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 learn the use of Bayesian inference with examples often seen in the ecological and environmental literature, where the response variable distribution models are standard distributions covered in statistical textbooks. We emphasize the flexibility of the Bayesian approach and explore how such models can often be expanded to make them more realistic (compared to how the data were generated). The guiding principle of Bayesian applications in environmental and ecological statistics should be the three criteria of an applied statistical model recommended by Cox (1995): - A probability distribution model for the response variable, - A parameter vector that defines the distribution model, at least some of the parameters should reflect features of the system under study - The inclusion or representation of the data-generating process.

The last criterion (data generating process) is most important as we want to model we developed to be relevant to the problem at hand. In other words, we want to model to reflect the likely causal relationship, rather than simply correlation. Furthermore, environmental and ecological data are often observational. That is, we collect the data without knowing what would be the proper model (method) to analyzed them. Statistics we learned from graduate school is mostly based on the assumption that we know the right model. We teach statistics chapter by chapter, each time assuming that we know the right model. When we actually know what assumptions we want to test, we design experiments to isolate the main factor of interest and using randomization to neutralize other factors that may also affect the response variable (confounding factors). When we are working on data from typical environmental and ecological monitoring programs, we often started collecting data without knowing what questions we want to answer. For example, most data we used for studying the effects of climate change. In my opinion, analyzing environmental/ecological data often starts with identifying what is the problem to be answered. As all models are wrong (and some are useful), we are interested in identifying how and when a wrong model is useful.

I find the following process is effective in analyzing environmental/ecological data: - Starting from existing (simple) models that can be directly implemented in R. - Explore the model fit and identify its weakness, using the "posterior simulation" concept – use the fitted model to predict what we want to learn and what we already know. - Based on the identified weakness, reformulate the model to redress the weakness. - Implement the model in Stan to further identify problems, such as computational stability and identifiability (e.g., the snake fungal disease example with unknown error rates). - Repeat the posterior simulation to identify where and when the model is useful.

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.

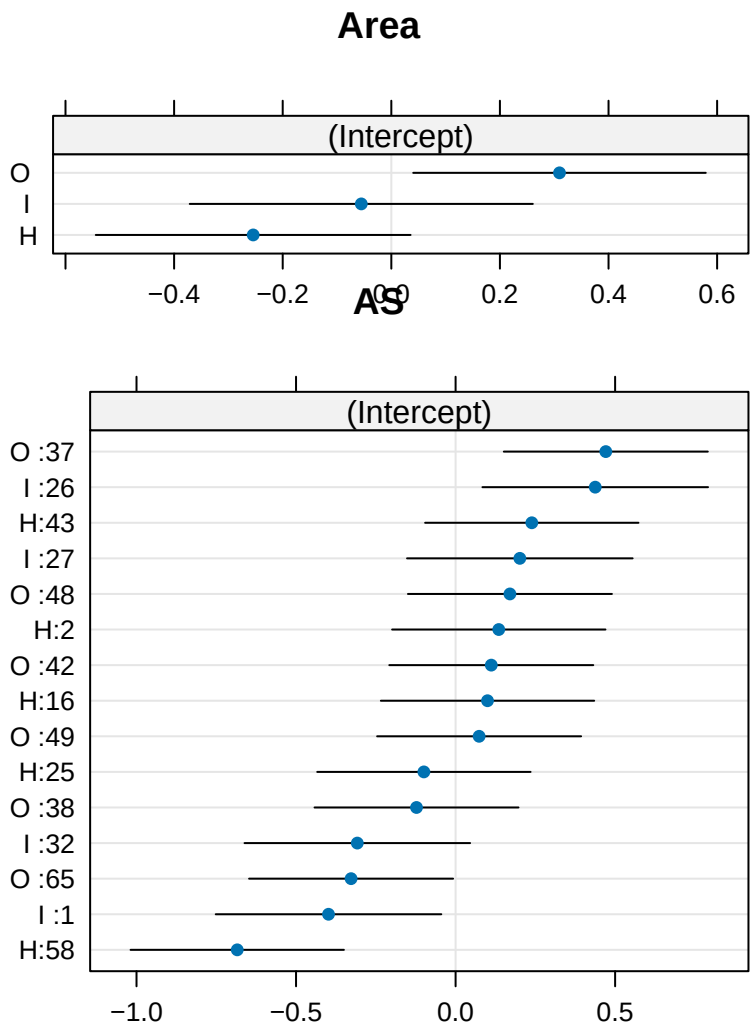
```
benthic.data$AS <- with(benthic.data, factor(paste(Area, Station, sep=":")))
benthicAb.Lme1 <- lmer(log(Abundance) ~ 1+(1|AS)+(1|Area), data=benthic.data)
benthicRn.Lme1 <- lmer(log(Richness) ~ 1+(1|AS)+(1|Area), data=benthic.data)
summary(benthicAb.Lme1)
```

```
## 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
```

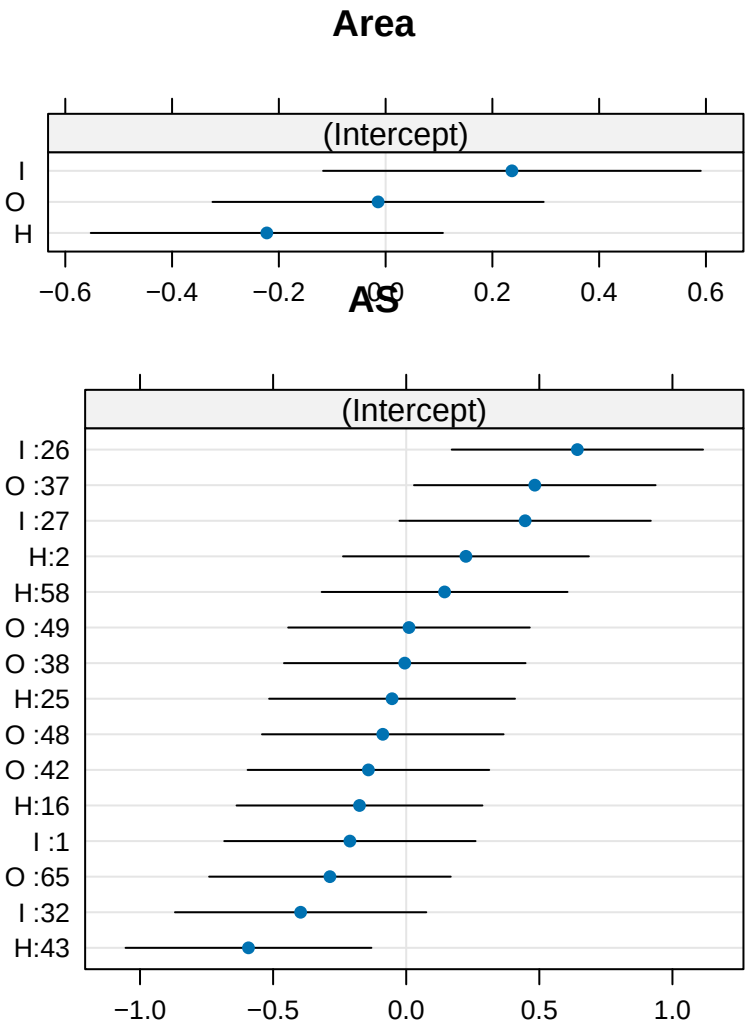
```
summary(benthicRn.Lme1)
```

```
## 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)
```

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

```
## Running /usr/lib/R/bin/R CMD SHLIB foo.c
## gcc -I"/usr/share/R/include" -DNDEBUG   -I"/usr/lib/R/site-library/Rcpp/include/"  -I"/usr/lib/R/sit
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:88,
##                  from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/lib/R/site-library/StanHeaders/include/stan/math/prim/mat/fun/Eigen.hpp:13
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:1: error: unknown type nam
##   628 | namespace Eigen {
##       | ^~~~~~~~
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:17: error: expected '=',
##   628 | namespace Eigen {
##       |                 ^
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/lib/R/site-library/StanHeaders/include/stan/math/prim/mat/fun/Eigen.hpp:13
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:96:10: fatal error: complex: No such file or dir
##    96 | #include <complex>
##       |          ^~~~~~~~
## compilation terminated.
## make: *** [/usr/lib/R/etc/Makeconf:169: foo.o] Error 1
```

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 20 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: 17bbf0791cbacc5326228fcfea998c45.
## 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  2465 1.00
## mu[2]       2.72    0.00 0.13  2.46  2.64  2.72  2.80  3.00  2480 1.00
## mu[3]       2.69    0.00 0.14  2.43  2.60  2.69  2.77  2.96  2777 1.00
## mu[4]       2.50    0.00 0.13  2.25  2.42  2.50  2.58  2.77  2416 1.00
## mu[5]       3.24    0.00 0.09  3.05  3.19  3.25  3.30  3.41  2414 1.00
## mu[6]       3.00    0.00 0.09  2.81  2.95  3.00  3.05  3.16  2042 1.01
## mu[7]       2.47    0.00 0.09  2.30  2.42  2.47  2.52  2.67  2366 1.00
```

```
## mu[8]       3.61    0.00 0.14  3.32  3.53  3.62  3.70  3.87  2294 1.00
## mu[9]       3.04    0.00 0.13  2.78  2.96  3.04  3.12  3.32  2342 1.00
## mu[10]      3.26    0.00 0.13  3.00  3.18  3.26  3.34  3.52  1992 1.00
## mu[11]      2.82    0.00 0.13  2.55  2.74  2.83  2.91  3.08  2532 1.00
## mu[12]      3.31    0.00 0.13  3.05  3.23  3.32  3.40  3.57  2495 1.00
## mu[13]      3.22    0.00 0.13  2.97  3.14  3.23  3.30  3.47  2296 1.00
## mu[14]      1.93    0.00 0.14  1.67  1.84  1.92  2.02  2.24  2083 1.00
## mu[15]      2.84    0.00 0.13  2.59  2.75  2.83  2.92  3.11  2254 1.00
## theta[1]    2.79    0.00 0.19  2.42  2.68  2.79  2.91  3.17  2092 1.00
## theta[2]    2.60    0.00 0.19  2.24  2.48  2.60  2.74  2.99  2548 1.00
## theta[3]    3.14    0.00 0.18  2.76  3.02  3.14  3.26  3.47  2261 1.00
## mu_hyp      2.85    0.01 0.40  1.99  2.66  2.84  3.04  3.71  2256 1.00
## sigma_y[1]  0.15    0.00 0.05  0.08  0.11  0.14  0.17  0.27  2288 1.00
## sigma_y[2]  0.23    0.00 0.06  0.15  0.19  0.22  0.27  0.39  2478 1.00
## sigma_y[3]  0.23    0.00 0.05  0.15  0.19  0.22  0.26  0.37  2218 1.00
## sigma_i     0.40    0.00 0.11  0.25  0.33  0.39  0.46  0.65  1802 1.00
## sigma_hyp   0.54    0.01 0.38  0.07  0.27  0.44  0.71  1.55  2249 1.00
## delta1     -0.19    0.01 0.25 -0.71 -0.35 -0.18 -0.02  0.28  2448 1.00
## delta2     -0.53    0.01 0.28 -1.05 -0.72 -0.54 -0.35  0.01  2134 1.00
## delta3     -0.34    0.01 0.25 -0.85 -0.51 -0.34 -0.18  0.14  1863 1.00
## lp__       52.82    0.09 4.40 43.29 49.99 53.23 56.03 60.07  2337 1.00
##
## Samples were drawn using NUTS(diag_e) at Tue May 16 13:52:44 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 60 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: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may be
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#bulk-ess

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

```
print(fit2keep)
```

```
## Inference for Stan model: 17bbf0791cbacc5326228fcfea998c45.
## 8 chains, each with iter=5000; warmup=2500; thin=8;
## post-warmup draws per chain=313, total post-warmup draws=2504.
```

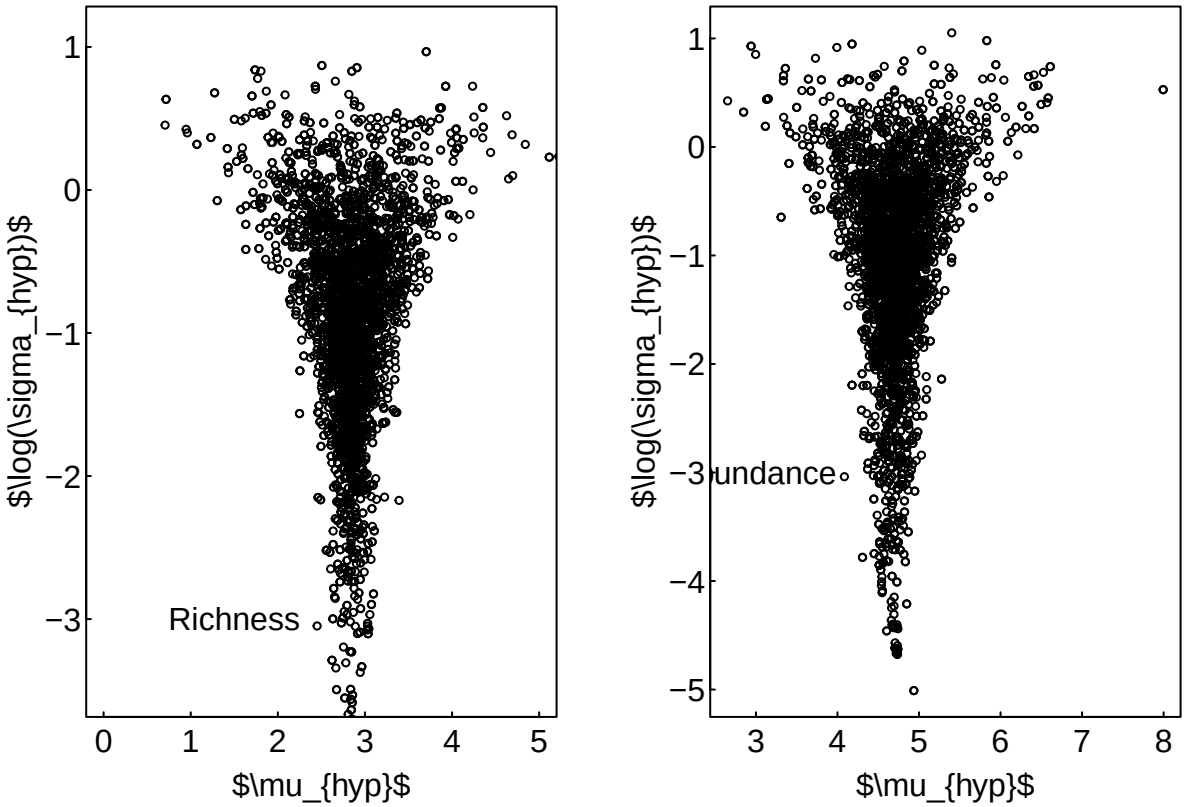
```
##
##             mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## mu[1]       4.76    0.01 0.26  4.30  4.60  4.75  4.92  5.29  1895 1.00
## mu[2]       4.67    0.01 0.24  4.19  4.52  4.68  4.84  5.13  1876 1.00
## mu[3]       4.33    0.01 0.24  3.84  4.18  4.33  4.49  4.80  1309 1.01
## mu[4]       4.43    0.01 0.24  3.96  4.28  4.43  4.59  4.91  2310 1.00
## mu[5]       5.48    0.01 0.31  4.80  5.30  5.50  5.69  6.03  2328 1.00
## mu[6]       5.32    0.01 0.27  4.73  5.15  5.34  5.51  5.82  2403 1.00
## mu[7]       4.61    0.01 0.29  4.09  4.41  4.59  4.79  5.20  1014 1.01
## mu[8]       5.16    0.00 0.22  4.71  5.02  5.16  5.30  5.55  2314 1.00
## mu[9]       4.68    0.00 0.19  4.29  4.56  4.69  4.81  5.07  2462 1.00
## mu[10]      4.56    0.01 0.20  4.16  4.42  4.55  4.69  4.97  1345 1.01
## mu[11]      3.97    0.01 0.28  3.43  3.78  3.96  4.14  4.53  2447 1.00
## mu[12]      4.60    0.00 0.19  4.21  4.48  4.60  4.73  4.99  2569 1.00
## mu[13]      4.70    0.00 0.19  4.31  4.58  4.70  4.82  5.09  2466 1.00
## mu[14]      4.61    0.00 0.25  4.12  4.45  4.62  4.77  5.09  2480 1.00
## mu[15]      4.41    0.00 0.20  4.00  4.28  4.41  4.55  4.82  2082 1.01
## theta[1]    4.94    0.01 0.26  4.48  4.76  4.93  5.12  5.49  1686 1.00
## theta[2]    4.48    0.00 0.22  4.04  4.34  4.49  4.63  4.91  2071 1.00
## theta[3]    4.69    0.00 0.18  4.32  4.58  4.69  4.80  5.04  2644 1.00
## mu_hyp      4.71    0.01 0.39  3.96  4.53  4.71  4.88  5.51  2524 1.00
## sigma_y[1]  0.53    0.00 0.18  0.29  0.40  0.48  0.60  0.99  2323 1.00
## sigma_y[2]  0.50    0.00 0.13  0.32  0.41  0.48  0.56  0.82   774 1.01
## sigma_y[3]  0.38    0.00 0.08  0.26  0.32  0.37  0.43  0.58  1335 1.01
## sigma_i     0.43    0.00 0.15  0.17  0.33  0.41  0.51  0.76  2211 1.00
## sigma_hyp   0.49    0.01 0.39  0.03  0.22  0.39  0.66  1.50  1740 1.00
## delta1     -0.46    0.01 0.35 -1.16 -0.70 -0.45 -0.19  0.08  1259 1.00
## delta2     -0.20    0.01 0.26 -0.73 -0.38 -0.19 -0.02  0.27  2386 1.00
## delta3      0.26    0.01 0.30 -0.27  0.04  0.23  0.45  0.90  1959 1.00
## lp__       20.40    0.10 4.48 10.64 17.71 20.71 23.55 27.98  2195 1.00
##
## Samples were drawn using NUTS(diag_e) at Tue May 16 13:52:47 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")
```



