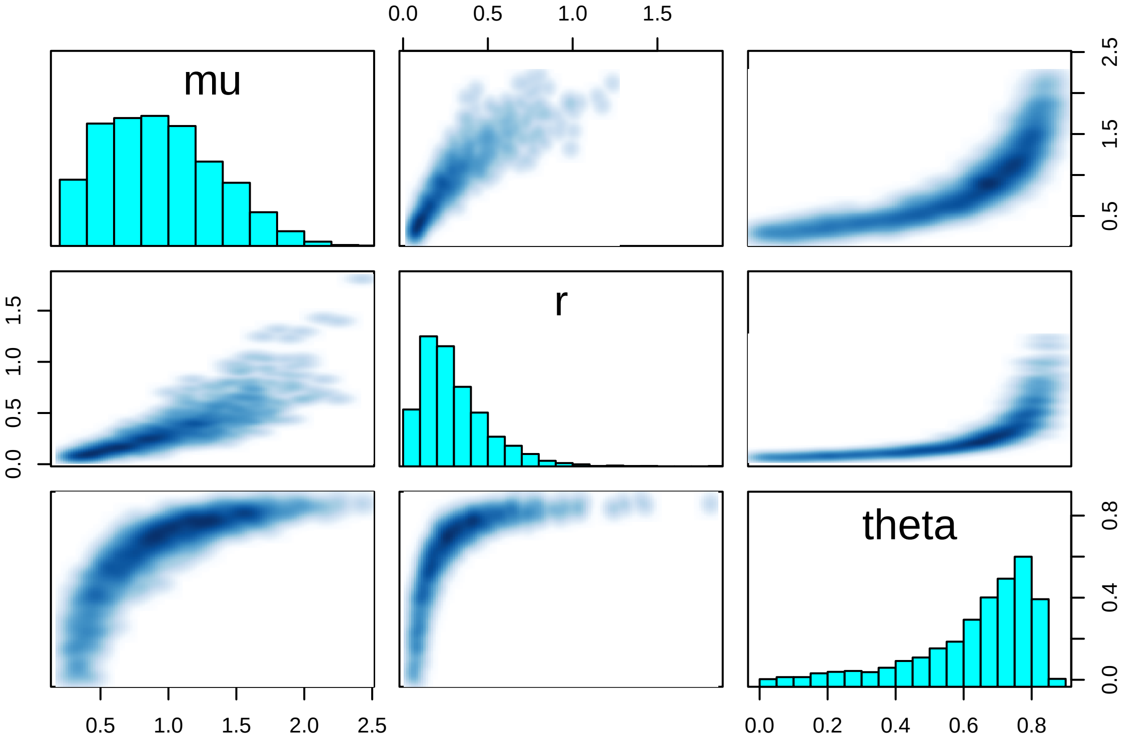
```
##save(keep,file="zinb_bycatch.RData")
pairs(keep_zip, pars=c("lambda","theta"))
```

```
## Warning in par(usr): argument 1 does not name a graphical parameter
## Warning in par(usr): argument 1 does not name a graphical parameter
```



```
pairs(keep_zinb, pars=c("mu","r","theta"))
```

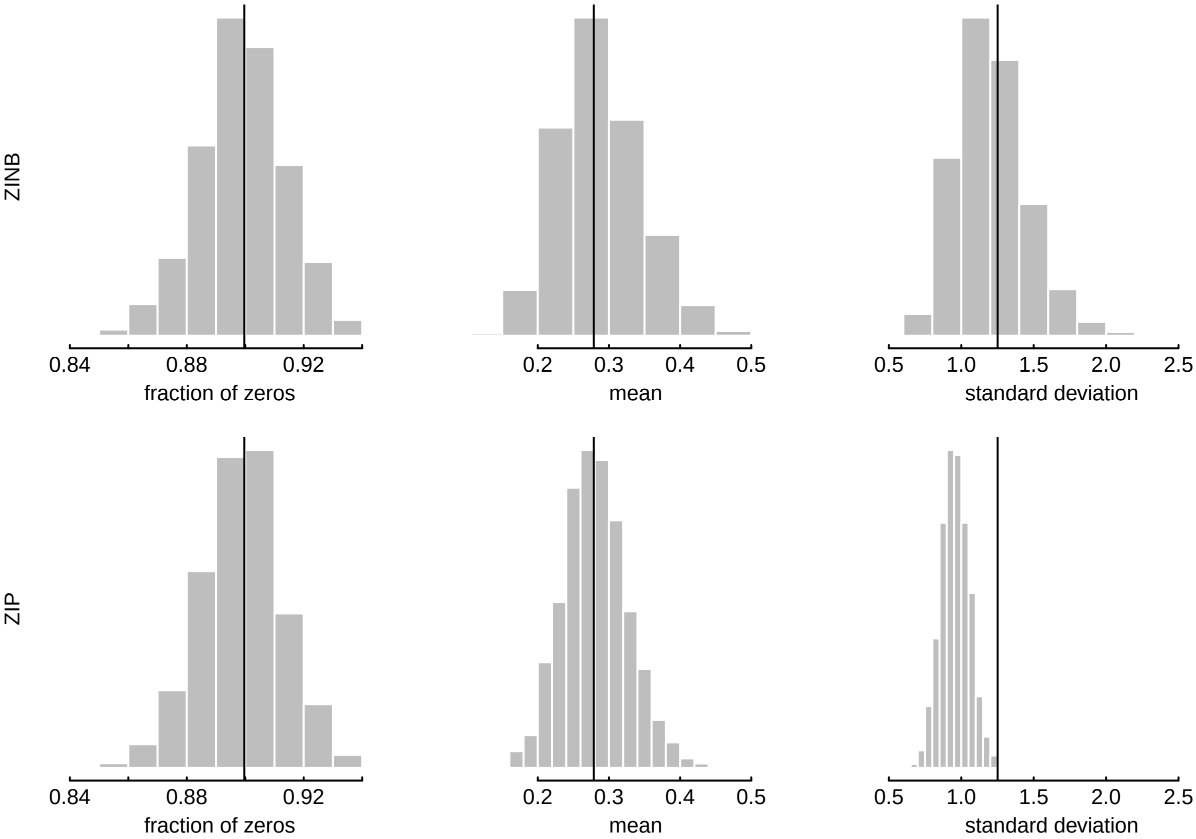
```
## Warning in par(usr): argument 1 does not name a graphical parameter

## Warning in par(usr): argument 1 does not name a graphical parameter

## Warning in par(usr): argument 1 does not name a graphical parameter
```



Strong correlation in the zinb model indicating non-identifiability. Additional information (e.g., of θ) is needed.

