Demo 1: Endangered tidewater gobies

First we will load the package:

```
# load the package
library(eDNAjoint)

# and other packages used in this workshop
library(tidyverse)
library(bayestestR)
library(patchwork)
library(wesanderson)
```

Prepare the data

Ensuring that your data is formatted correctly is essential for successfully using eDNAjoint. Let's first explore the structure of the goby data:

```
data(goby_data)
str(goby_data)
```

```
## List of 4
##  $ pcr_n   : num [1:39, 1:22] 6 6 6 6 6 6 6 6 6 6 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:22] "1" "2" "3" "4" ...
##  $ pcr_k   : num [1:39, 1:22] 0 0 0 0 6 0 0 0 0 5 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:22] "1" "2" "3" "4" ...
##  $ count   : int [1:39, 1:22] 0 0 0 0 0 0 0 0 0 1 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:22] "1" "2" "3" "4" ...
##  $ site_cov: num [1:39, 1:5] -0.711 -0.211 -1.16 -0.556 -0.988 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:5] "Salinity" "Filter_time" "Other_fishes" "Hab_size" ...
```

The input data is a list of matrices, where the rows in all matrices correspond to the number of sites.

```
names(goby_data)
```

```
## [1] "pcr_n"    "pcr_k"    "count"    "site_cov"
```

Count data

Let's look at the dimensions of the count data:

```
dim(goby_data$count)
```

```
## [1] 39 22
```

Number of primary samples (sites) = 39 Maximum number of secondary samples (replicates per site) = 22

These are the number of gobies in each seine sample, at each site.

```
head(goby_data$count)
```

```
##      1 2 3  4  5  6  7   8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
## [1,] 0 0 0  0  0  0  0   0  0  0  0 NA NA NA NA NA NA NA NA NA NA NA
## [2,] 0 0 0  0  0  0  0   0  0  0  0 NA NA NA NA NA NA NA NA NA NA NA
## [3,] 0 0 0  0  0  0  0   0  0  0  0 NA NA NA NA NA NA NA NA NA NA NA
## [4,] 0 4 1  0  2  1 38 112  1 15 NA NA NA NA NA NA NA NA NA NA NA NA
## [5,] 0 0 0  2  0  0  0   0  0  0  0  0  4  1  0  2  0  8 NA NA NA NA
## [6,] 0 0 0 NA NA NA NA  NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
```

Because we are building these matrices based on the maximum dimensions, we fill in the matrix with NAs for samples that don't exist. For example, at site 1, there were only 11 seine replicates, so columns 12-22 in row 1 are NA.

Site 11 has the maximum number of replicate observations (22):

```
goby_data$count[11,]
```

```
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
##  58  44  54   7  27  49  56   7  27  15   0   0   4   2   5 217   0   0   0   0
##  21  22
##   0  14
```

eDNA (PCR) data

Next let's look at the PCR data. `pcr_n` is the number of PCR replicates, per eDNA secondary sample (water sample), per site. These are the PCR "attempts".

```
head(goby_data$pcr_n)
```

```
##      1 2 3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
## [1,] 6 6 6  6  6  6  6  6  6  6  6 NA NA NA NA NA NA NA NA NA NA NA
## [2,] 6 6 6  6  6  6  6  6  6  6  6 NA NA NA NA NA NA NA NA NA NA NA
## [3,] 6 6 6  6  6  6  6  6  6  6  6 NA NA NA NA NA NA NA NA NA NA NA
## [4,] 6 6 6  6  6  6  6  6  6  6 NA NA NA NA NA NA NA NA NA NA NA NA
## [5,] 6 6 6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6 NA NA NA NA
## [6,] 6 6 6 NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
```

How many PCR replicates were there at site 4, water sample 6?

```
as.numeric(goby_data$pcr_n[4, 6])
```

```
## [1] 6
```

Again, site 11 had the maximum number of water samples:

```
goby_data$pcr_n[11,]
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
##  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6
```

`pcr_k` is the number of PCR successes. Of the number of PCR replicates, how many were detections were there?

```
head(goby_data$pcr_k)
```

```
##      1 2 3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
## [1,] 0 0 0  0  0  0  0  0  0  0  0 NA NA NA NA NA NA NA NA NA NA NA
## [2,] 0 0 0  0  0  0  0  0  0  0  0 NA NA NA NA NA NA NA NA NA NA NA
## [3,] 0 0 0  0  0  0  0  0  0  0  0 NA NA NA NA NA NA NA NA NA NA NA
## [4,] 0 6 6  4  6  5  4  6  5  3 NA NA NA NA NA NA NA NA NA NA NA NA
## [5,] 6 6 4  6  6  6  5  4  2  2  0  6  5  5  6  6  6  5  5  4 NA NA
## [6,] 0 0 0 NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
```