What we encountered is the famous Neal's funnel, indicating the lack of information to properly quantify both μ_{hyp} and σ^2_{hyp} together. This feature often leads to highly correlated θ_k. We need informative prior on one of the two parameters. In this case, I used a strong prior for σ^2_{hyp} (a half normal N(0,1)). When using a noninformative prior, the program runs very slowly. Computationally, we can avoid the computational difficulties by reparameterizing θ_k. Instead of modeling it as a normal random variable with parameters μ_{hyp} and σ^2_{hyp}, we introduce a new parameter $z_k \sim N(0,1)$ and model θ_k as a transformed variable: $\theta_k = \mu_{hyp} + \sigma_{hyp} z_k$. As a result, we no longer directly sample θ_k. It is an effective way to reduce the number of divergent transitions. But the model stays the same and the Neal's funnel remains.

The computational issues of this model made us to look at this problem more. We realized that the data we have do not have enough information to simultaneously quantify θ_k, μ_{hyp}, and σ^2_{hyp}, largely due to the large within zone (among core) variation. The multilevel model suggested that including zone as a random effect is not effective in explaining the total variance. As such, we proposed two alternative models without directly parameterizing the among zone variation.

In the first alternative we removed zone as a hierarchy and model the core means as exchangeable:

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

We then estimate each zone mean as the average of the means of cores inside the respective zone. The response variable is standardized to allow the use of relatively constrained priors for variance parameters (e.g., half normal N(0,0.5)).

```
###
### 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)
```

```
## Trying to compile a simple C file
## Running /usr/lib/R/bin/R CMD SHLIB foo.c
## gcc -I"/usr/share/R/include" -DNDEBUG   -I"/usr/lib/R/site-library/Rcpp/include/"  -I"/usr/lib/R/sit
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:88,
##                  from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/lib/R/site-library/StanHeaders/include/stan/math/prim/mat/fun/Eigen.hpp:13
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:1: error: unknown type nam
##   628 | namespace Eigen {
##       | ^~~~~~~~~
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:17: error: expected '=',
##   628 | namespace Eigen {
##       |                 ^
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/lib/R/site-library/StanHeaders/include/stan/math/prim/mat/fun/Eigen.hpp:13
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:96:10: fatal error: complex: No such file or di
##    96 | #include <complex>
##       |          ^~~~~~~~~
## compilation terminated.
## make: *** [/usr/lib/R/etc/Makeconf:169: foo.o] Error 1
```

```
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: 84753f0cdb4a2fa08760f40500713c4d.
## 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 -1.44 -1.15 -1.00 -0.84 -0.55  2414    1
## mu[2]     -0.25    0.00 0.23 -0.69 -0.40 -0.25 -0.10  0.22  2604    1
## mu[3]     -0.34    0.00 0.23 -0.78 -0.49 -0.34 -0.19  0.13  2704    1
## mu[4]     -0.75    0.00 0.23 -1.18 -0.90 -0.75 -0.59 -0.29  2517    1
## mu[5]      0.77    0.00 0.23  0.30  0.62  0.77  0.93  1.21  2628    1
## mu[6]      0.27    0.00 0.22 -0.18  0.12  0.27  0.42  0.70  2325    1
## mu[7]     -0.81    0.00 0.23 -1.25 -0.96 -0.81 -0.66 -0.36  2601    1
## mu[8]      1.54    0.00 0.23  1.09  1.38  1.54  1.69  1.97  2691    1
## mu[9]      0.29    0.00 0.23 -0.17  0.13  0.28  0.44  0.76  2563    1
## mu[10]     0.78    0.00 0.23  0.33  0.63  0.78  0.93  1.22  2520    1
## mu[11]    -0.03    0.00 0.23 -0.49 -0.18 -0.04  0.12  0.41  2443    1
## mu[12]     0.91    0.00 0.22  0.47  0.76  0.91  1.05  1.34  2466    1
## mu[13]     0.71    0.00 0.23  0.26  0.56  0.71  0.86  1.17  2496    1
## mu[14]    -1.97    0.00 0.24 -2.43 -2.13 -1.98 -1.82 -1.49  2483    1
## mu[15]    -0.14    0.00 0.22 -0.59 -0.29 -0.14  0.00  0.28  2308    1
## mu_hyp     0.00    0.00 0.24 -0.47 -0.15  0.00  0.16  0.47  2477    1
## sigma_y    0.40    0.00 0.06  0.31  0.37  0.40  0.43  0.54  2402    1
## sigma_hyp  0.90    0.00 0.16  0.64  0.78  0.88  0.99  1.25  2456    1
## lp__       9.11    0.07 3.58  1.01  7.00  9.44 11.71 14.98  2419    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 16 13:53:04 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_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)
print(fit2keep)
```

```
## Inference for Stan model: 84753f0cdb4a2fa08760f40500713c4d.
## 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  -0.70  -0.26 -0.04  0.20  0.65  2403    1
## mu[2]       0.11    0.01 0.35  -0.57  -0.11  0.12  0.34  0.78  2383    1
## mu[3]      -0.55    0.01 0.35  -1.24  -0.78 -0.55 -0.32  0.13  2428    1
## mu[4]      -0.36    0.01 0.34  -1.03  -0.58 -0.36 -0.12  0.30  2369    1
## mu[5]       1.39    0.01 0.39   0.59   1.14  1.39  1.65  2.13  2333    1
## mu[6]       1.07    0.01 0.36   0.36   0.84  1.07  1.32  1.79  2516    1
## mu[7]      -0.33    0.01 0.34  -1.00  -0.56 -0.33 -0.11  0.37  2662    1
## mu[8]       0.82    0.01 0.36   0.09   0.59  0.81  1.06  1.51  2434    1
## mu[9]       0.00    0.01 0.34  -0.67  -0.21  0.00  0.22  0.67  2548    1
## mu[10]     -0.24    0.01 0.34  -0.89  -0.46 -0.23 -0.01  0.43  2517    1
## mu[11]     -1.26    0.01 0.38  -1.99  -1.52 -1.27 -1.01 -0.51  2289    1
## mu[12]     -0.14    0.01 0.34  -0.81  -0.36 -0.14  0.07  0.57  2267    1
## mu[13]      0.02    0.01 0.33  -0.62  -0.19  0.02  0.24  0.69  2543    1
```

```
## mu[14]     -0.02    0.01 0.34  -0.68  -0.24 -0.02  0.19  0.64  2273    1
## mu[15]     -0.48    0.01 0.34  -1.17  -0.71 -0.48 -0.25  0.18  2506    1
## mu_hyp      0.00    0.00 0.22  -0.44  -0.14  0.00  0.13  0.42  2277    1
## sigma_y     0.67    0.00 0.09   0.52   0.61  0.66  0.72  0.87  2256    1
## sigma_hyp   0.74    0.00 0.16   0.45   0.63  0.72  0.83  1.09  2136    1
## lp__      -10.24    0.07 3.50 -18.30 -12.27 -9.89 -7.71 -4.72  2262    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 16 13:53:05 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).
```

```
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
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:

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

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

```
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)
```

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

```
## Running /usr/lib/R/bin/R CMD SHLIB foo.c
## gcc -I"/usr/share/R/include" -DNDEBUG   -I"/usr/lib/R/site-library/Rcpp/include/"  -I"/usr/lib/R/sit
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:88,
##                  from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/lib/R/site-library/StanHeaders/include/stan/math/prim/mat/fun/Eigen.hpp:13
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:1: error: unknown type nam
##   628 | namespace Eigen {
##       | ^~~~~~~~~
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:17: error: expected '=',
##   628 | namespace Eigen {
##       |                 ^
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/lib/R/site-library/StanHeaders/include/stan/math/prim/mat/fun/Eigen.hpp:13
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:96:10: fatal error: complex: No such file or di
##    96 | #include <complex>
##       |          ^~~~~~~~~
## compilation terminated.
```

```
## make: *** [/usr/lib/R/etc/Makeconf:169: foo.o] Error 1
```

```
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 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: c7e67690f2e5d8b232a67f2588c13147.
## 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     -0.04    0.01 0.39 -0.87 -0.27 -0.04  0.20  0.75  2315    1
## muK[1]  -0.84    0.01 0.38 -1.60 -1.08 -0.85 -0.59 -0.10  2207    1
## muK[2]   0.18    0.01 0.38 -0.58 -0.07  0.19  0.43  0.93  2211    1
## muK[3]   0.11    0.01 0.38 -0.68 -0.14  0.12  0.36  0.87  2073    1
## muK[4]  -0.30    0.01 0.38 -1.08 -0.55 -0.29 -0.05  0.45  2167    1
## muK[5]   0.86    0.01 0.38  0.12  0.60  0.85  1.10  1.63  2028    1
## muK[6]   0.38    0.01 0.38 -0.37  0.13  0.38  0.62  1.15  2372    1
## muK[7]  -0.66    0.01 0.37 -1.36 -0.90 -0.66 -0.41  0.09  2274    1
```