Making comparisons

```
zinb_coef <- as.data.frame(extract(keep_zinb, permute=T,
                                    pars=c("theta","mu","r")))
zip_coef <- as.data.frame(extract(keep_zip, permute=T,
                                   pars=c("theta","lambda")))

mu <- mean(c(rep(0,zip0),zippos))
s <- sd(c(rep(0,zip0),zippos))

## percent of 0 from zinb model
n0_zinb <- NULL
mu_zinb <- NULL
sig_zinb <- NULL
for (i in 1:dim(zinb_coef)[1]){
    n0tmp <- rbinom(1, n, zinb_coef$theta[i])
    tmp <- c(rep(0, n0tmp), rnbinom(n-n0tmp, mu=zinb_coef$mu[i],
                                    size=zinb_coef$r[i]))
    n0_zinb <- c(n0_zinb, mean(tmp==0))
    mu_zinb <- c(mu_zinb, mean(tmp))
    sig_zinb <- c(sig_zinb, sd(tmp))
}

## percent of 0 from zip model
n0_zip <- NULL
mu_zip <- NULL
sig_zip <- NULL
for (i in 1:dim(zip_coef)[1]){
    n0tmp <- rbinom(1, n, zip_coef$theta[i])
    tmp <- c(rep(0, n0tmp), rpois(n-n0tmp,
                                    lambda=zip_coef$lambda[i]))
    n0_zip <- c(n0_zip, mean(tmp==0))
    mu_zip <- c(mu_zip, mean(tmp))
    sig_zip <- c(sig_zip, sd(tmp))
}

par(mfrow=c(2,3), mar=c(2.5,2.5,1,1), mgp=c(1.25,0.125,0), tck=0.01)
hist(n0_zinb, xlim=range(c(n0_zinb, zip0/n, n0_zip)), yaxt="n",
     xlab="fraction of zeros", ylab="ZINB", main="",
     border="white", col="gray", density=-1)
abline(v=zip0/n)
## mean
hist(mu_zinb, xlim=range(c(mu_zinb, mu, mu_zip)), yaxt="n",
     xlab="mean", ylab="",  main="",
     border="white", col="gray", density=-1)
abline(v=mu)
## sd
hist(sig_zinb, xlim=range(c(sig_zinb, s, sig_zip)), yaxt="n",
     xlab="standard deviation", ylab="",  main="",
     border="white", col="gray", density=-1)
abline(v=s)

hist(n0_zip, xlim=range(c(n0_zip, zip0/n, n0_zinb)), yaxt="n",
     xlab="fraction of zeros", ylab="ZIP",  main="",
     border="white", col="gray", density=-1)
abline(v=zip0/n)
## mean
hist(mu_zip, xlim=range(c(mu_zip, mu, mu_zinb)), yaxt="n",
     xlab="mean", ylab="",  main="",
     border="white", col="gray", density=-1)
abline(v=mu)
## sd
hist(sig_zip, xlim=range(c(sig_zip, s, sig_zinb)), yaxt="n",
     xlab="standard deviation", ylab="",  main="",
     border="white", col="gray", density=-1)
abline(v=s)
```



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)
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
```

```
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)

## 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

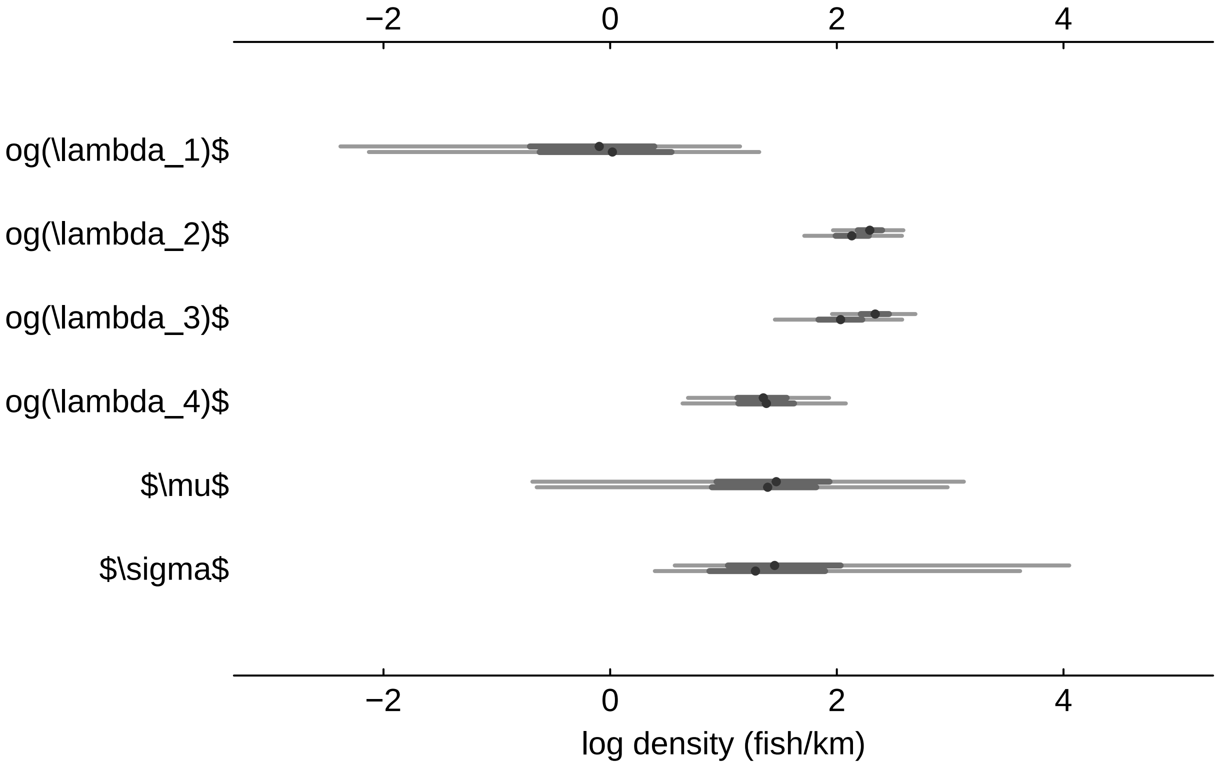
$$\begin{array}{rl} \pi(y_i \mid \lambda, p_d) = & \pi(y_i, z_i = 1 \mid \lambda, p_d) + \pi(y_i, z_i = 0 \mid \lambda, p_d) \\ = & \pi(y_i \mid z_i = 0, p_d)\pi(z_i = 0 \mid \lambda) + \\ & \pi(y_i \mid z_i = 1, p_d)\pi(z_i = 1 \mid \lambda) \\ = & \begin{cases} 0 \times e^{-\lambda d_i} + p_d \times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

```
### 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)

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:

```r
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_λ^2. The hyper-parameter μ_λ represents the log-mean of segment means, while τ_λ^2 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)
```

Now let's implement the sequential updating

```
## 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:

```
### 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]   -0.85    0.02 0.74   -2.51   -1.29   -0.78   -0.32    0.40
## log_lambda[2]    1.30    0.01 0.37    0.58    1.04    1.29    1.55    2.03
## log_lambda[3]    1.28    0.01 0.31    0.69    1.08    1.28    1.49    1.90
## log_lambda[4]    0.96    0.01 0.42    0.13    0.69    0.97    1.23    1.78
## mu               0.99    0.01 0.51   -0.05    0.69    1.01    1.31    2.00
## sigma2           1.96    0.02 1.16    0.68    1.17    1.65    2.41    5.21
## pd               0.51    0.00 0.04    0.43    0.48    0.51    0.54    0.59
## lp__          -227.54    0.04 1.91 -232.21 -228.62 -227.17 -226.14 -224.76
##               n_eff Rhat
## log_lambda[1]  2297    1
## log_lambda[2]  2753    1
## log_lambda[3]  2703    1
## log_lambda[4]  2534    1
## mu             2623    1
## sigma2         2202    1
## pd             2762    1
## lp__           2614    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 23 12:07:07 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]   -0.91    0.01 0.60   -2.21   -1.27   -0.85   -0.49    0.09
## log_lambda[2]   -0.57    0.01 0.40   -1.45   -0.82   -0.55   -0.30    0.16
## log_lambda[3]    0.02    0.01 0.27   -0.52   -0.16    0.03    0.20    0.54
## log_lambda[4]    0.18    0.01 0.33   -0.50   -0.03    0.19    0.40    0.80
## mu               0.55    0.01 0.41   -0.30    0.29    0.55    0.80    1.35
## sigma2           1.78    0.02 0.79    0.79    1.24    1.59    2.14    3.80
## pd               0.51    0.00 0.04    0.44    0.49    0.51    0.54    0.59
## lp__          -248.23    0.04 1.88 -252.77 -249.21 -247.91 -246.90 -245.54
##               n_eff Rhat
## log_lambda[1]  2600    1
## log_lambda[2]  2416    1
## log_lambda[3]  2520    1
## log_lambda[4]  2315    1
## mu             2400    1
## sigma2         2472    1
## pd             2240    1
## lp__           2472    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 23 12:07:38 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 0.56   -1.87   -1.06   -0.66   -0.32    0.28
## log_lambda[2]   -0.66    0.01 0.44   -1.63   -0.94   -0.63   -0.37    0.11
## log_lambda[3]   -1.36    0.01 0.65   -2.74   -1.76   -1.31   -0.90   -0.22
## log_lambda[4]   -1.22    0.01 0.72   -2.75   -1.65   -1.15   -0.71    0.02
## mu               0.12    0.01 0.37   -0.61   -0.13    0.14    0.37    0.82
## sigma2           1.86    0.02 0.74    0.90    1.34    1.70    2.20    3.65
## pd               0.51    0.00 0.04    0.43    0.48    0.51    0.54    0.59
## lp__          -177.17    0.04 1.91 -181.78 -178.22 -176.80 -175.79 -174.46
##               n_eff Rhat
## log_lambda[1]  2569    1
## log_lambda[2]  2237    1
## log_lambda[3]  2368    1
## log_lambda[4]  2626    1
## mu             2460    1
## sigma2         2407    1
## pd             2476    1
## lp__           2605    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 23 12:07:55 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)
```