How many PCR detections (successes) were there at site 4, water sample 6?

```
as.numeric(goby_data$pcr_k[4, 6])
```

```
## [1] 5
```

A few checks to make sure the data is structured correctly:

The locations of the NA should be the same in the `pcr_k` and `pcr_n` matrices.

```
all(which(is.na(goby_data$pcr_k)) == which(is.na(goby_data$pcr_n)))
```

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

Both PCR and count data should have the same number of sites (i.e., number of rows):

```
all.equal(nrow(goby_data$pcr_k), nrow(goby_data$pcr_n), nrow(goby_data$count))
```

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

Site-level covariate data

Site-level covariate data is totally optional! PCR and count data are the minimum.

```
head(goby_data$site_cov)
```

```
##        Salinity Filter_time Other_fishes   Hab_size Veg
## [1,] -0.7114925       -1.17   -0.4738419 -0.2715560   0
## [2,] -0.2109183       -1.24   -0.4738419 -0.2663009   0
## [3,] -1.1602831       -1.29   -0.4738419 -0.2717707   0
## [4,] -0.5561419        0.11    0.5479118 -0.2164312   1
## [5,] -0.9876713       -0.70    0.2437353  4.9981956   1
## [6,]  1.2562818       -0.55   -0.3512823 -0.2934710   0
```

Notice that the continuous data is normalized:

```
mean(goby_data$site_cov[, "Salinity"])
```

```
## [1] -3.974359e-09
```

```
sd(goby_data$site_cov[, "Salinity"])
```

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

Fit the model

Now that we understand our data, let's fit the model using the function `joint_model`. The key arguments of this function include:

- `data`: list of pcr_k, pcr_n, count, and site_cov matrices
- `cov`: character vector of site-level covariates
- `family`: probability distribution used to model the seine count data. Options include a poisson, negative binomial, and gamma
- `p10_priors`: Beta distribution parameters for the prior on the probability of false positive eDNA detection, p10. c(1,20) is the default specification. More on this later.
- `q`: logical value indicating the presence of multiple traditional gear types.

```
# run the joint model with two covariates
goby_fit_cov1 <- joint_model(
  data = goby_data, # data
  cov = c("Filter_time", "Salinity"), # site-level covariates
  family = "poisson", # distribution for traditional data
  p10_priors = c(1, 20), # specify prior distribution for p10
  q = FALSE, # only one traditional gear type
  multicore = TRUE # run MCMC chains in parallel
)
```

```
## Refer to the eDNAjoint guide for visualization tips:  https://ednajoint.netlify.app/tips#visualizati
```

There are many more arguments, including more to customize the MCMC algorithm, but these are the primary arguments.

Now let's look at the return object:

```
names(goby_fit_cov1)
```

```
## [1] "model" "inits"
```

The first element of this list is the model, of class `stanfit`. This model object can be used with functions in the `rstan` package.

```
class(goby_fit_cov1$model)
```

```
## [1] "stanfit"
## attr(,"package")
## [1] "rstan"
```

And the second element of the list are the initial values used to start the MCMC. And this will indicate the initial values used for each of the four MCMC chains.

```
length(goby_fit_cov1$inits)
```

```
## [1] 4
```

Model selection

In the first model, we used salinity and filter time as our site level covariates. Perhaps we want to see how this model compares to models that include other covariates.

Here we will include the binary variable that indicates vegetation presence:

```
# fit a new model with one site-level covariate
goby_fit_cov2 <- joint_model(data = goby_data,
                             cov = "Veg",
                             family = "poisson",
                             p10_priors = c(1, 20),
                             q = FALSE,
                             multicore = TRUE,
                             verbose = FALSE # don't print messages!
                            )
```

```
## Refer to the eDNAjoint guide for visualization tips:  https://ednajoint.netlify.app/tips#visualizati
```

And now we will use the `joint_select` function to compare our two models. We will input our two models as a list:

```
# perform model selection
joint_select(model_fits = list(goby_fit_cov1$model, goby_fit_cov2$model))
```

```
##        elpd_diff se_diff
## model1   0.0       0.0
## model2 -43.0      29.3
```

This is performing model selection with leave-one-out cross validation and is a measure of how well a model predicts new, unseen data. A higher ELPD (Expected Log Predictive Density) value indicates better predictive performance, meaning the model is better at predicting new data.