```
## muK[8]   1.06    0.01 0.39  0.33  0.80  1.06  1.31  1.84  2148    1
## muK[9]  -0.14    0.01 0.38 -0.87 -0.40 -0.15  0.12  0.66  2268    1
## muK[10]  0.33    0.01 0.38 -0.39  0.08  0.32  0.57  1.10  2109    1
## muK[11]  0.38    0.01 0.37 -0.37  0.15  0.39  0.62  1.12  2294    1
## muK[12]  0.46    0.01 0.38 -0.24  0.20  0.45  0.71  1.27  2112    1
## muK[13]  0.26    0.01 0.38 -0.45 -0.01  0.25  0.51  1.03  2252    1
## muK[14] -1.50    0.01 0.40 -2.29 -1.76 -1.49 -1.23 -0.75  2118    1
## muK[15] -0.55    0.01 0.37 -1.26 -0.80 -0.56 -0.31  0.19  2223    1
## muZ[1]  -0.09    0.01 0.43 -1.02 -0.32 -0.06  0.16  0.77  2512    1
## muZ[2]  -0.43    0.01 0.45 -1.47 -0.67 -0.39 -0.13  0.38  2258    1
## muZ[3]   0.51    0.01 0.44 -0.27  0.21  0.49  0.78  1.46  2258    1
## sigmaY   0.40    0.00 0.05  0.31  0.36  0.40  0.44  0.53  2461    1
## sigmaK   0.75    0.00 0.15  0.50  0.63  0.73  0.84  1.10  2332    1
## sigmaZ   0.55    0.01 0.26  0.08  0.37  0.53  0.70  1.12  1717    1
## delta1  -0.34    0.01 0.41 -1.21 -0.60 -0.32 -0.06  0.46  2074    1
## delta2  -0.94    0.01 0.51 -1.90 -1.28 -0.97 -0.59  0.01  1788    1
## delta3  -0.60    0.01 0.46 -1.53 -0.90 -0.60 -0.28  0.22  1965    1
## lp__    11.72    0.08 3.95  3.36  9.22 12.04 14.56 18.55  2375    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 16 13:53:24 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 4 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)
```

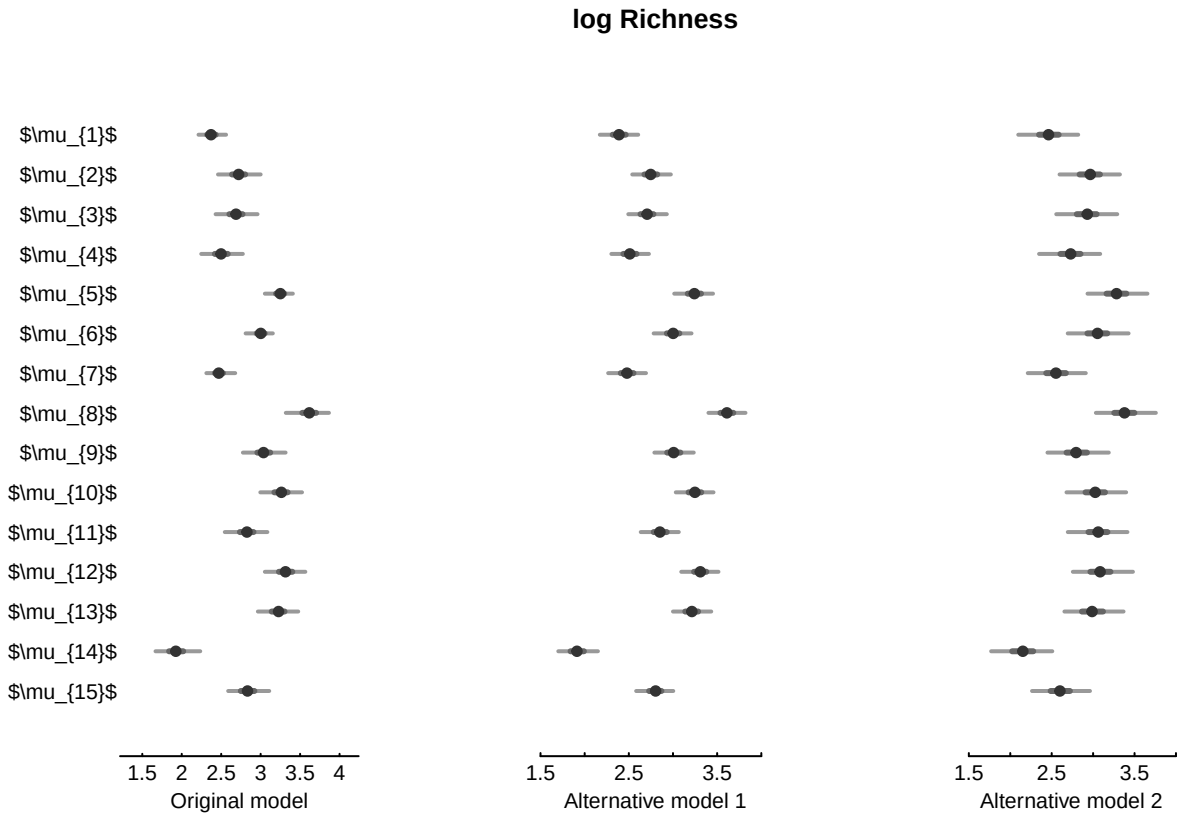
```
## Inference for Stan model: c7e67690f2e5d8b232a67f2588c13147.
## 8 chains, each with iter=5000; warmup=2500; thin=8;
## post-warmup draws per chain=313, total post-warmup draws=2504.
##
##          mean se_mean   sd   2.5%    25%   50%   75% 97.5% n_eff Rhat
## mu0      0.02    0.01 0.36  -0.70  -0.20  0.02  0.23  0.72  2543    1
## muK[1]  -0.27    0.01 0.42  -1.11  -0.55 -0.26  0.00  0.55  2139    1
## muK[2]   0.28    0.01 0.39  -0.48   0.02  0.27  0.53  1.10  2562    1
## muK[3]  -0.33    0.01 0.41  -1.13  -0.61 -0.32 -0.05  0.46  2491    1
## muK[4]  -0.13    0.01 0.41  -0.94  -0.40 -0.13  0.13  0.67  2167    1
## muK[5]   1.07    0.01 0.47   0.21   0.73  1.05  1.38  2.02  2451    1
## muK[6]   0.76    0.01 0.45  -0.11   0.45  0.74  1.06  1.67  2388    1
## muK[7]  -0.56    0.01 0.44  -1.45  -0.85 -0.55 -0.26  0.26  2314    1
## muK[8]   0.76    0.01 0.40   0.00   0.49  0.75  1.03  1.56  2494    1
## muK[9]  -0.02    0.01 0.39  -0.77  -0.28 -0.02  0.22  0.75  2550    1
## muK[10] -0.22    0.01 0.39  -1.02  -0.47 -0.22  0.04  0.54  2384    1
## muK[11] -0.98    0.01 0.46  -1.92  -1.30 -0.98 -0.66 -0.10  2433    1
## muK[12] -0.15    0.01 0.39  -0.94  -0.40 -0.13  0.11  0.61  2528    1
## muK[13]  0.02    0.01 0.39  -0.74  -0.25  0.02  0.27  0.78  2436    1
## muK[14]  0.17    0.01 0.40  -0.60  -0.09  0.17  0.44  0.96  2321    1
## muK[15] -0.45    0.01 0.38  -1.22  -0.70 -0.45 -0.19  0.31  2653    1
## muZ[1]   0.32    0.01 0.41  -0.34   0.04  0.25  0.55  1.31  2546    1
## muZ[2]  -0.30    0.01 0.38  -1.14  -0.51 -0.25 -0.03  0.34  2306    1
## muZ[3]  -0.02    0.01 0.35  -0.78  -0.19 -0.01  0.16  0.70  2296    1
## sigmaY   0.67    0.00 0.09   0.52   0.61  0.66  0.72  0.88  2367    1
## sigmaK   0.66    0.00 0.17   0.36   0.55  0.65  0.77  1.03  2521    1
## sigmaZ   0.43    0.01 0.26   0.04   0.23  0.39  0.59  1.00  2390    1
## delta1  -0.62    0.01 0.50  -1.66  -0.97 -0.59 -0.21  0.15  2293    1
## delta2  -0.28    0.01 0.36  -1.07  -0.52 -0.23 -0.01  0.34  2449    1
## delta3   0.34    0.01 0.40  -0.36   0.04  0.29  0.61  1.19  2309    1
## lp__    -8.15    0.08 3.83 -16.21 -10.52 -7.96 -5.37 -1.54  2194    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 16 13:53:26 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
```

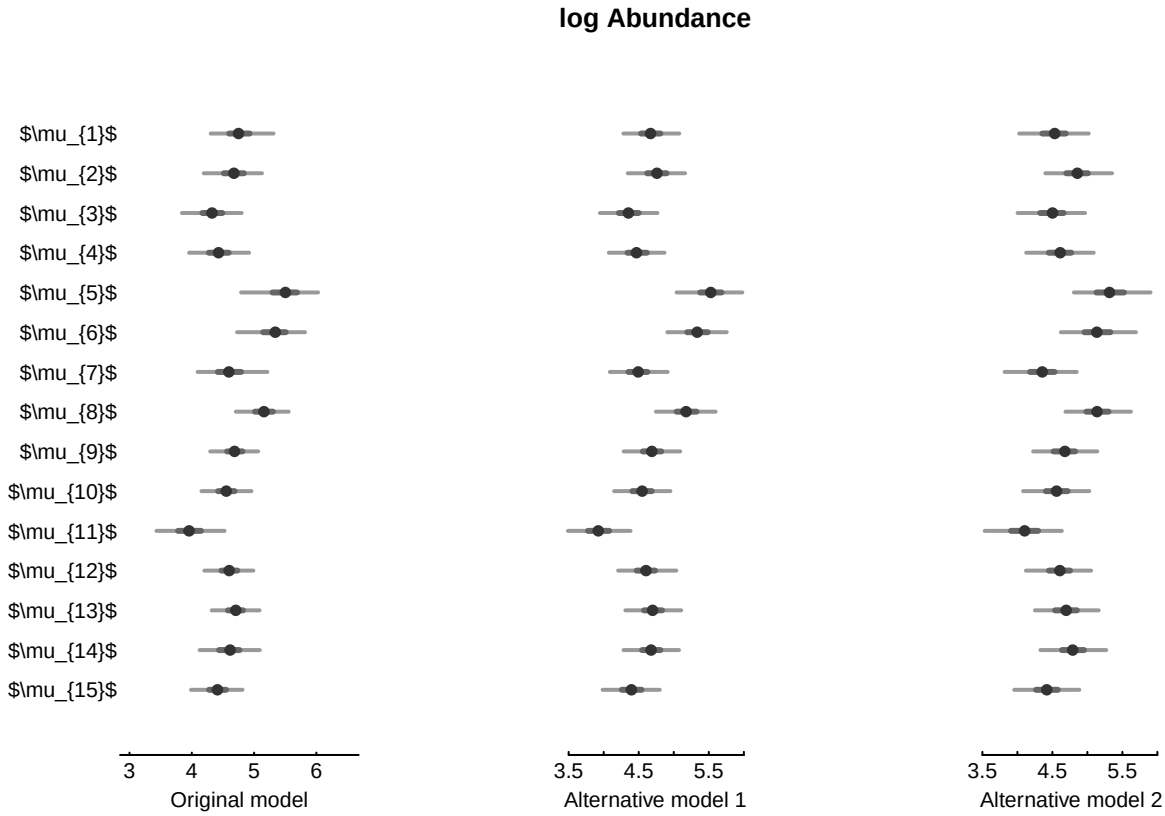
```
delta2A <- c(deltaA21, deltaA22, deltaA23)
names(delta2A) <- c("H-I","H-O","I-O")
```

Now we make some comparisons of the three alternative models

```
## 1. Estimated core means
par(mfrow=c(1,3), mar=c(3,3,2,1), 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)
```

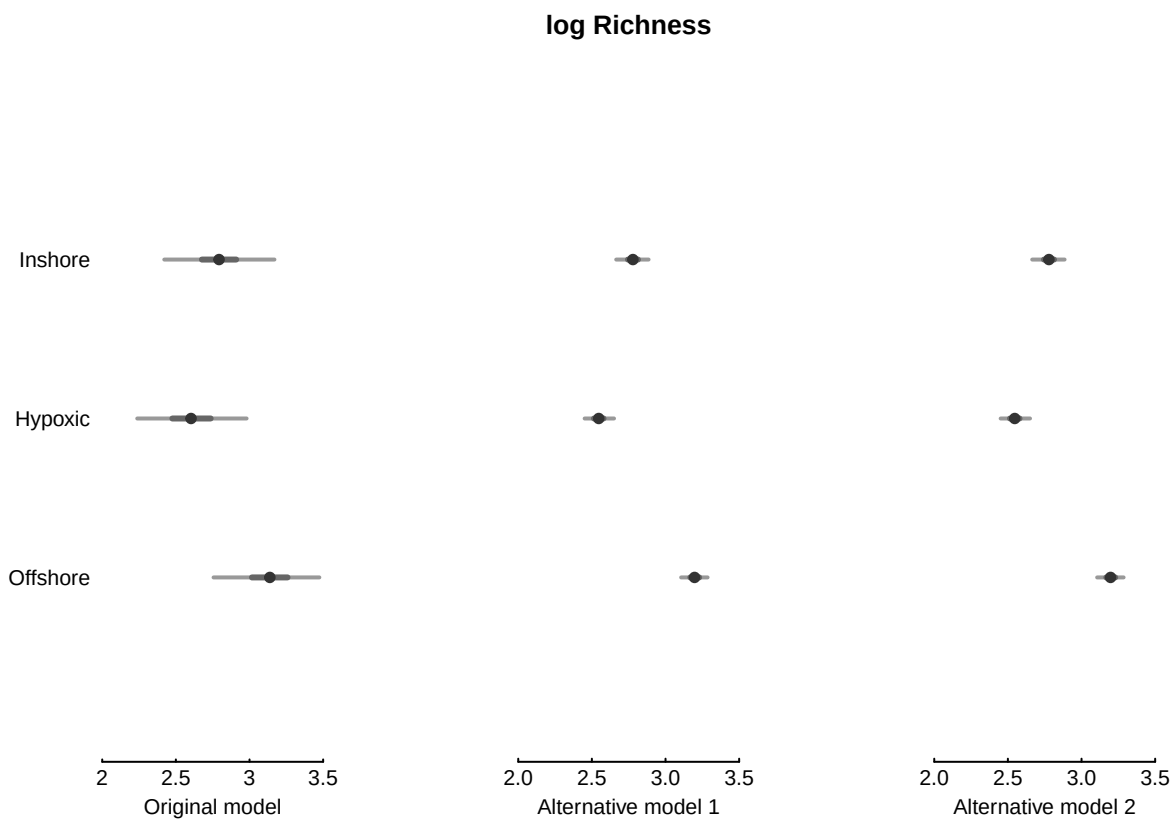



```
par(mfrow=c(1,3), mar=c(3,3,2,1), mgp=c(1.25,0.125,0), las=1, tck=0.01)
mlplot(tempA, xlab="Original model", top.axis=F)
mlplot(core1.musA, xlab="Alternative model 1",
       main="log Abundance", axes=F)
axis(1)
mlplot(core2.musA, xlab="Alternative model 2", axes=F)
axis(1)
```

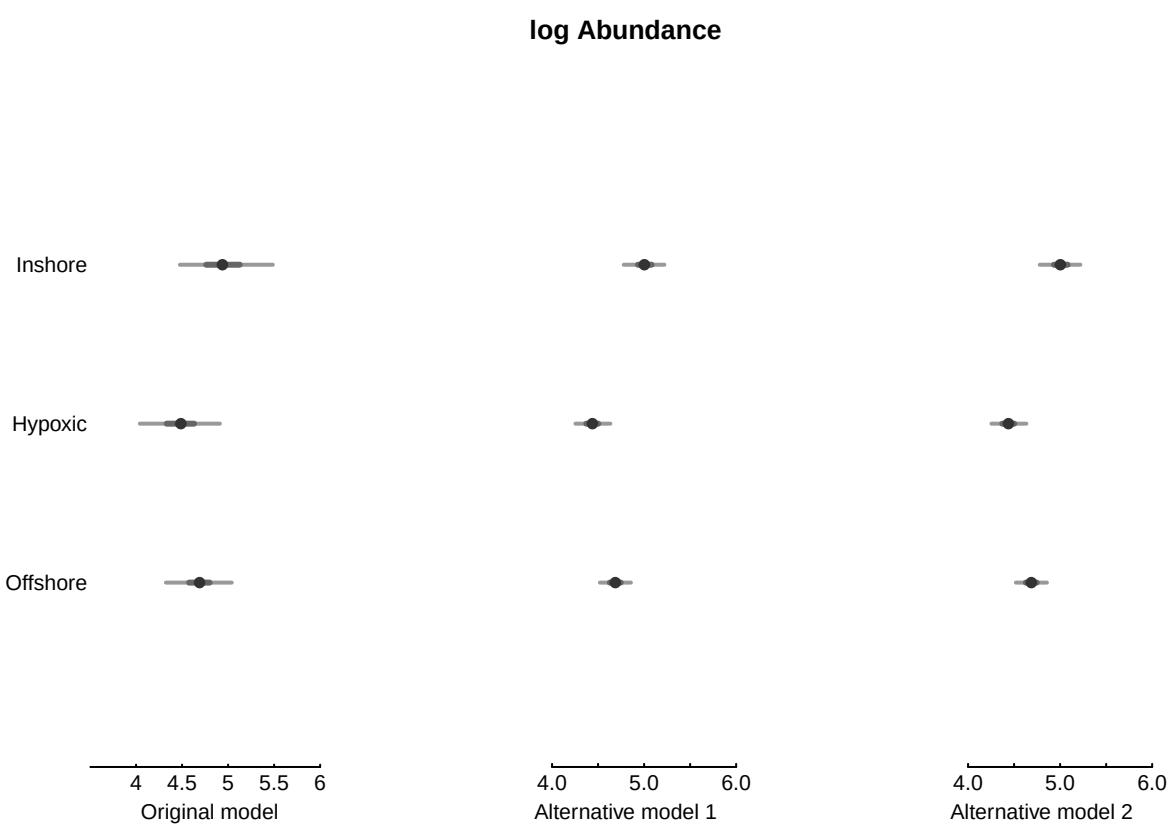