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(λ_{js}) as a (one-dimension) vector instead of a (two-dimension) matrix. Some care must be taken in organizing input data:

```
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]      -1.09    0.03 1.33   -4.19   -1.76   -0.93   -0.25    1.12
## log_lambda[2]      -0.82    0.02 0.89   -2.81   -1.36   -0.73   -0.20    0.72
## log_lambda[3]      -1.33    0.02 1.21   -4.13   -1.92   -1.13   -0.55    0.52
## log_lambda[4]       0.78    0.03 1.28   -2.23    0.21    0.95    1.54    2.90
## log_lambda[5]       1.35    0.01 0.69    0.21    0.94    1.32    1.69    2.64
## log_lambda[6]       1.40    0.01 0.52    0.50    1.09    1.38    1.67    2.47
## log_lambda[7]       1.10    0.03 1.35   -2.10    0.48    1.18    1.77    3.76
## log_lambda[8]       1.30    0.01 0.39    0.55    1.04    1.29    1.55    2.07
## log_lambda[9]       1.34    0.01 0.50    0.46    1.03    1.31    1.61    2.29
## log_lambda[10]      0.50    0.03 1.31   -2.58   -0.13    0.63    1.22    2.70
## log_lambda[11]      0.43    0.01 0.69   -1.11    0.02    0.49    0.90    1.62
## log_lambda[12]      1.62    0.02 0.88    0.40    1.08    1.50    1.95    3.97
## mu[1]              -0.67    0.02 0.94   -2.65   -1.24   -0.65   -0.05    1.08
## mu[2]               1.16    0.01 0.72   -0.46    0.75    1.20    1.60    2.53
## mu[3]               1.21    0.01 0.69   -0.29    0.82    1.24    1.63    2.53
## mu[4]               0.88    0.02 0.76   -0.71    0.45    0.88    1.33    2.39
## m                   0.98    0.01 0.54   -0.10    0.66    0.98    1.32    2.01
## sigma2              1.43    0.04 1.87    0.02    0.25    0.68    1.84    7.44
## tau2              2.05    0.03 1.32    0.68    1.20    1.67    2.46    5.66
## pd                0.50    0.00 0.04    0.42    0.48    0.50    0.53    0.58
## lp__           -233.04    0.08 3.62 -240.83 -235.26 -232.76 -230.50 -226.67
##                 n_eff Rhat
## log_lambda[1]    2557    1
## log_lambda[2]    2449    1
## log_lambda[3]    2580    1
## log_lambda[4]    2417    1
## log_lambda[5]    2424    1
## log_lambda[6]    2438    1
## log_lambda[7]    2639    1
## log_lambda[8]    2567    1
## log_lambda[9]    2578    1
## log_lambda[10]   2418    1
## log_lambda[11]   2404    1
## log_lambda[12]   2507    1
## mu[1]            2611    1
## mu[2]            2383    1
## mu[3]            2397    1
## mu[4]            2503    1
## m                2577    1
## sigma2           2154    1
## tau2             2396    1
## pd               2482    1
## lp__             2283    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 23 12:09:17 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 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]     -1.73    0.03 1.26   -4.70   -2.42   -1.58   -0.83    0.23
## log_lambda[2]     -1.86    0.03 1.26   -4.89   -2.53   -1.69   -1.01    0.10
## log_lambda[3]     -0.56    0.02 0.79   -2.24   -1.05   -0.53   -0.03    0.85
## log_lambda[4]     -0.46    0.01 0.70   -1.99   -0.90   -0.43    0.02    0.85
## log_lambda[5]     -1.98    0.02 1.14   -4.64   -2.55   -1.80   -1.21   -0.27
## log_lambda[6]     -0.27    0.01 0.55   -1.40   -0.62   -0.25    0.11    0.72
## log_lambda[7]      0.94    0.01 0.44    0.12    0.65    0.93    1.21    1.83
## log_lambda[8]    -0.82    0.01 0.55   -2.04   -1.16   -0.77   -0.44    0.14
## log_lambda[9]    -0.35    0.01 0.50   -1.44   -0.67   -0.32    0.02    0.50
## log_lambda[10]    0.90    0.01 0.56   -0.09    0.54    0.89    1.23    2.04
## log_lambda[11]   -0.93    0.02 0.81   -2.78   -1.40   -0.84   -0.39    0.40
## log_lambda[12]   -0.18    0.01 0.58   -1.47   -0.52   -0.13    0.23    0.83
## mu[1]            -0.86    0.02 0.85   -2.60   -1.40   -0.83   -0.30    0.69
## mu[2]            -0.58    0.01 0.75   -2.09   -1.06   -0.58   -0.10    0.86
## mu[3]            -0.10    0.01 0.67   -1.46   -0.52   -0.10    0.32    1.17
## mu[4]            -0.05    0.01 0.68   -1.46   -0.48   -0.05    0.39    1.28
## m                -0.17    0.01 0.58   -1.44   -0.52   -0.14    0.20    0.96
## sigma2            2.07    0.04 1.78    0.28    0.85    1.49    2.71    7.38
## tau2              1.07    0.02 0.77    0.33    0.61    0.84    1.29    3.03
## pd                0.51    0.00 0.04    0.43    0.49    0.51    0.54    0.59
## lp__           -240.42    0.08 3.90 -248.79 -242.81 -240.09 -237.67 -233.75
##                n_eff Rhat
## log_lambda[1]   2545    1
## log_lambda[2]   2397    1
## log_lambda[3]   2679    1
## log_lambda[4]   2493    1
## log_lambda[5]   2719    1
## log_lambda[6]   2475    1
## log_lambda[7]   2218    1
## log_lambda[8]   2644    1
## log_lambda[9]   2509    1
## log_lambda[10]  2280    1
## log_lambda[11]  2621    1
## log_lambda[12]  2358    1
## mu[1]           2531    1
## mu[2]           2635    1
## mu[3]           2191    1
## mu[4]           2306    1
## m               2577    1
## sigma2          2429    1
## tau2            2428    1
## pd              2588    1
## lp__            2541    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 23 12:10:12 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)
```