You could test this with all the covariates, but we will stick with model 1 (salinity and filter time) for now.

Summarize the posteriors

The model fit object contains our posterior samples, and we can use the `joint_summarize` function to create summaries of the distributions.

First let's look at the `p10` parameter:

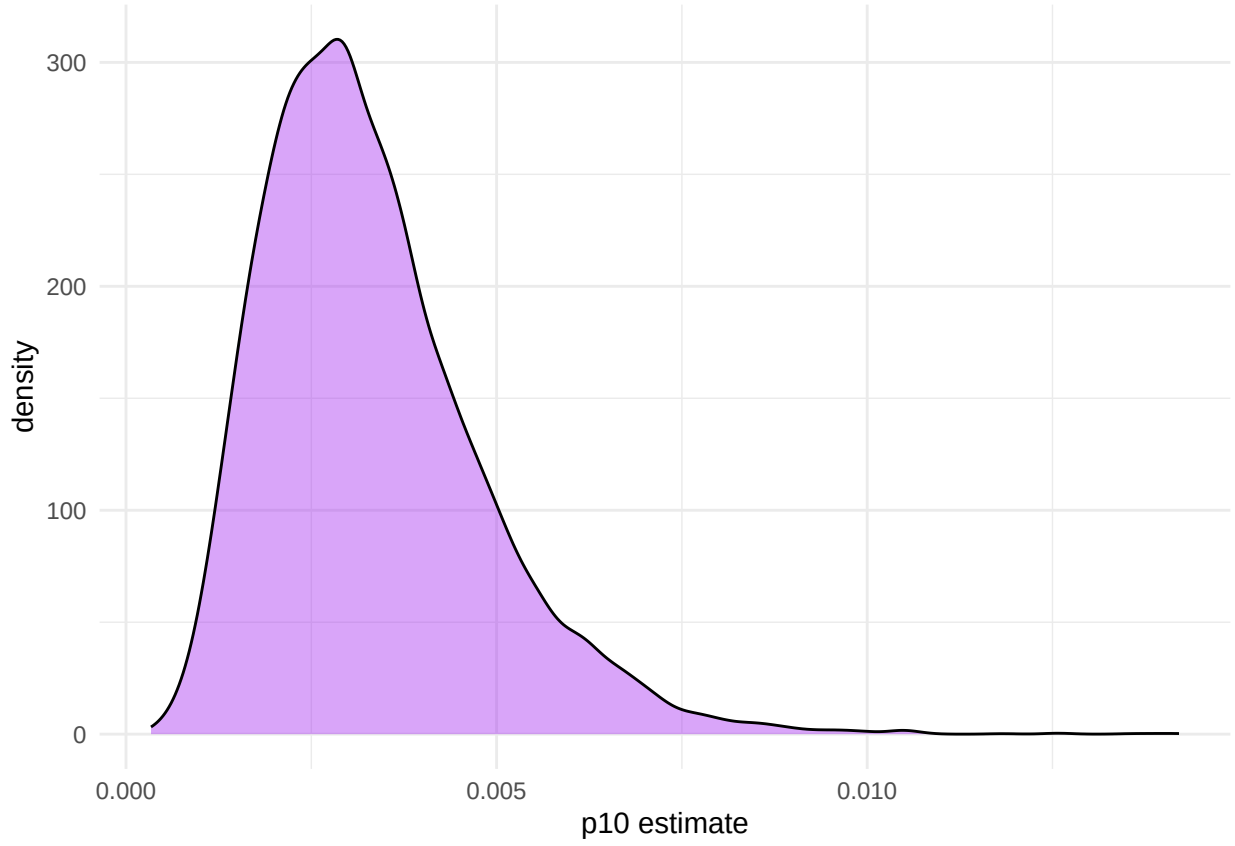
```
joint_summarize(goby_fit_cov1$model, par = "p10")
```

```
##      mean se_mean    sd  2.5% 97.5%    n_eff Rhat
## p10 0.003       0 0.001 0.001 0.007 16934.95    1
```

Here, we see the mean, standard deviation, lower and upper bounds of the 95% credibility interval, the effective sample size, and Rhat.