```
## 2. Estimated zone means
#### extract zone means and mean differences
zone0R.thetas <- rich_stan[16:18]
names(zone0R.thetas) <- c("Inshore", "Hypoxic","Offshore")
zone0A.thetas <- abun_stan[16:18]
names(zone0A.thetas) <- c("Inshore", "Hypoxic","Offshore")

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(3,3,2,1), 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)
```

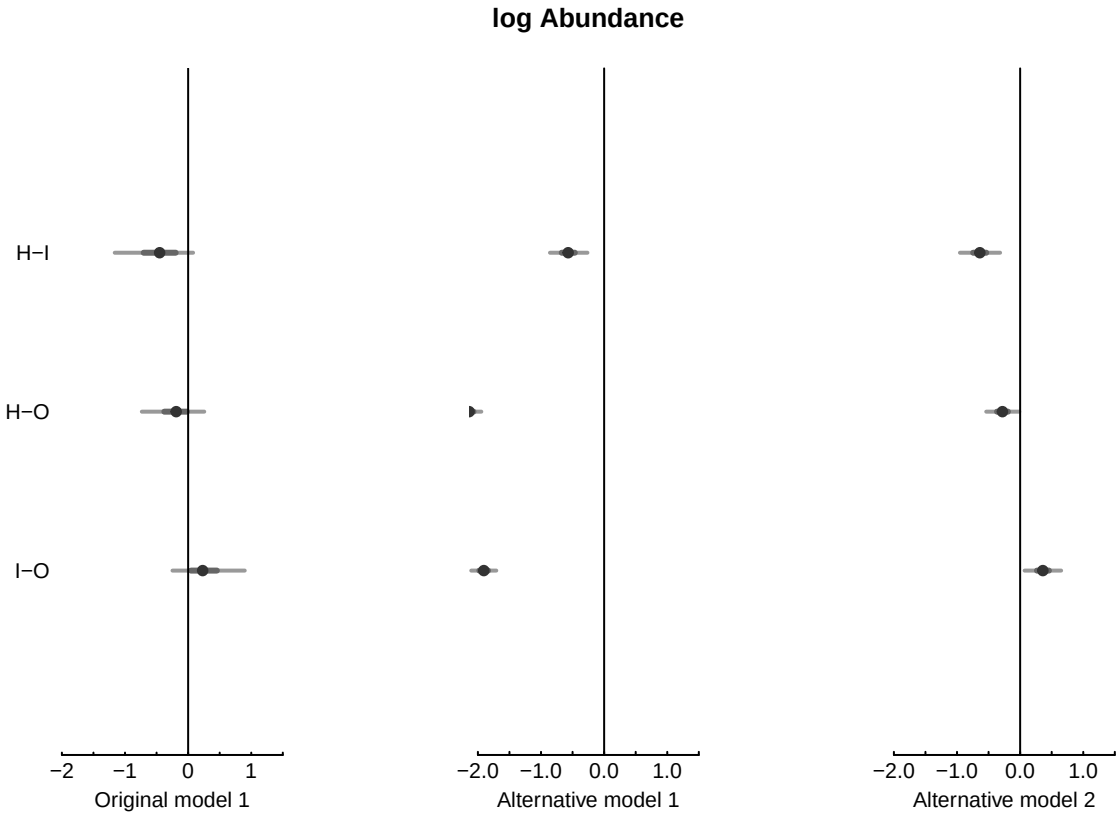
```
abline(v=0)
```




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




```
dev.off()
```

```
## null device
##           1
```

```
par(mfrow=c(1,3), mar=c(3,3,2,1), 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*) – determine the minimum sample size of "statistically meaningful data," data that will result in an estimate of the mean by-catch rate with a limited level of uncertainty. The original study (Bartle 1991) used the Central-Limit-Theorem-based confidence interval to define the acceptable level of uncertainty. As a result, the goal is to estimate the mean and variance of the bycatch numbers. Hilborn and Mangel (1997) used the negative binomial distribution to describe the bycatch data. 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 Poisson is one of three types of distribution for biological measurements.

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

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

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 of λ, because the posterior distribution of λ is also a gamma distribution. If the prior is $\lambda \sim gamma(\alpha, \beta)$, the posterior is $\lambda \mid y_1, \cdots, y_n \sim gamma(\alpha + \sum_{i=1}^{y}, \beta + n)$. 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.

```
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)
```

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

```
## Running /usr/lib/R/bin/R CMD SHLIB foo.c
## gcc -I"/usr/share/R/include" -DNDEBUG   -I"/usr/lib/R/site-library/Rcpp/include/"  -I"/usr/lib/R/sit
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:88,
##                  from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/lib/R/site-library/StanHeaders/include/stan/math/prim/mat/fun/Eigen.hpp:13
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:1: error: unknown type nam
##   628 | namespace Eigen {
##       | ^~~~~~~~~
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:17: error: expected '=',
##   628 | namespace Eigen {
```

```
##       |                 ^
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/lib/R/site-library/StanHeaders/include/stan/math/prim/mat/fun/Eigen.hpp:13
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:96:10: fatal error: complex: No such file or di
##    96 | #include <complex>
##       |          ^~~~~~~~
## compilation terminated.
## make: *** [/usr/lib/R/etc/Makeconf:169: foo.o] Error 1
```

```
## 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)
```

```
## Trying to compile a simple C file
## Running /usr/lib/R/bin/R CMD SHLIB foo.c
## gcc -I"/usr/share/R/include" -DNDEBUG   -I"/usr/lib/R/site-library/Rcpp/include/"  -I"/usr/lib/R/site
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:88,
##                  from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/lib/R/site-library/StanHeaders/include/stan/math/prim/mat/fun/Eigen.hpp:13
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:1: error: unknown type nam
##   628 | namespace Eigen {
##       | ^~~~~~~~~
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:17: error: expected '=',
##   628 | namespace Eigen {
##       |                 ^
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/lib/R/site-library/StanHeaders/include/stan/math/prim/mat/fun/Eigen.hpp:13
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:96:10: fatal error: complex: No such file or di
##    96 | #include <complex>
##       |          ^~~~~~~~
## compilation terminated.
## make: *** [/usr/lib/R/etc/Makeconf:169: foo.o] Error 1
```

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

```
## 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: 2c8d146bbcde4a70e338a76b67a23c93.
## 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  2620    1
## lp__   -789.88    0.01 0.67 -791.86 -790.04 -789.63 -789.44 -789.39  2648    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 16 13:53:58 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).
```

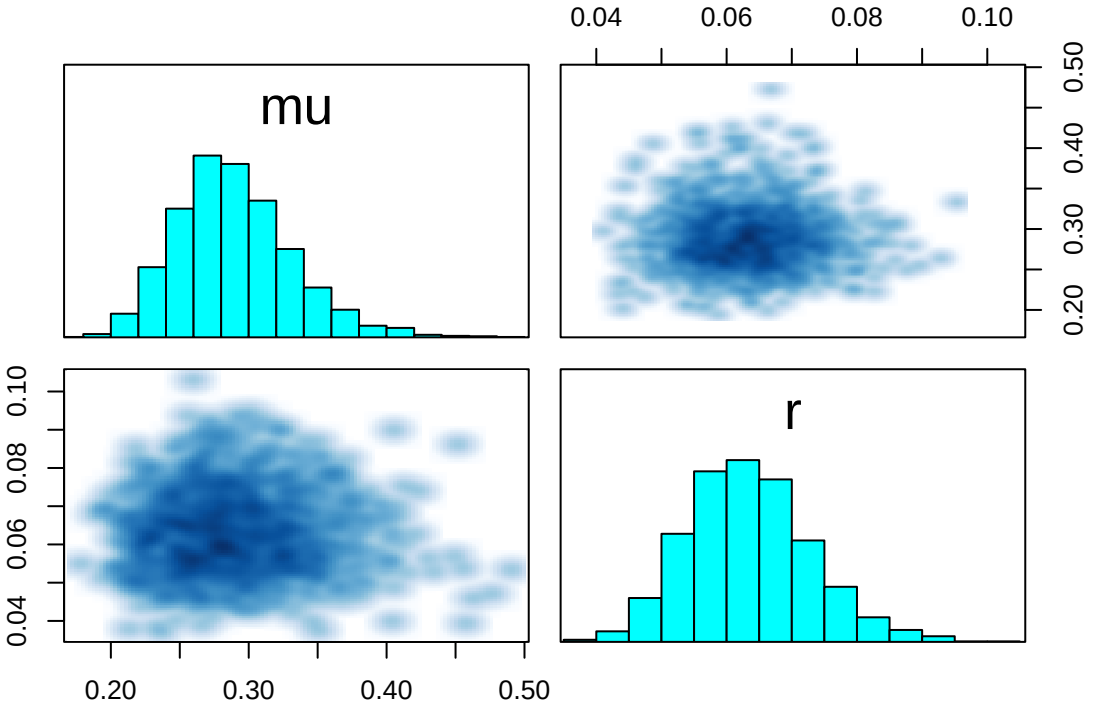
```
Pois_stanout <- extract(keep_Pois, pars="lambda")
Pois_coef <- rvsims(as.matrix(as.data.frame(Pois_stanout)))

## 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)
```

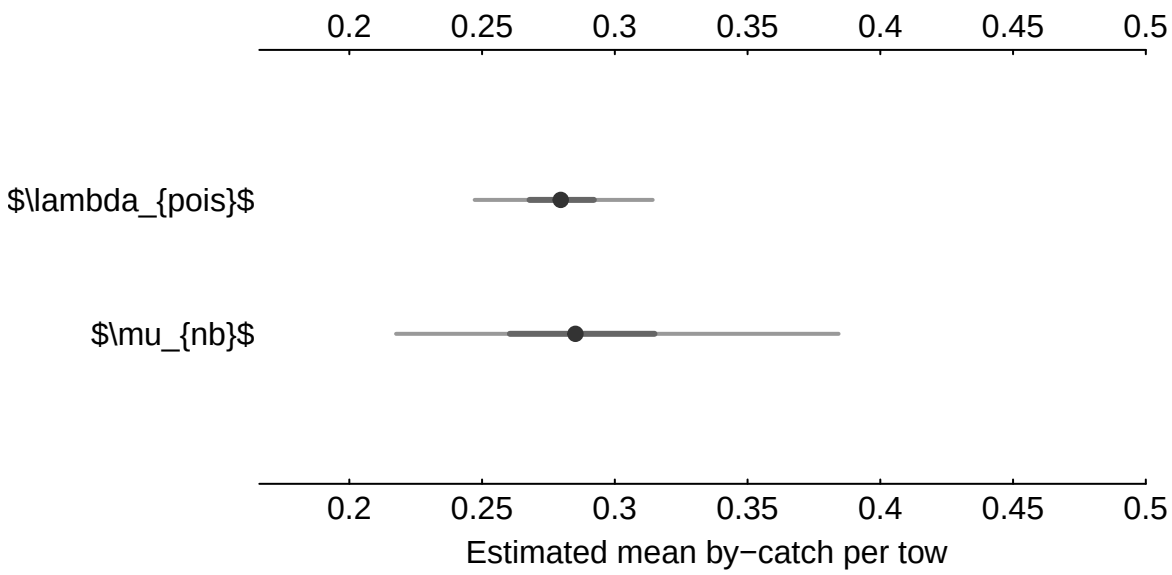
```
## Inference for Stan model: b02914d3ac9f958140e10509f1fb8175.
## 8 chains, each with iter=5000; warmup=2500; thin=8;
## post-warmup draws per chain=313, total post-warmup draws=2504.
##
##         mean se_mean   sd    2.5%     25%     50%     75%   97.5% n_eff Rhat
## mu      0.29    0.00 0.04    0.22    0.26    0.28    0.31    0.38  2677    1
## r       0.06    0.00 0.01    0.05    0.06    0.06    0.07    0.08  2557    1
## lp__ -457.13    0.02 0.98 -459.57 -457.50 -456.81 -456.44 -456.19  2370    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 16 13:54:00 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
```



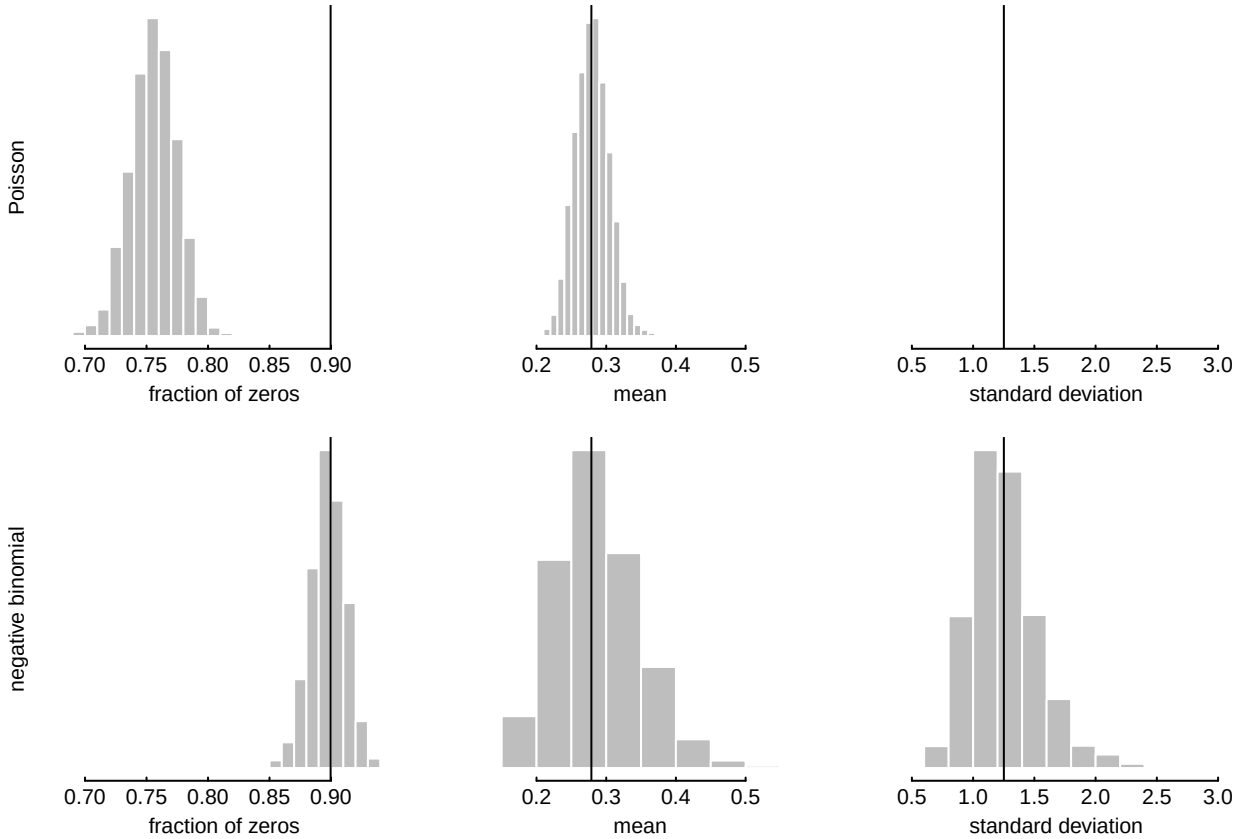

```
NB_stanout <- extract(keep_NB, pars=c("mu","r"))
NB_coef <- rvsims(as.matrix(as.data.frame(NB_stanout)))

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

par(mgp=c(1.25,0.125,0), 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))
}

### 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))
}

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="gray")
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

A zero-inflated model is a mixture model. 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 then,

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

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);
}
```