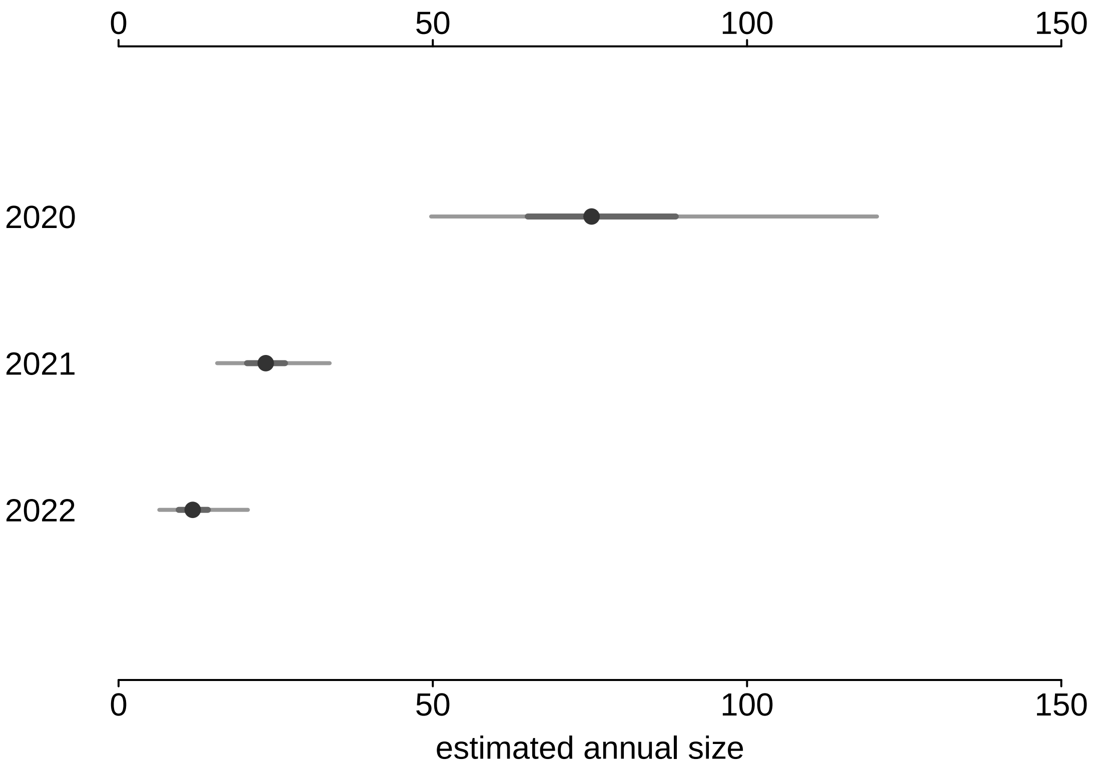
```
## Inference for Stan model: anon_model.
## 8 chains, each with iter=5000; warmup=2500; thin=8;
## post-warmup draws per chain=313, total post-warmup draws=2504.
##
##                   mean se_mean   sd    2.5%     25%     50%     75%   97.5%
## log_lambda[1]    -0.38    0.02 0.79   -1.91   -0.90   -0.37    0.15    1.17
## log_lambda[2]    -1.77    0.02 1.15   -4.74   -2.30   -1.55   -0.99   -0.13
## log_lambda[3]    -0.67    0.02 0.91   -2.59   -1.22   -0.64   -0.08    1.06
## log_lambda[4]    -0.62    0.01 0.70   -2.08   -1.07   -0.60   -0.14    0.67
## log_lambda[5]    -0.99    0.01 0.63   -2.38   -1.36   -0.93   -0.55    0.08
## log_lambda[6]    -1.04    0.02 0.81   -2.88   -1.53   -0.95   -0.47    0.31
## log_lambda[7]    -1.34    0.02 0.90   -3.28   -1.89   -1.28   -0.72    0.31
## log_lambda[8]    -1.80    0.02 0.86   -3.68   -2.31   -1.70   -1.21   -0.35
## log_lambda[9]    -1.83    0.02 1.21   -4.60   -2.47   -1.67   -1.03    0.17
## log_lambda[10]   -1.93    0.02 1.25   -4.83   -2.61   -1.77   -1.07    0.05
## log_lambda[11]   -1.53    0.02 0.90   -3.51   -2.07   -1.45   -0.88   -0.02
## log_lambda[12]   -1.79    0.03 1.29   -4.83   -2.48   -1.67   -0.93    0.42
## mu[1]            -0.90    0.02 0.76   -2.56   -1.36   -0.87   -0.41    0.55
## mu[2]            -0.87    0.01 0.70   -2.26   -1.31   -0.87   -0.43    0.46
## mu[3]            -1.46    0.02 0.81   -3.16   -2.00   -1.43   -0.91    0.04
## mu[4]            -1.54    0.02 0.89   -3.48   -2.09   -1.51   -0.96    0.14
## m                -0.86    0.01 0.54   -1.97   -1.19   -0.84   -0.52    0.13
## sigma2            1.37    0.03 1.58    0.03    0.31    0.80    1.83    6.02
## tau2              1.55    0.02 0.89    0.58    0.96    1.32    1.87    3.96
## pd                0.52    0.00 0.04    0.44    0.49    0.52    0.54    0.59
## lp__           -175.67    0.07 3.62 -183.54 -177.95 -175.35 -173.12 -169.28
##                n_eff Rhat
## log_lambda[1]   2664    1
## log_lambda[2]   2526    1
## log_lambda[3]   2608    1
## log_lambda[4]   2242    1
## log_lambda[5]   2522    1
## log_lambda[6]   2403    1
## log_lambda[7]   2403    1
## log_lambda[8]   2520    1
## log_lambda[9]   2553    1
## log_lambda[10]  2497    1
## log_lambda[11]  2436    1
## log_lambda[12]  2532    1
## mu[1]           2389    1
## mu[2]           2426    1
## mu[3]           2186    1
## mu[4]           2597    1
## m               2465    1
## sigma2          2516    1
## tau2            2492    1
## pd              2407    1
## lp__            2533    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 23 12:10:39 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.

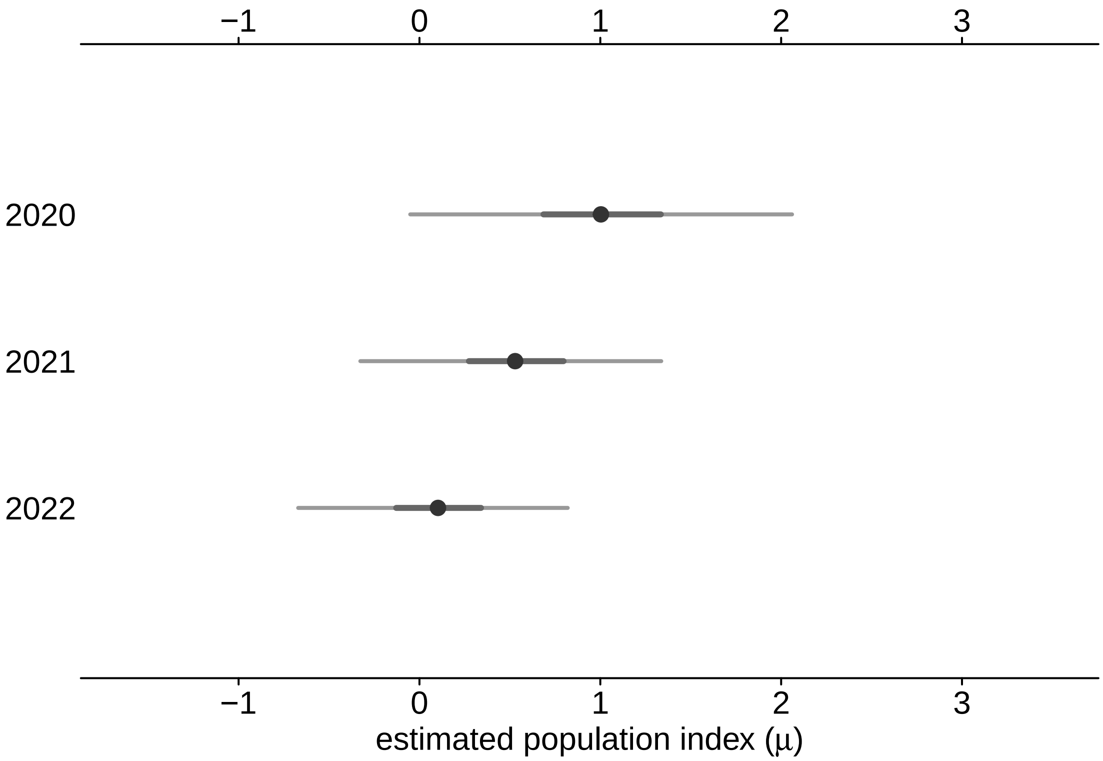
```
load("seqUpdate2020_2022.RData")
coef2020seg <- extract(fit2keepSeg_2020, pars=c("log_lambda","mu"))
segCoefrv20 <- rvsims(as.matrix(as.data.frame(coef2020seg)))
coef2020sn <- extract(fit2keepSn_2020, pars=c("log_lambda","mu"))
snCoefrv20 <- rvsims(as.matrix(as.data.frame(coef2020sn)))
coef2020nested <- 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))
```

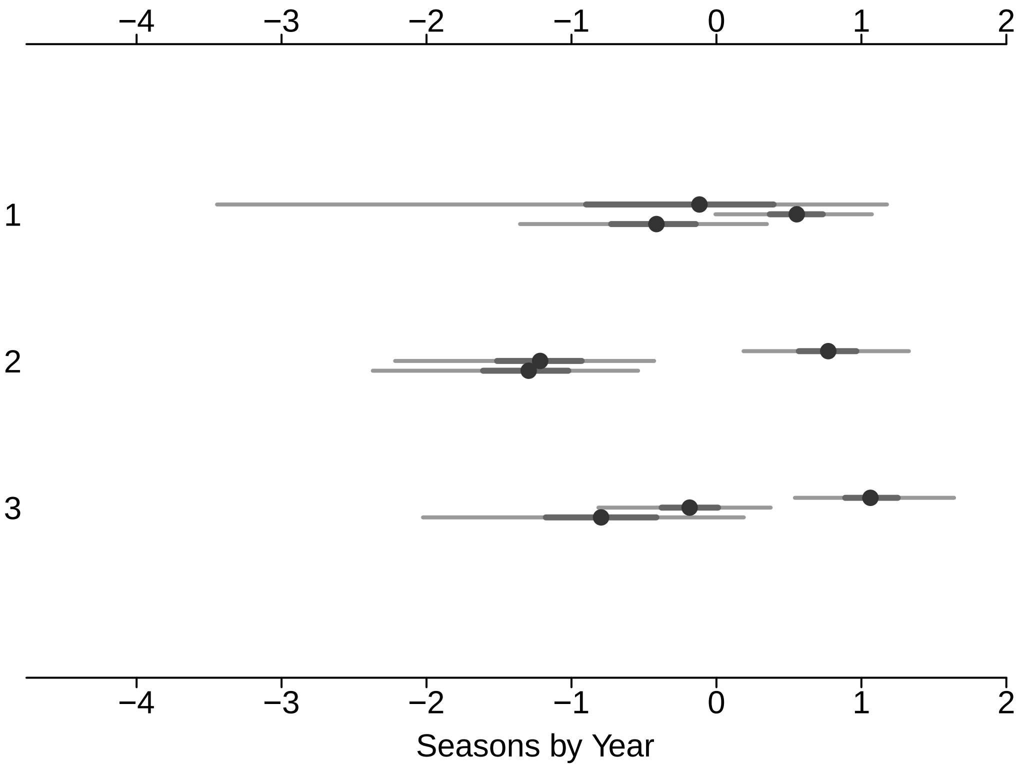