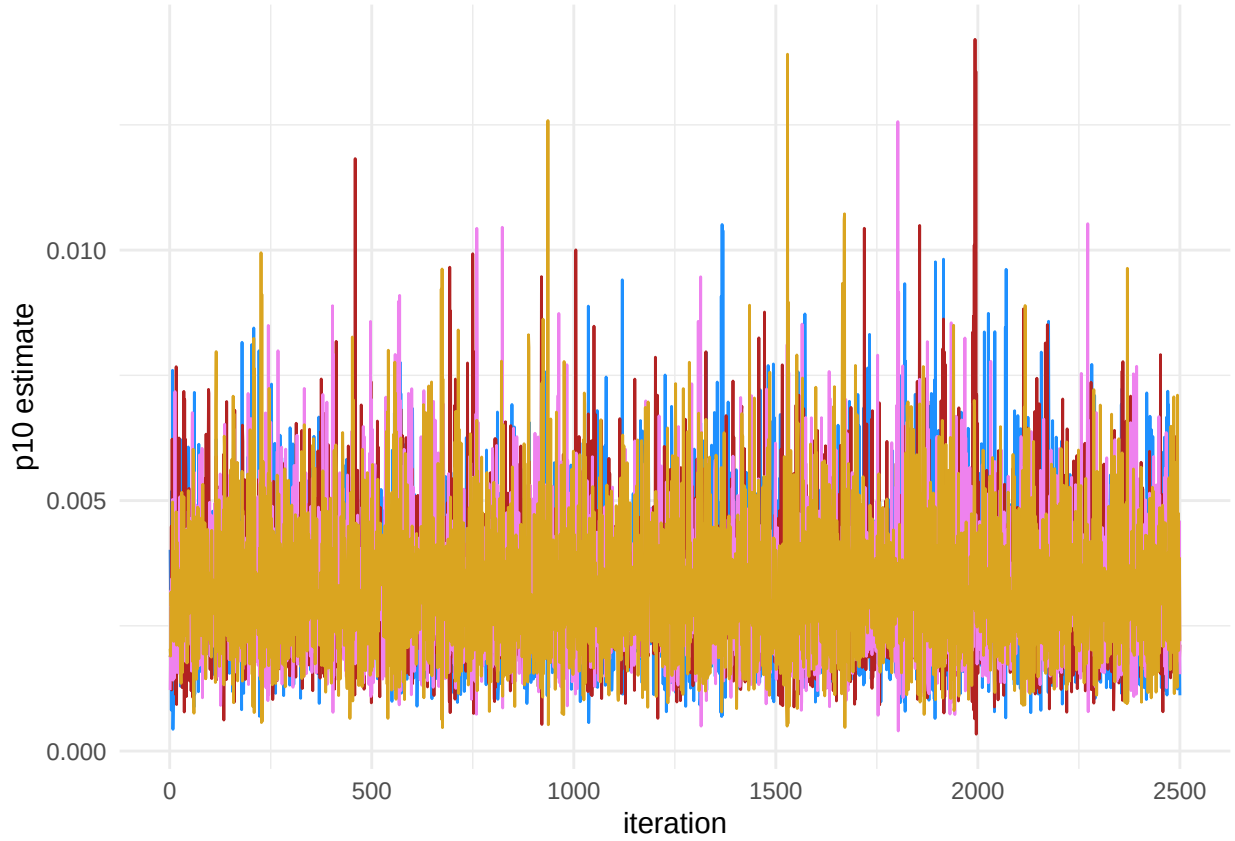
Let's plot this marginal posterior as a density plot:

```
ggplot() +
  geom_density(aes(x = as.matrix(goby_fit_cov1$model)[, "p10"]),
               fill = "purple", alpha = 0.4) +
  labs(x = "p10 estimate", y = "density") +
  theme_minimal()
```



Let's plot the traceplot of `p10`:

```
# get chains for param p10
chain1_p10 <- as.array(goby_fit_cov1$model)[, "chain:1", "p10"]
chain2_p10 <- as.array(goby_fit_cov1$model)[, "chain:2", "p10"]
chain3_p10 <- as.array(goby_fit_cov1$model)[, "chain:3", "p10"]
chain4_p10 <- as.array(goby_fit_cov1$model)[, "chain:4", "p10"]

ggplot() +
  geom_line(aes(x = 1:length(chain1_p10), y = chain1_p10),
            color = "dodgerblue") +
  geom_line(aes(x = 1:length(chain2_p10), y = chain2_p10),
            color = "firebrick") +
  geom_line(aes(x = 1:length(chain3_p10), y = chain3_p10),
            color = "violet") +
  geom_line(aes(x = 1:length(chain4_p10), y = chain4_p10),
            color = "goldenrod") +
  labs(x = "iteration", y = "p10 estimate") +
  theme_minimal()
```