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:

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

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)
```

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

```
## Running /usr/lib/R/bin/R CMD SHLIB foo.c
## gcc -I"/usr/share/R/include" -DNDEBUG   -I"/usr/lib/R/site-library/Rcpp/include/"  -I"/usr/lib/R/sit
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:88,
##                  from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/lib/R/site-library/StanHeaders/include/stan/math/prim/mat/fun/Eigen.hpp:1
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:1: error: unknown type na
##   628 | namespace Eigen {
##       | ^~~~~~~~~
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:17: error: expected ‘=’,
##   628 | namespace Eigen {
##       |                 ^
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/lib/R/site-library/StanHeaders/include/stan/math/prim/mat/fun/Eigen.hpp:1
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:96:10: fatal error: complex: No such file or di
##    96 | #include <complex>
##       |          ^~~~~~~~~
## compilation terminated.
## make: *** [/usr/lib/R/etc/Makeconf:169: foo.o] Error 1
```

```
fit_zinb <- stan_model(model_code = zinb_bycatch)
```

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

```
## Running /usr/lib/R/bin/R CMD SHLIB foo.c
## gcc -I"/usr/share/R/include" -DNDEBUG   -I"/usr/lib/R/site-library/Rcpp/include/"  -I"/usr/lib/R/sit
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:88,
##                  from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/lib/R/site-library/StanHeaders/include/stan/math/prim/mat/fun/Eigen.hpp:13
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:1: error: unknown type nam
##   628 | namespace Eigen {
##       | ^~~~~~~~~
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:17: error: expected '=',
##   628 | namespace Eigen {
##       |                 ^
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/lib/R/site-library/StanHeaders/include/stan/math/prim/mat/fun/Eigen.hpp:13
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:96:10: fatal error: complex: No such file or di
##    96 | #include <complex>
##       |          ^~~~~~~~~
## compilation terminated.
## make: *** [/usr/lib/R/etc/Makeconf:169: foo.o] Error 1
```

```
################################################
```

Processing data and initial values

```
################################################
## The data and initial values
################################################
ZI_bycatch <- function(yp=zippos, n0=zip0, model="zip", 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 == "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)
```

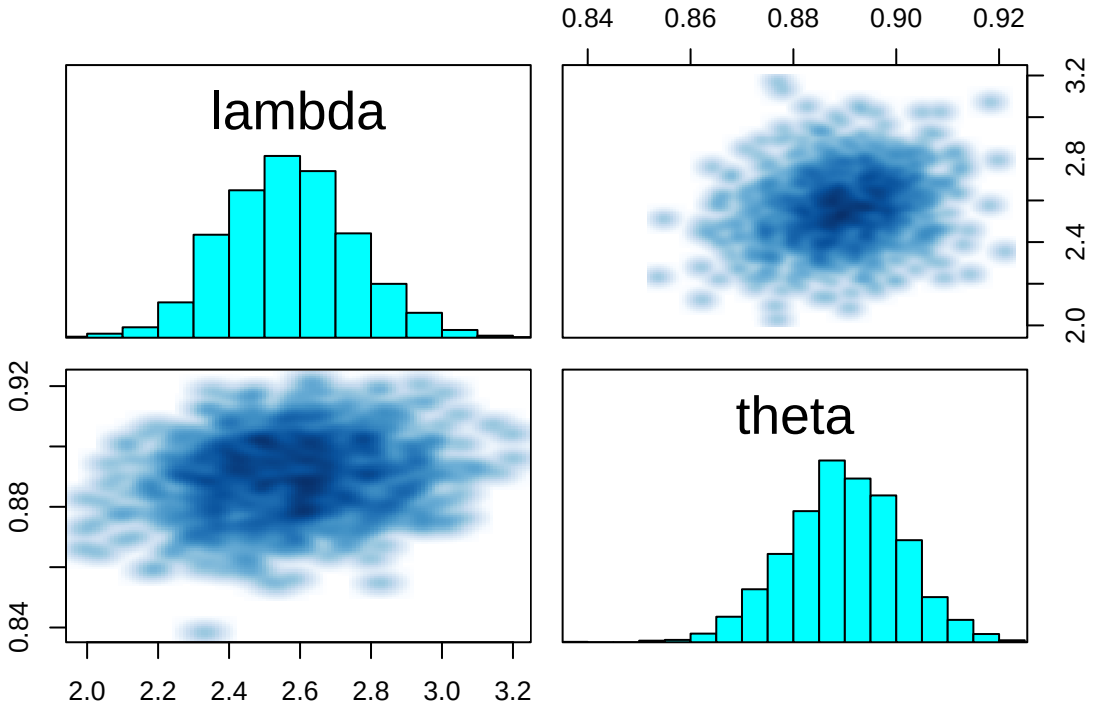
```
## Inference for Stan model: 5fb6aad29cbb96fea10ae791e7385309.
## 8 chains, each with iter=5000; warmup=2500; thin=8;
## post-warmup draws per chain=313, total post-warmup draws=2504.
##
##           mean se_mean   sd    2.5%     25%     50%     75%   97.5% n_eff Rhat
## lambda    2.57    0.00 0.18    2.23    2.45    2.57    2.69    2.94  2415    1
## theta     0.89    0.00 0.01    0.87    0.88    0.89    0.90    0.91  2328    1
## lp__   -501.65    0.02 1.02 -504.48 -502.03 -501.33 -500.93 -500.68  2611    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 16 13:54:33 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: bc371fca729fff26b4b5f7c54296de55.
## 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    0.63    0.00 0.18    0.13    0.54    0.68    0.77    0.84  1902    1
## mu       0.94    0.01 0.41    0.31    0.62    0.89    1.22    1.82  2028    1
## r        0.30    0.00 0.20    0.07    0.15    0.25    0.39    0.81  2012    1
## lp__  -456.84    0.03 1.30 -460.46 -457.39 -456.50 -455.89 -455.37  2178    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 16 13:54:34 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).
```

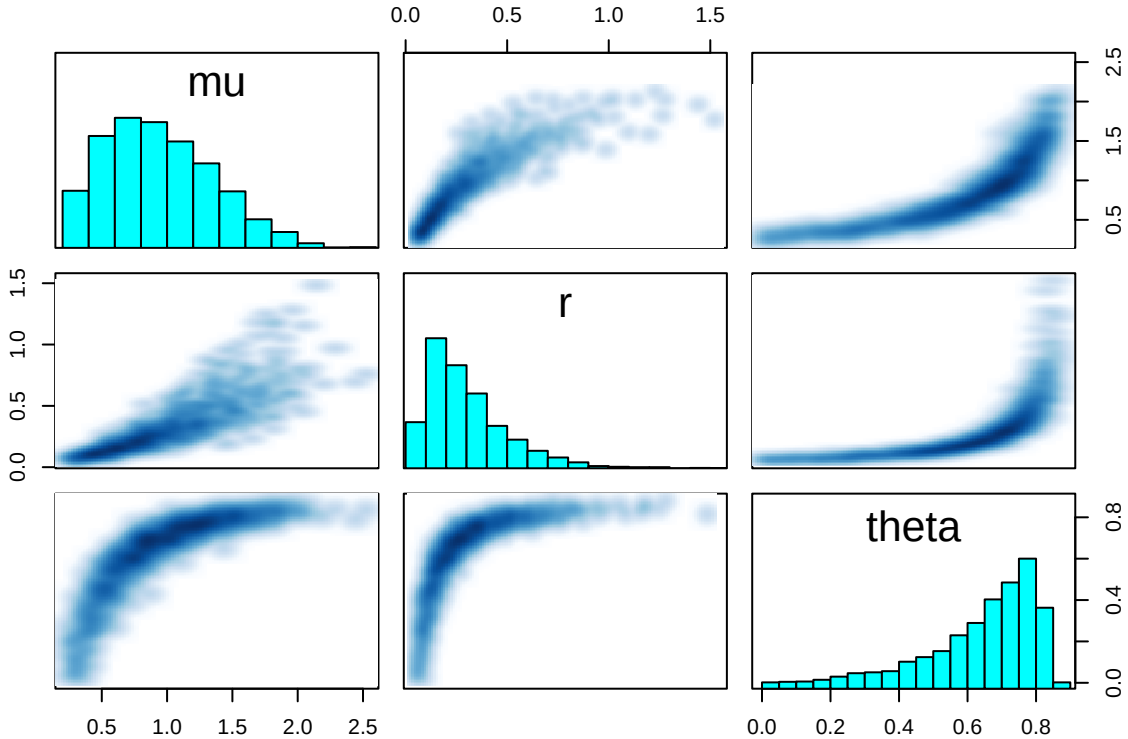
```
##save(keep,file="zinb_bycatch.RData")
pairs(keep_zip, pars=c("lambda","theta"))
```

```
## 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
```

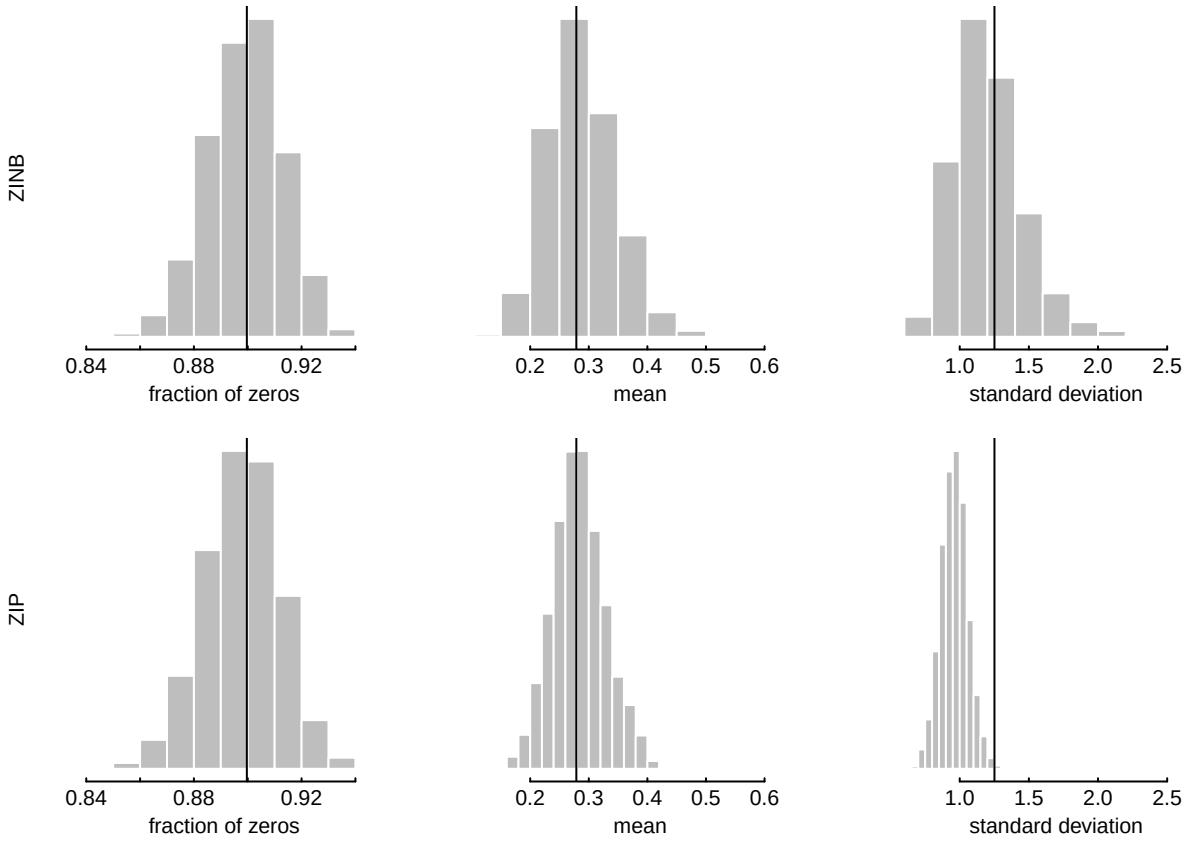


Strong correlation in the zinb model indicating non-identifiability. Additional information (e.g., of θ) is needed.

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

Grass carp, a popular aquaculture species in Asia and an invasive nuisance species in the Great Lakes region. Once grass carp reproduction in Sandusky River, a Lake Erie tributary was confirmed in 2016, a regional task force was established to develop control methods. One approach was to physically remove these fish using electrofishing method. Using capture data from Sandusky River, we developed a crude mode for estimating grass carp population in the area. This example illustrates how to use Bayesian approach to systematically improve an imperfect (or wrong) model to produce useful information.

The objective of estimating grass carp population is to evaluate the effectiveness of the removal effort. We faced several difficulties in obtaining statistically "meaningful" data. First, grass carp is still rare in the region. Although more than 100 fish were removed every year from Sandusky River, the vast majority of sampling trips failed to capture grass carp. When successful, the number of captures is low, mostly 1, rarely 2, and only one time 3. Even with a properly designed (random) sampling plan, we do not have existing

statistical model for this kind of data.

```
## 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 was based on the N-mixture model because the response variable data (number of captures) are generated through two separate processes: 1. Fish movement resulting in an unknown number of individuals present at the sampling site, and 2. The imperfect data collection method (electrofishing).

The N-mixture model explicitly models the two processes using a latent variable method. Assuming there are N individuals at the sampling site when electrofishing statred, the number of captures is a binomial random variable:

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

where p is the "detection" probability, the probability of capturing a fish when it is present. The unknown number N is a 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. The typical statistical solution to this mixture model problem is to fisrt write out the joint

distribution of y_i, p, N_i, and λ:

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

Because N_i is an unknown (latent) variable, we integrate N_i out to obtain the marginal posterior density of p, 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:

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

The first term in the summation is a binomial probability function, the second term is a Poisson probability function, the the last two terms are priors of λ and p.