```
par(mar=c(3, 2, 1, 1), mgp=c(1.25,0.125,0), tck=-0.01)
mlplot(musSeg, xlab=expression(paste("estimated population index (",mu, ")", sep="")))
```

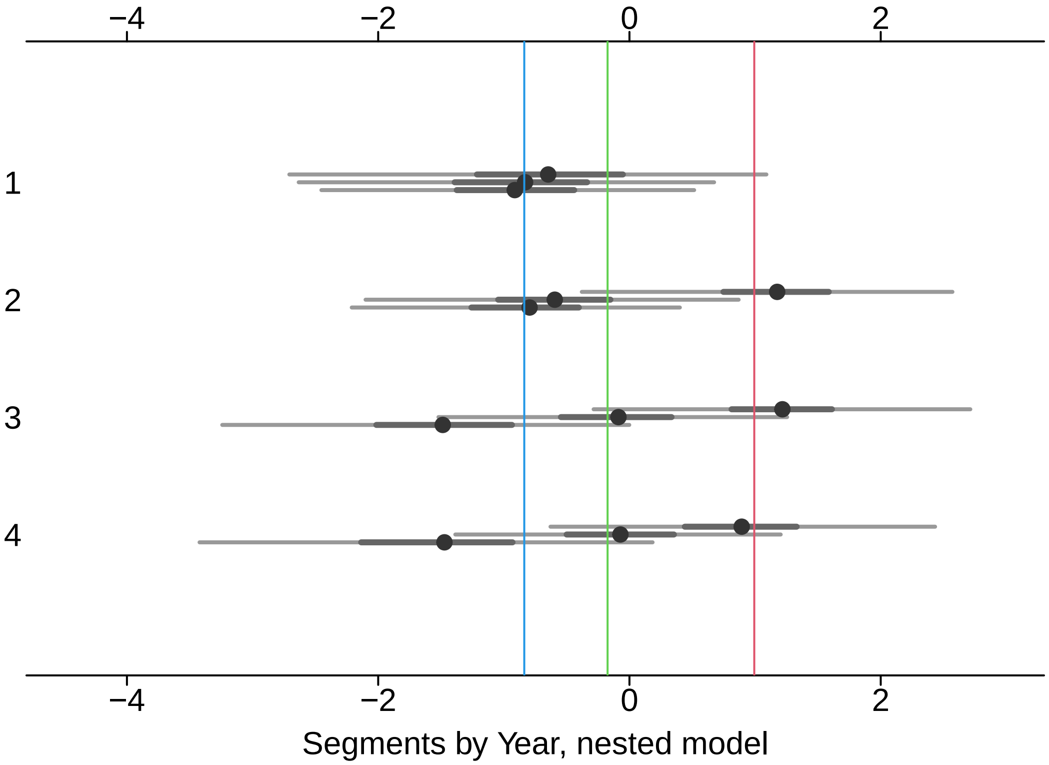