First, we do not expect the observed response variable data have information for both pp and λ. p is associated with the sampling method, while λ is a characteristics of the target population. As a result, a meaningful prior distribution of p should be quantified separately. Second, although the summation operation is relatively simple for a computer, the infinite upper bound is impossible computationally. Realistically, we do not expect there would be more than 50 fish in the sampling site (roughly 200-500 meters river segment). We can use 50 or 25 as a reasonable upper bound. The log-likelihood is:

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

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 the efficient computation function `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);
  }
}
```

This model is obviously wrong because the fish density along the river cannot be a constant. We initially divided the river into 4 segments and assumed constant density within a segment. It is "less wrong." When the model was implemented with four river segments, the parameter λ is indexed by segments (i.e., λ_j, where j is the segment). We assume that λ_j's are exchangeable. That is, they are different but we do not know a priori how they differ from each other. As such, a natural Bayesian approach is to impose 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 known as the hyper-distribution and its parameters (μ and τ^2) hyper parameters. We used log-transformed λ_j because it is the parameter of a Poisson distribution. Likewise, when the model was implemented for three years, the parameter λ is indexed by year, and the three year-specific densities are assumed exchangeable. It has long been established in statistics literature (e.g., Efron and Morris 1977), that the hierarchical modeling estimated segment- and year-specific parameters (i.e., λ_j) are more accurate

than the estimated parameters using data from individual units separately. Only recently, the hierarchical modeling approach has been introduced to ecologists (REFs).

The full N-mixture model is :

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

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

```
## Running /usr/lib/R/bin/R CMD SHLIB foo.c
## gcc -I"/usr/share/R/include" -DNDEBUG   -I"/usr/lib/R/site-library/Rcpp/include/"  -I"/usr/lib/R/site
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:88,
##                  from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/lib/R/site-library/StanHeaders/include/stan/math/prim/mat/fun/Eigen.hpp:13
```

```
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:1: error: unknown type nam
##   628 | namespace Eigen {
##       | ^~~~~~~~~
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:17: error: expected '=',
##   628 | namespace Eigen {
##       |                 ^
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/lib/R/site-library/StanHeaders/include/stan/math/prim/mat/fun/Eigen.hpp:13
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:96:10: fatal error: complex: No such file or dir
##    96 | #include <complex>
##       |          ^~~~~~~~~
## compilation terminated.
## make: *** [/usr/lib/R/etc/Makeconf:169: foo.o] Error 1
```

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

The model is very slow.

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

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)
```

Because the non-zero count values are mostly 1, there is not much information to quantify the Poisson model

parameter. A simplified model in Qian et al (2022) transformed the count variable into a binary (captured or not captured) variable. For the electronic fishing process, the capture of grass carp is a Bernoulli random variable:

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

where y_i is the either 1 (capture) or 0 (no capture), p_d is the electronic fishing detection probability, θ_i is the probability of at least one grass carp was present, and $Bern$ represents the Bernoulli distribution. The number of grass carp present in the sampling site (N_i) can be modeled as a Poisson random variable with a mean to be a product of the average fish density λ (in number of fish per kilometer) and the size of the sampling site f (in kilometers). That is $N_i \sim Pois(\lambda f)$, from which, we have the probability of at least one grass carp present:

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

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

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

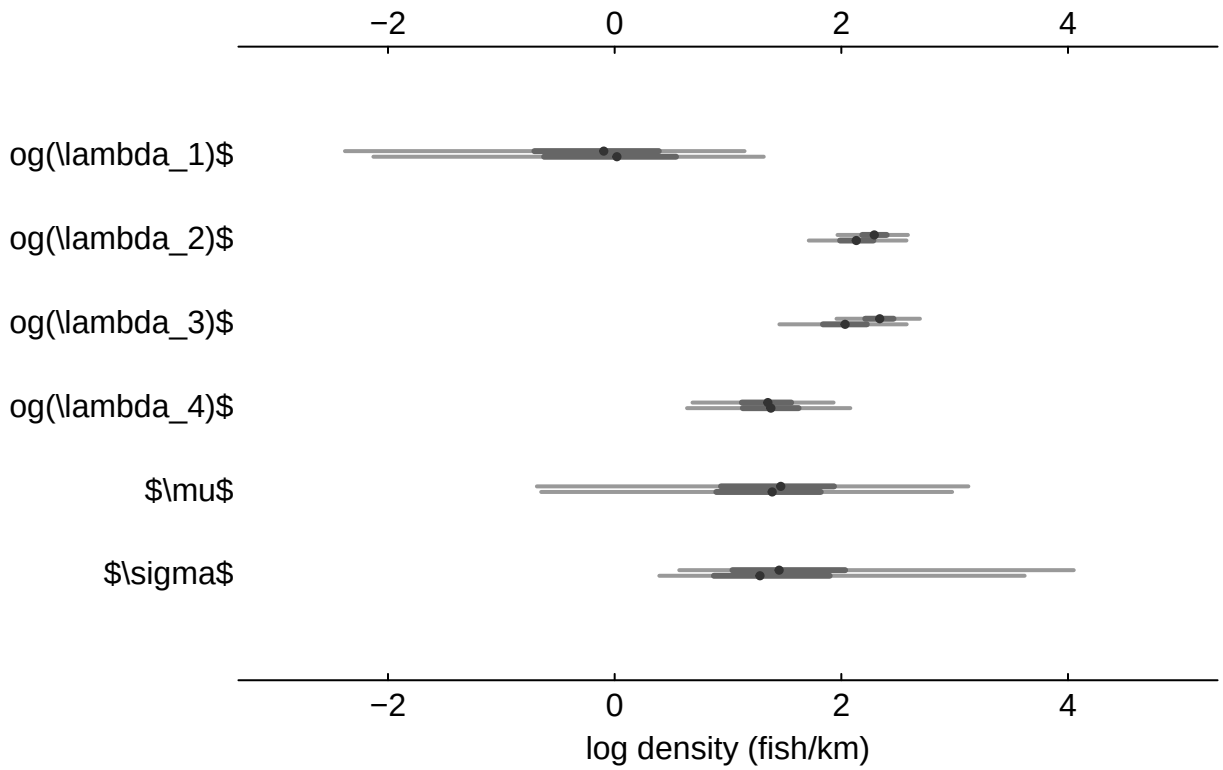
```
## Trying to compile a simple C file
## Running /usr/lib/R/bin/R CMD SHLIB foo.c
## gcc -I"/usr/share/R/include" -DNDEBUG   -I"/usr/lib/R/site-library/Rcpp/include/"  -I"/usr/lib/R/sit
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:88,
##                  from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/lib/R/site-library/StanHeaders/include/stan/math/prim/mat/fun/Eigen.hpp:1
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:1: error: unknown type na
##   628 | namespace Eigen {
##       | ^~~~~~~~
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:17: error: expected '=',
##   628 | namespace Eigen {
##       |                 ^
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/lib/R/site-library/StanHeaders/include/stan/math/prim/mat/fun/Eigen.hpp:1
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:96:10: fatal error: complex: No such file or di
##    96 | #include <complex>
##       |          ^~~~~~~~
## compilation terminated.
## make: *** [/usr/lib/R/etc/Makeconf:169: foo.o] Error 1
```

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

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

##save(fit2keep, fit2keepBin, file="sandusky_compare.RData")
load("sandusky_compare.RData")
## model comparison to justify the use of the simplified model
```

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

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

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

Mixture_coef <- cbind(NM_coef, NMbin_coef)

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

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



This model satisfies Cox's three criteria of an applied statistical model. The response variable model is the Bernoulli distribution, the parameter p_d describes the effectiveness of the sampling method and the parameter λ is the parameter of interest describing the average fish density from which we can derive estimate of fish population. Although the division of the river segments were selected based on data availability (not ecological considerations), we can use this model to evaluate temporal trend of the population through annual sequential updating. The concept of updating upon new data is uniquely Bayesian. We treat the posterior distribution of model parameters at one time to be the priors for the next time step. Because the grass carp population is under systematic removal pressure, we have to assume that the population changes every year. When we have multiple years of data, we often use a Bayesian hierarchical model to reflect such uncertainty – annual populations are different but we are uncertain how they differ. The hierarchical model can be summarized as a hyper-distribution of segment-specific densities. That is:

$$\log(\lambda_j) \sim N(\mu_\lambda, \tau_\lambda^2)$$

where the λ_j is the parameter representing the mean population density in segment j. The hyper-parameter μ_λ is the log-mean of segment means and the hyper-parameter τ_λ^2 is the among segment variance. In order to represent the changes of population over time, we can divide the data into segment (j) and year (t) and write the model for each year:

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

As we don't have information about how the hyper-parameters vary by year, we can use a second layer of hierarchical model:

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

That is, we impose a common prior for the annual hyper-parameters $\mu_{\lambda,t}, \tau_{\lambda,t}^2$. The common prior is the normal-inverse-gamma distribution, the conjugate family of priors. In a Bayesian hierarchical model, we impose vague (or flat) priors for the hyper-parameters and use data (here multiple years of data) to estimate the parameters. Qian et al (2022) proposed to sequentially update a hierarchical model without waiting to have data from multiple years. Specifically, we use estimated hyper-parameters based on available data to estimate their prior parameters (i.e., $\mu_0, n_0, \alpha, \beta$) and with data from each additional year, we directly apply the Bayesian estimator (i.e., $\log(\lambda_{jt}) \sim N(\mu_{\lambda,t}, \nu_{\lambda,t}^2)$) and update the hyper-parameters to derive the prior parameters for the next year.

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

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

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

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

```
##### Single detection probability
BernBern_seq <- "
data {
  int<lower=0> Nobs;
  int<lower=0> Nseg;
  int<lower=0,upper=Nseg> segments[Nobs];
  int<lower=0,upper=1> y[Nobs];
  real<lower=0> a;
```

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

```
## Trying to compile a simple C file
## Running /usr/lib/R/bin/R CMD SHLIB foo.c
## gcc -I"/usr/share/R/include" -DNDEBUG   -I"/usr/lib/R/site-library/Rcpp/include/"  -I"/usr/lib/R/sit
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:88,
##                  from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/lib/R/site-library/StanHeaders/include/stan/math/prim/mat/fun/Eigen.hpp:13
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:1: error: unknown type nam
##   628 | namespace Eigen {
##       | ^~~~~~~~~
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:17: error: expected '=',
##   628 | namespace Eigen {
##       |                 ^
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/lib/R/site-library/StanHeaders/include/stan/math/prim/mat/fun/Eigen.hpp:13
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:96:10: fatal error: complex: No such file or di
```

```
##    96 | #include <complex>
##       |          ^~~~~~~~
## compilation terminated.
## make: *** [/usr/lib/R/etc/Makeconf:169: foo.o] Error 1
```

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: 8fdc837ff8c762f4d39bf130f261ad07.
## 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]   -0.83    0.02 0.74   -2.55   -1.26   -0.75   -0.30    0.40
```

```
## log_lambda[2]    1.30    0.01 0.37    0.55    1.05    1.30    1.55    2.03
## log_lambda[3]    1.29    0.01 0.30    0.70    1.09    1.29    1.49    1.87
## log_lambda[4]    0.96    0.01 0.42    0.14    0.69    0.97    1.24    1.76
## mu               0.99    0.01 0.52   -0.08    0.69    1.00    1.31    1.99
## sigma2           1.95    0.03 1.19    0.67    1.15    1.63    2.38    5.25
## pd               0.51    0.00 0.04    0.43    0.49    0.51    0.54    0.59
## lp__          -227.52    0.04 1.87 -231.94 -228.59 -227.20 -226.15 -224.81
##               n_eff Rhat
## log_lambda[1]  2336    1
## log_lambda[2]  2599    1
## log_lambda[3]  2426    1
## log_lambda[4]  2239    1
## mu             2370    1
## sigma2         2067    1
## pd             2653    1
## lp__           2426    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 16 13:55:33 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: 8fdc837ff8c762f4d39bf130f261ad07.
## 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 0.61   -2.26   -1.26   -0.86   -0.51    0.15
## log_lambda[2]   -0.58    0.01 0.40   -1.39   -0.84   -0.57   -0.31    0.18
## log_lambda[3]    0.03    0.01 0.27   -0.52   -0.15    0.04    0.22    0.54
## log_lambda[4]    0.18    0.01 0.33   -0.50   -0.04    0.19    0.40    0.81
## mu               0.52    0.01 0.40   -0.25    0.26    0.51    0.78    1.33
## sigma2           1.78    0.02 0.78    0.77    1.23    1.61    2.15    3.74
## pd               0.51    0.00 0.04    0.44    0.49    0.51    0.54    0.59
## lp__          -247.86    0.04 1.86 -252.35 -248.91 -247.52 -246.47 -245.19
##               n_eff Rhat
## log_lambda[1]  2524    1
## log_lambda[2]  2539    1
## log_lambda[3]  2510    1
## log_lambda[4]  2724    1
## mu             2400    1
## sigma2         2419    1
## pd             2424    1
```

```
## lp__           2468    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 16 13:55:43 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: 8fdc837ff8c762f4d39bf130f261ad07.
## 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]   -0.70    0.01 0.55   -1.89   -1.04   -0.68   -0.32    0.29
## log_lambda[2]   -0.70    0.01 0.45   -1.66   -0.98   -0.67   -0.39    0.10
## log_lambda[3]   -1.33    0.01 0.62   -2.67   -1.71   -1.29   -0.88   -0.24
## log_lambda[4]   -1.23    0.01 0.74   -2.87   -1.68   -1.17   -0.70    0.02
## mu               0.13    0.01 0.35   -0.61   -0.11    0.13    0.36    0.79
## sigma2           1.86    0.01 0.71    0.93    1.36    1.72    2.17    3.63
## pd               0.51    0.00 0.04    0.43    0.48    0.51    0.54    0.59
## lp__          -177.36    0.04 1.91 -182.19 -178.37 -177.00 -175.97 -174.67
##                n_eff Rhat
## log_lambda[1]   2524    1
## log_lambda[2]   2492    1
## log_lambda[3]   2341    1
## log_lambda[4]   2686    1
## mu              2483    1
## sigma2          2455    1
## pd              2518    1
## lp__            2656    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 16 13:55:48 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

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

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

expressed as:

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

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

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

```
}
"

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

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

```
## Running /usr/lib/R/bin/R CMD SHLIB foo.c
## gcc -I"/usr/share/R/include" -DNDEBUG   -I"/usr/lib/R/site-library/Rcpp/include/"  -I"/usr/lib/R/site
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:88,
##                  from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/lib/R/site-library/StanHeaders/include/stan/math/prim/mat/fun/Eigen.hpp:13
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:1: error: unknown type nam
##   628 | namespace Eigen {
##       | ^~~~~~~~~
## /usr/lib/R/site-library/RcppEigen/include/Eigen/src/Core/util/Macros.h:628:17: error: expected '=',
##   628 | namespace Eigen {
##       |                 ^
## In file included from /usr/lib/R/site-library/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/lib/R/site-library/StanHeaders/include/stan/math/prim/mat/fun/Eigen.hpp:13
##                  from <command-line>:
## /usr/lib/R/site-library/RcppEigen/include/Eigen/Core:96:10: fatal error: complex: No such file or di
##    96 | #include <complex>
##       |          ^~~~~~~~~
## compilation terminated.
## make: *** [/usr/lib/R/etc/Makeconf:169: foo.o] Error 1
```