```
### end of annual numbers

logLambdaSn <- rvmatrix(c(snCoefrv20[1:3], snCoefrv21[1:3], snCoefrv22[1:3]), ncol=3)
```

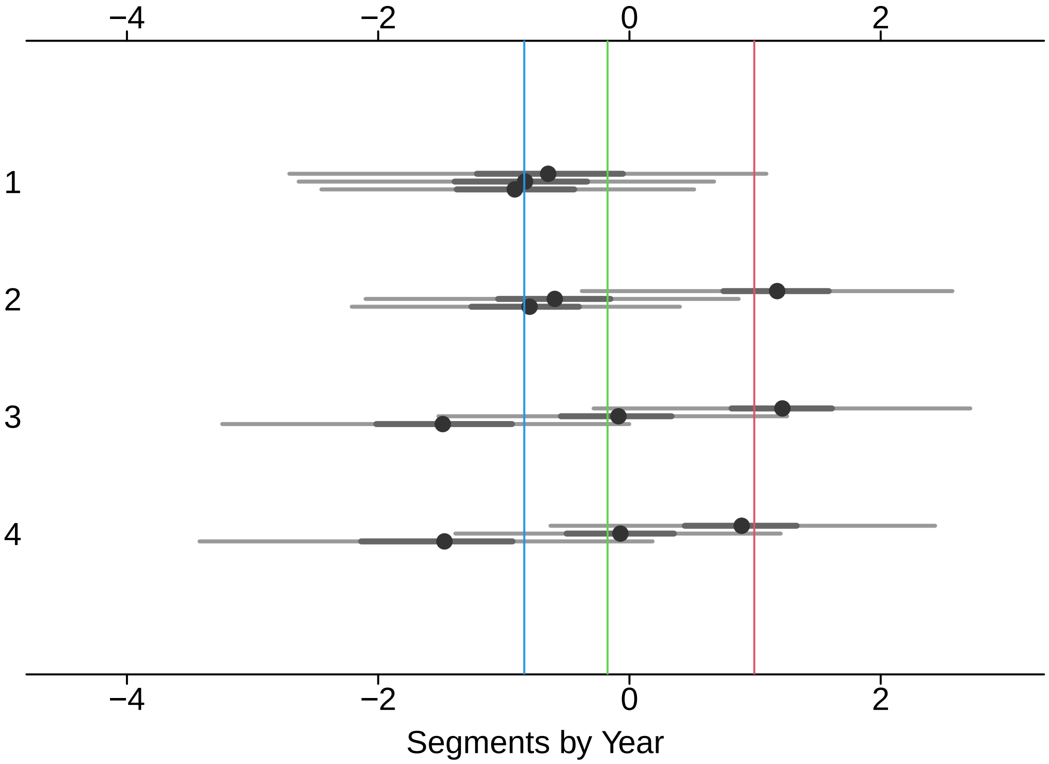
```
musSn <- c(snCoefrv20[4], snCoefrv21[4], snCoefrv22[4])
par(mar=c(3, 1, 1, 2), mgp=c(1.25,0.125,0), tck=-0.015)
mlplot(logLambdaSn, xlab="Seasons by Year", xlim=c(-4.5, 2))
```



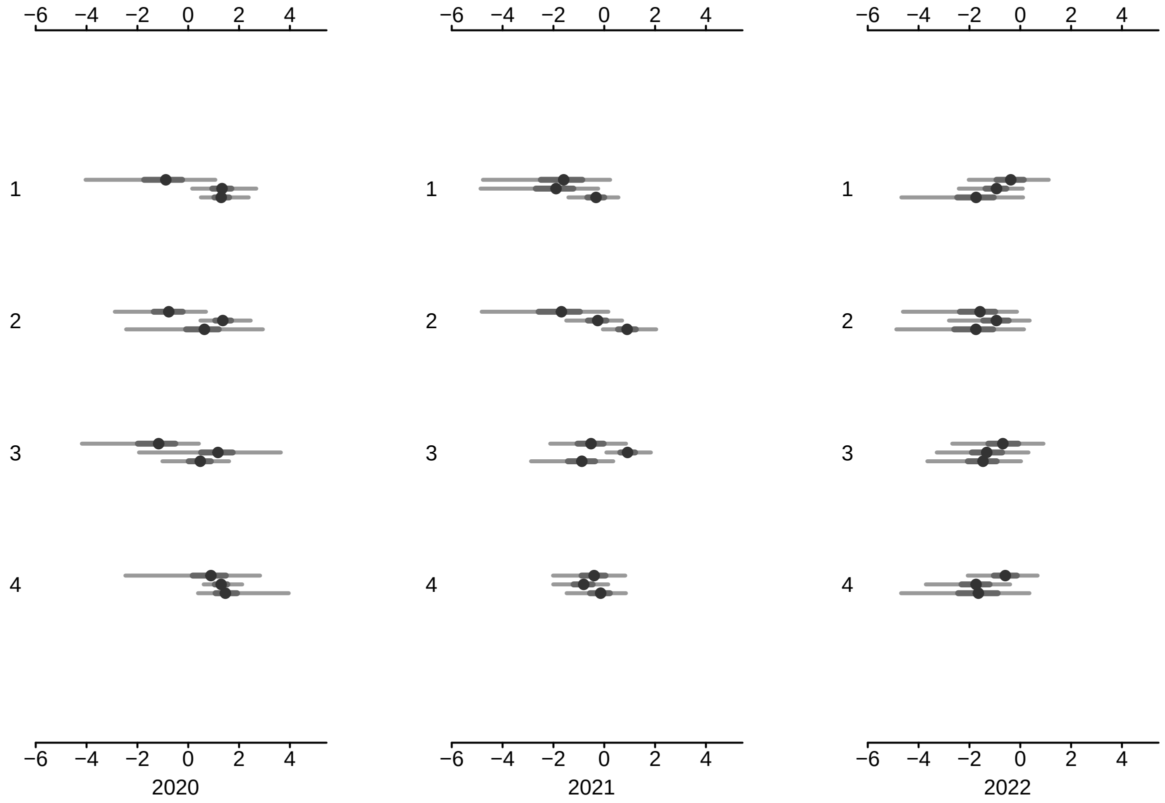
```
##abline(v=summary(musSn)$mean, lty=1:3)
##legend(x="topleft",lty=1:3, legend=c("2020","2020-2021","2020-2022"), bty="n")