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

```
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: 32d348c48f9897da035bade2cfb83e51.
## 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]    -1.06    0.03 1.28   -3.93   -1.74   -0.96   -0.20    1.13
## log_lambda[2]    -0.84    0.02 0.88   -2.75   -1.36   -0.75   -0.26    0.73
## log_lambda[3]    -1.39    0.02 1.25   -4.38   -1.99   -1.23   -0.54    0.52
## log_lambda[4]     0.80    0.02 1.27   -2.32    0.24    0.98    1.54    2.85
## log_lambda[5]     1.37    0.01 0.72    0.19    0.94    1.32    1.71    2.86
## log_lambda[6]     1.40    0.01 0.52    0.50    1.08    1.37    1.67    2.43
## log_lambda[7]     1.12    0.03 1.31   -1.90    0.52    1.17    1.80    3.64
## log_lambda[8]     1.30    0.01 0.40    0.56    1.05    1.29    1.54    2.09
## log_lambda[9]     1.36    0.01 0.51    0.51    1.05    1.33    1.62    2.36
## log_lambda[10]    0.47    0.03 1.29   -2.50   -0.11    0.61    1.19    2.60
## log_lambda[11]    0.42    0.01 0.67   -1.08    0.03    0.48    0.87    1.61
## log_lambda[12]    1.64    0.02 0.92    0.36    1.07    1.50    2.00    4.01
## mu[1]            -0.69    0.02 0.97   -2.73   -1.28   -0.67   -0.06    1.14
## mu[2]             1.17    0.01 0.73   -0.30    0.74    1.19    1.62    2.62
## mu[3]             1.22    0.01 0.72   -0.26    0.83    1.23    1.62    2.69
## mu[4]             0.87    0.01 0.75   -0.65    0.45    0.90    1.31    2.38
## m                 0.98    0.01 0.55   -0.17    0.64    1.00    1.32    2.03
## sigma2            1.45    0.04 1.83    0.03    0.27    0.75    1.82    7.24
## tau2              2.02    0.02 1.23    0.66    1.20    1.67    2.46    5.34
## pd                0.50    0.00 0.04    0.42    0.48    0.50    0.53    0.58
## lp__           -233.15    0.08 3.79 -241.16 -235.61 -232.70 -230.41 -226.86
##                n_eff Rhat
## log_lambda[1]   2574    1
## log_lambda[2]   2863    1
```

```
## log_lambda[3]   2696    1
## log_lambda[4]   2649    1
## log_lambda[5]   2555    1
## log_lambda[6]   2548    1
## log_lambda[7]   2549    1
## log_lambda[8]   2500    1
## log_lambda[9]   2647    1
## log_lambda[10]  2339    1
## log_lambda[11]  2361    1
## log_lambda[12]  2455    1
## mu[1]           2674    1
## mu[2]           2470    1
## mu[3]           2544    1
## mu[4]           2546    1
## m               2478    1
## sigma2          2513    1
## tau2            2474    1
## pd              2480    1
## lp__            2345    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 16 13:56:14 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: 32d348c48f9897da035bade2cfb83e51.
## 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 1.27   -4.68   -2.47   -1.53   -0.83    0.30
## log_lambda[2]    -1.83    0.02 1.27   -4.74   -2.52   -1.66   -0.95    0.16
## log_lambda[3]    -0.54    0.02 0.77   -2.19   -1.02   -0.48   -0.01    0.84
## log_lambda[4]    -0.48    0.01 0.72   -2.04   -0.93   -0.45    0.02    0.79
## log_lambda[5]    -2.02    0.02 1.18   -4.90   -2.67   -1.82   -1.18   -0.24
## log_lambda[6]    -0.28    0.01 0.56   -1.43   -0.64   -0.24    0.10    0.71
## log_lambda[7]     0.93    0.01 0.46    0.07    0.63    0.93    1.22    1.83
## log_lambda[8]    -0.84    0.01 0.56   -2.03   -1.19   -0.79   -0.46    0.16
## log_lambda[9]    -0.34    0.01 0.51   -1.42   -0.65   -0.30    0.02    0.57
## log_lambda[10]    0.92    0.01 0.57   -0.12    0.55    0.90    1.26    2.05
## log_lambda[11]   -0.95    0.02 0.84   -2.89   -1.43   -0.86   -0.37    0.42
## log_lambda[12]   -0.16    0.01 0.58   -1.39   -0.51   -0.12    0.23    0.88
## mu[1]            -0.89    0.02 0.83   -2.61   -1.40   -0.84   -0.35    0.70
```

```
## mu[2]            -0.63    0.02 0.74   -2.10   -1.10   -0.62   -0.16    0.79
## mu[3]            -0.12    0.01 0.67   -1.47   -0.51   -0.11    0.29    1.25
## mu[4]            -0.09    0.01 0.70   -1.53   -0.51   -0.07    0.34    1.26
## m                -0.20    0.01 0.59   -1.37   -0.55   -0.18    0.17    0.93
## sigma2            2.04    0.04 1.73    0.25    0.86    1.48    2.62    7.04
## tau2              1.07    0.02 0.78    0.35    0.61    0.85    1.26    3.02
## pd                0.51    0.00 0.04    0.44    0.49    0.51    0.54    0.59
## lp__           -240.68    0.08 3.94 -249.57 -243.14 -240.33 -237.79 -234.00
##                n_eff Rhat
## log_lambda[1]   2342    1
## log_lambda[2]   2602    1
## log_lambda[3]   2535    1
## log_lambda[4]   2606    1
## log_lambda[5]   2349    1
## log_lambda[6]   2454    1
## log_lambda[7]   2576    1
## log_lambda[8]   2455    1
## log_lambda[9]   2653    1
## log_lambda[10]  2534    1
## log_lambda[11]  2543    1
## log_lambda[12]  2702    1
## mu[1]           2635    1
## mu[2]           2366    1
## mu[3]           2516    1
## mu[4]           2394    1
## m               2421    1
## sigma2          2358    1
## tau2            2221    1
## pd              2720    1
## lp__            2446    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 16 13:56:30 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: 32d348c48f9897da035bade2cfb83e51.
## 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]   -0.41    0.02 0.81   -2.07   -0.91   -0.40    0.13    1.13
## log_lambda[2]   -1.77    0.02 1.18   -4.80   -2.31   -1.58   -0.96   -0.09
```

```
## log_lambda[3]    -0.69    0.02 0.90   -2.64   -1.24   -0.64   -0.09    0.93
## log_lambda[4]    -0.61    0.01 0.69   -2.06   -1.03   -0.60   -0.14    0.67
## log_lambda[5]    -0.97    0.01 0.65   -2.47   -1.34   -0.91   -0.53    0.13
## log_lambda[6]    -1.01    0.02 0.80   -2.78   -1.47   -0.92   -0.46    0.37
## log_lambda[7]    -1.38    0.02 0.89   -3.27   -1.95   -1.34   -0.76    0.26
## log_lambda[8]    -1.85    0.02 0.85   -3.82   -2.34   -1.76   -1.26   -0.41
## log_lambda[9]    -1.88    0.02 1.23   -4.75   -2.50   -1.74   -1.11    0.12
## log_lambda[10]   -1.88    0.03 1.23   -4.75   -2.51   -1.72   -1.07    0.15
## log_lambda[11]   -1.52    0.02 0.90   -3.62   -2.05   -1.44   -0.89   -0.01
## log_lambda[12]   -1.73    0.03 1.31   -4.84   -2.44   -1.59   -0.88    0.47
## mu[1]            -0.92    0.02 0.73   -2.49   -1.38   -0.87   -0.43    0.41
## mu[2]            -0.87    0.01 0.69   -2.28   -1.27   -0.85   -0.43    0.45
## mu[3]            -1.52    0.02 0.81   -3.19   -2.02   -1.50   -0.99   -0.01
## mu[4]            -1.52    0.02 0.89   -3.47   -2.06   -1.47   -0.93    0.07
## m                -0.87    0.01 0.53   -1.96   -1.19   -0.86   -0.53    0.17
## sigma2            1.37    0.03 1.65    0.03    0.31    0.79    1.76    6.54
## tau2              1.51    0.02 0.90    0.58    0.94    1.29    1.77    3.86
## pd                0.52    0.00 0.04    0.44    0.49    0.52    0.55    0.59
## lp__           -175.73    0.08 3.71 -183.98 -178.04 -175.39 -173.04 -169.51
##                n_eff Rhat
## log_lambda[1]   2473    1
## log_lambda[2]   2681    1
## log_lambda[3]   2418    1
## log_lambda[4]   2488    1
## log_lambda[5]   2366    1
## log_lambda[6]   2270    1
## log_lambda[7]   2484    1
## log_lambda[8]   2548    1
## log_lambda[9]   2456    1
## log_lambda[10]  2399    1
## log_lambda[11]  2388    1
## log_lambda[12]  2515    1
## mu[1]           2268    1
## mu[2]           2387    1
## mu[3]           2608    1
## mu[4]           2503    1
## m               2350    1
## sigma2          2453    1
## tau2            2342    1
## pd              2545    1
## lp__            2347    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 16 13:56:39 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).
```

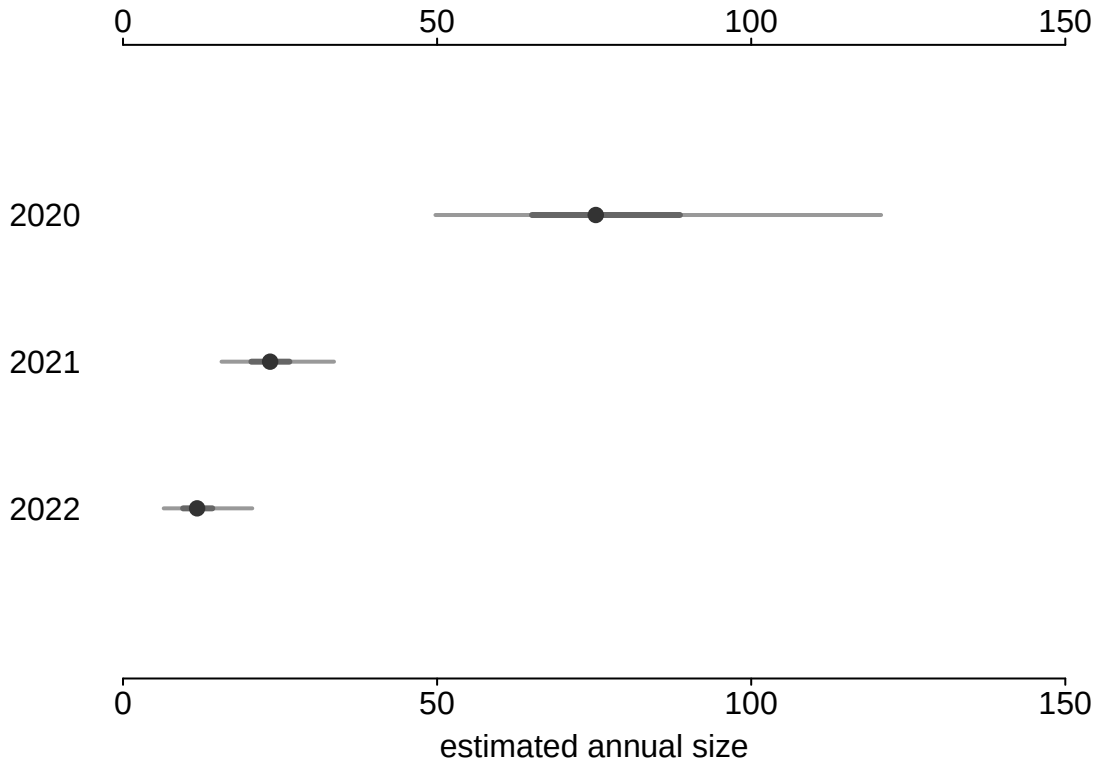
Running the model is not the most challenging task, communicating the results to ecologists is. I tried some plots but am not entirely happy with them.

```
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)))
```

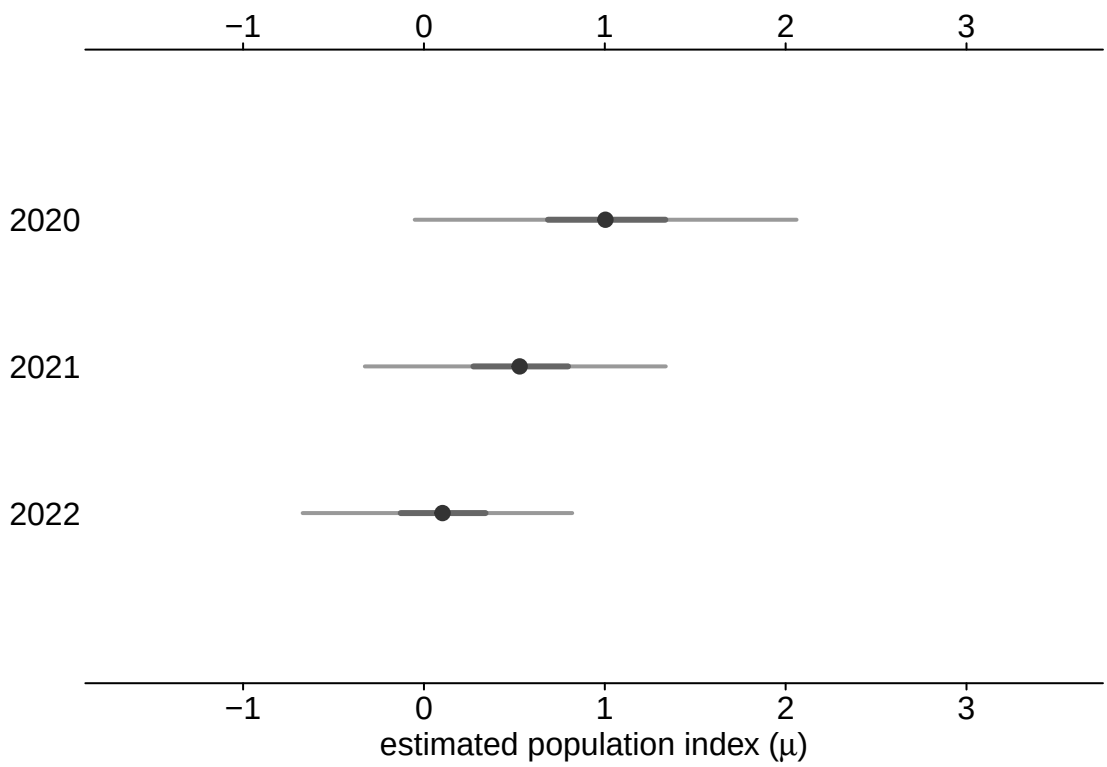
```
coef2020nested <- extract(fit2keepNested2020, pars=c("log_lambda","mu","m"))
nestCoefrv20 <- rvsims(as.matrix(as.data.frame(coef2020nested)))
coef2021seg <- extract(fit2keepSeg_2021, pars=c("log_lambda","mu"))
segCoefrv21 <- rvsims(as.matrix(as.data.frame(coef2021seg)))
coef2021sn <- extract(fit2keepSn_2021, 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))
```