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)
```



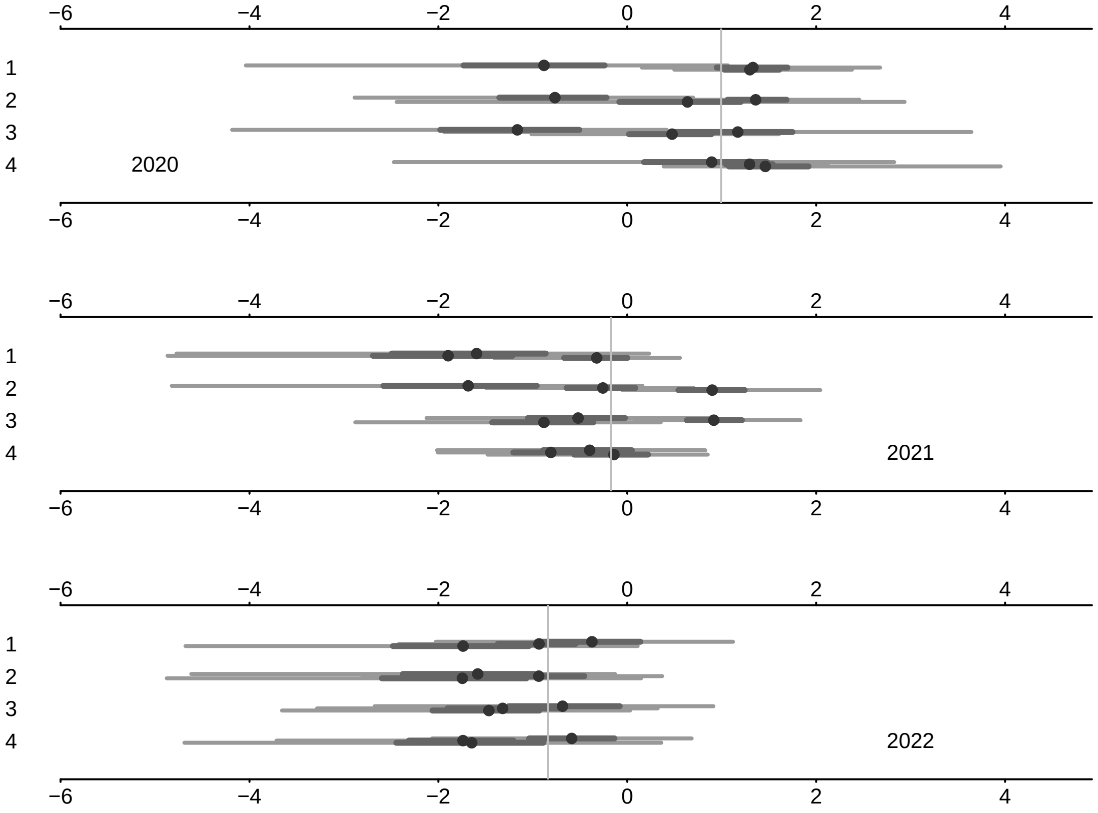
```
par(mar=c(3, 1, 1, 2), mgp=c(1.25,0.125,0), tck=-0.015)
mlplot(musNst, xlab="Segments by Year", xlim=c(-4.5, 3))
abline(v=summary(mNst)$mean, col=2:4)
```



```
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))
```



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

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
2021
2022