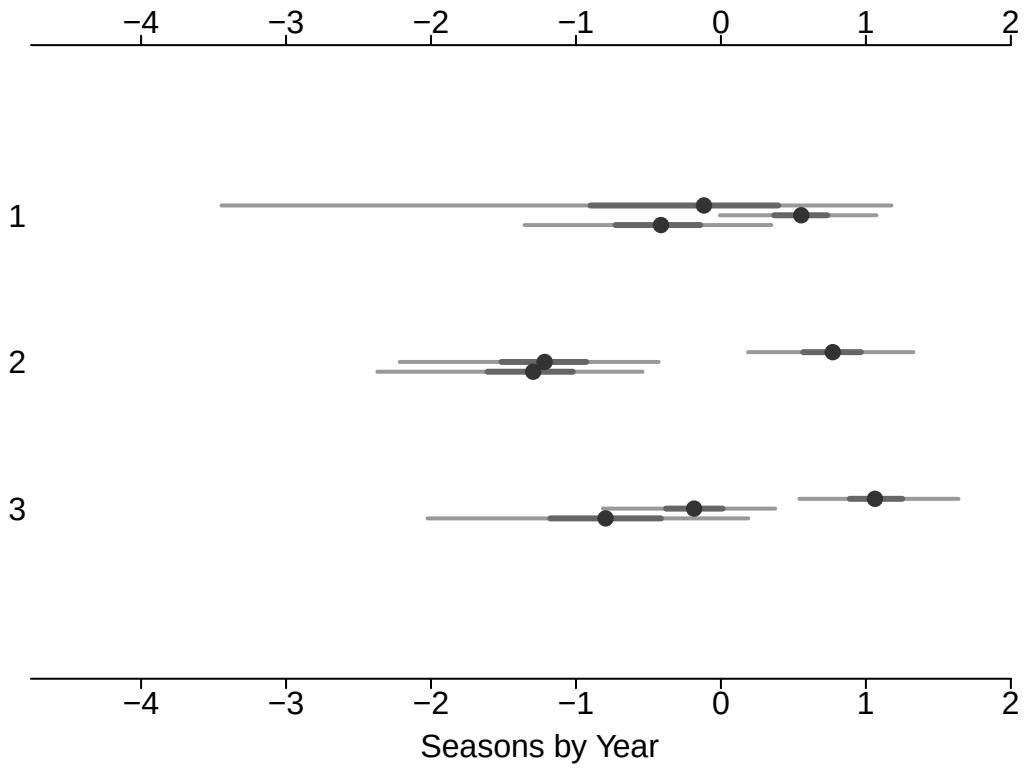



```
par(mar=c(3, 2, 1, 1), mgp=c(1.25,0.125,0), tck=-0.01)
mlplot(musSeg, xlab=expression(paste("estimated population index (",mu, ")", sep="")))
```

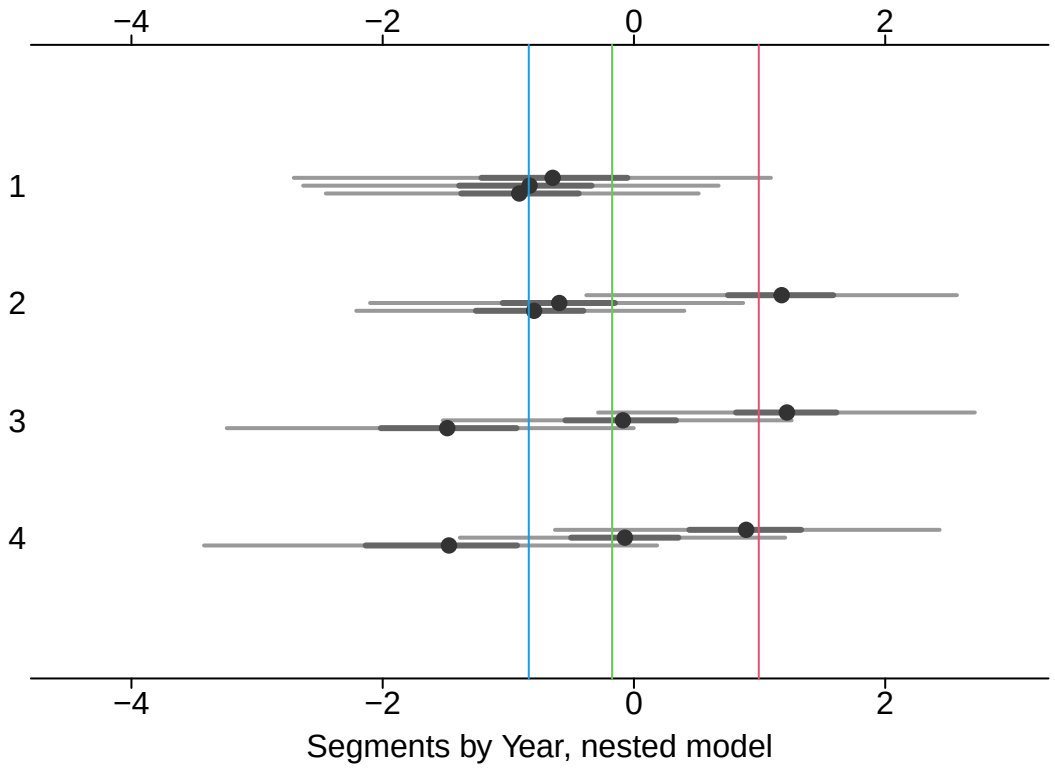



```
### end of annual numbers

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

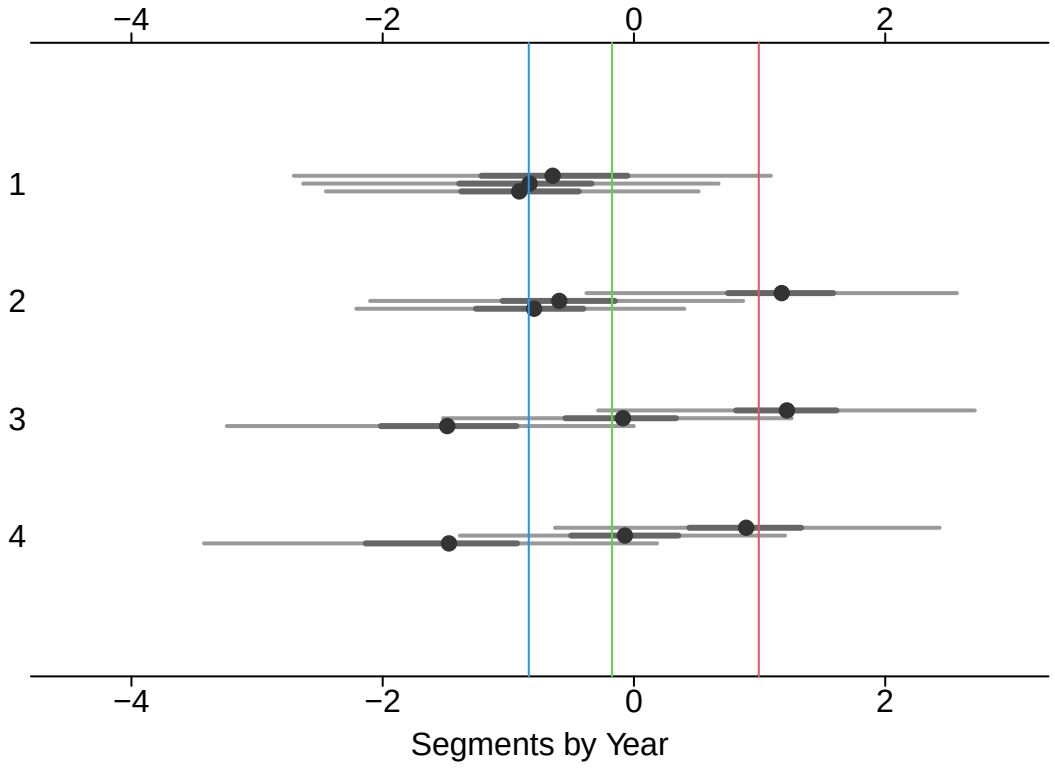
```
musSn <- c(snCoefrv20[4], snCoefrv21[4], snCoefrv22[4])
par(mar=c(3, 1, 1, 2), mgp=c(1.25,0.125,0), tck=-0.015)
mlplot(logLambdaSn, xlab="Seasons by Year", xlim=c(-4.5, 2))
```



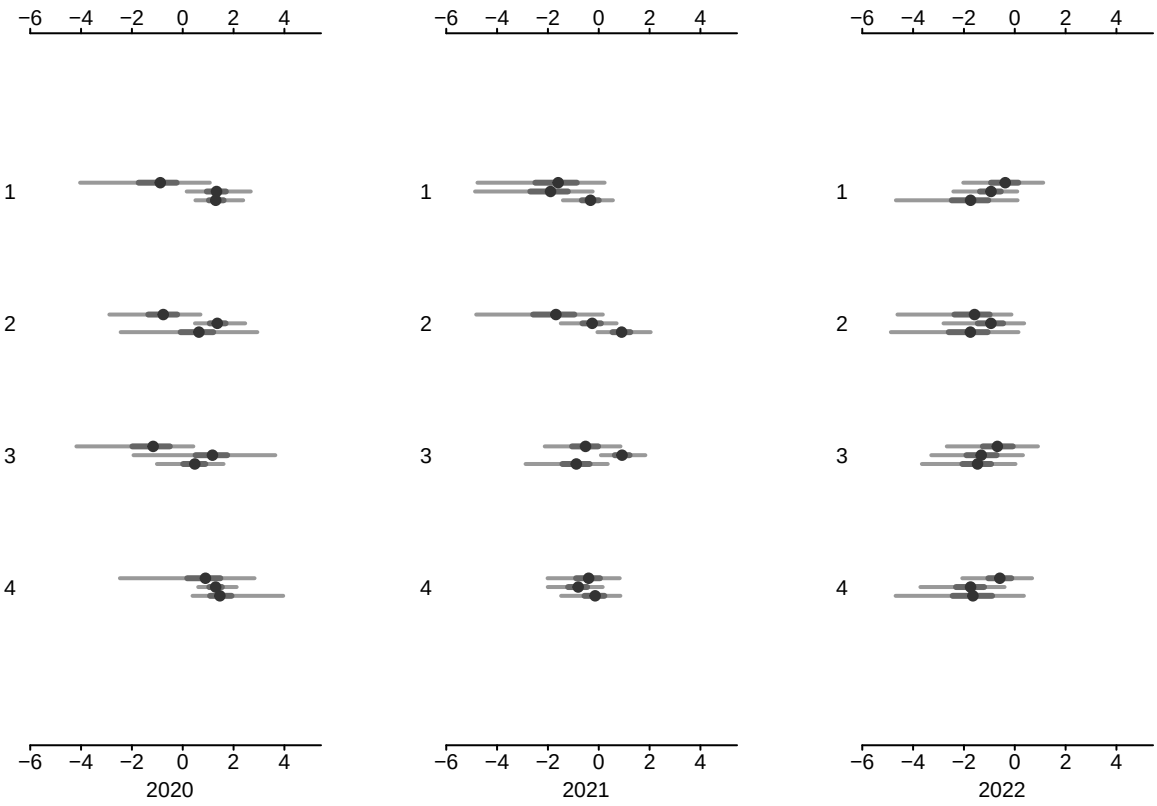

```
##abline(v=summary(musSn)$mean, lty=1:3)
##legend(x="topleft",lty=1:3, legend=c("2020","2020-2021","2020-2022"), bty="n")

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)
```



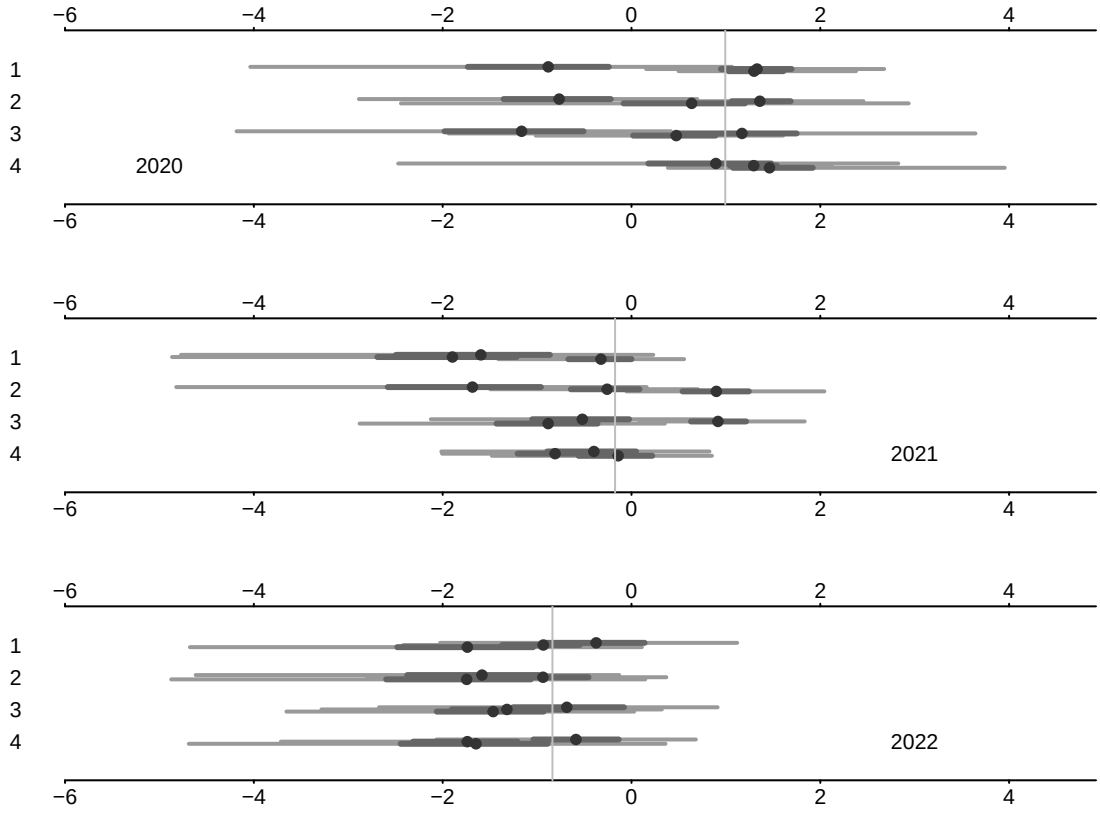

```
par(mar=c(3, 1, 1, 2), mgp=c(1.25,0.125,0), tck=-0.015)
mlplot(musNst, xlab="Segments by Year", xlim=c(-4.5, 3))
abline(v=summary(mNst)$mean, col=2:4)
```



```
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