Now let's look at the summaries of `mu`, which is the expected catch rate at each site:

```
joint_summarize(goby_fit_cov1$model, par = "mu")
```

```
##           mean se_mean    sd    2.5%   97.5%    n_eff Rhat
## mu[1]    0.009   0.000 0.010   0.000   0.035 14647.51    1
## mu[2]    0.008   0.000 0.008   0.000   0.029 15460.58    1
## mu[3]    0.010   0.000 0.010   0.000   0.036 17021.03    1
## mu[4]   16.703   0.008 1.259  14.363  19.274 21994.12    1
## mu[5]    2.052   0.002 0.281   1.548   2.647 17954.94    1
## mu[6]    0.035   0.000 0.038   0.001   0.138 15574.17    1
## mu[7]    0.254   0.001 0.080   0.123   0.435 17091.37    1
## mu[8]    0.655   0.005 0.671   0.014   2.469 15127.95    1
## mu[9]    0.057   0.000 0.059   0.001   0.215 14886.53    1
## mu[10]   1.163   0.001 0.177   0.847   1.539 15998.69    1
## mu[11]  26.878   0.007 1.091  24.803  29.038 22306.31    1
## mu[12]   0.016   0.000 0.017   0.000   0.061 16623.79    1
## mu[13]   0.006   0.000 0.007   0.000   0.024 13925.00    1
## mu[14]   0.055   0.000 0.055   0.002   0.203 16481.07    1
## mu[15]   0.030   0.000 0.021   0.004   0.084 13025.07    1
## mu[16]   0.027   0.000 0.022   0.002   0.082 15734.10    1
## mu[17]   0.014   0.000 0.015   0.000   0.055 15808.64    1
## mu[18]   0.029   0.000 0.029   0.001   0.109 16649.71    1
## mu[19]   0.092   0.001 0.092   0.002   0.344 16347.52    1
## mu[20]   0.268   0.001 0.096   0.116   0.488 20645.22    1
## mu[21]   0.142   0.001 0.146   0.003   0.532 16532.73    1
```

```
## mu[22]   0.412   0.001 0.186   0.133   0.846 19858.69    1
## mu[23]  90.747   0.021 3.207  84.594  97.155 23364.33    1
## mu[24]   1.681   0.002 0.312   1.138   2.347 19770.07    1
## mu[25]  20.881   0.010 1.636  17.834  24.204 24401.65    1
## mu[26]   1.829   0.002 0.303   1.287   2.483 19144.20    1
## mu[27]   3.244   0.004 0.521   2.312   4.334 19338.47    1
## mu[28]   8.107   0.005 0.767   6.684   9.695 24437.88    1
## mu[29]   2.444   0.003 0.414   1.716   3.329 17544.22    1
## mu[30]  15.094   0.015 2.165  11.132  19.664 22034.19    1
## mu[31]  94.651   0.029 4.307  86.371 103.197 22587.02    1
## mu[32]   0.027   0.000 0.027   0.001   0.099 17080.46    1
## mu[33]   0.055   0.000 0.059   0.001   0.213 15521.59    1
## mu[34]   0.402   0.002 0.224   0.088   0.949 21009.08    1
## mu[35]   0.081   0.001 0.086   0.002   0.316 14344.53    1
## mu[36]   0.007   0.000 0.007   0.000   0.026 13532.05    1
## mu[37]   0.011   0.000 0.011   0.000   0.041 15074.06    1
## mu[38] 170.604   0.042 5.872 159.354 182.467 19736.80    1
## mu[39]   0.006   0.000 0.006   0.000   0.023 15153.44    1
```

Now on to interpreting the site-level covariates:

`beta` is the parameter in the model that scales the sensitivity of eDNA samples, relative to traditional samples, and it is site specific and a function of our covariates:

$$\beta_i = \alpha_1 + \alpha_2 \times FilterTime_i + \alpha_3 \times Salinity_i$$

As `beta` increases, the sensitivity of eDNA sampling decreases. So a positive regression coefficient would indicate an inverse relationship between the value of the covariate and the sensitivity of eDNA.

So let's look at the marginal posterior summaries of `alpha`:

```
joint_summarize(goby_fit_cov1$model, par = "alpha")
```

```
##             mean se_mean    sd   2.5%  97.5%    n_eff Rhat
## alpha[1]  0.543   0.001 0.099  0.349  0.734 11885.65    1
## alpha[2]  1.021   0.001 0.119  0.783  1.250 10915.72    1
## alpha[3] -0.349   0.001 0.108 -0.557 -0.136 11794.69    1
```

`alpha[1]` is the regression intercept and indicates the sensitivity of eDNA at the *average* site.

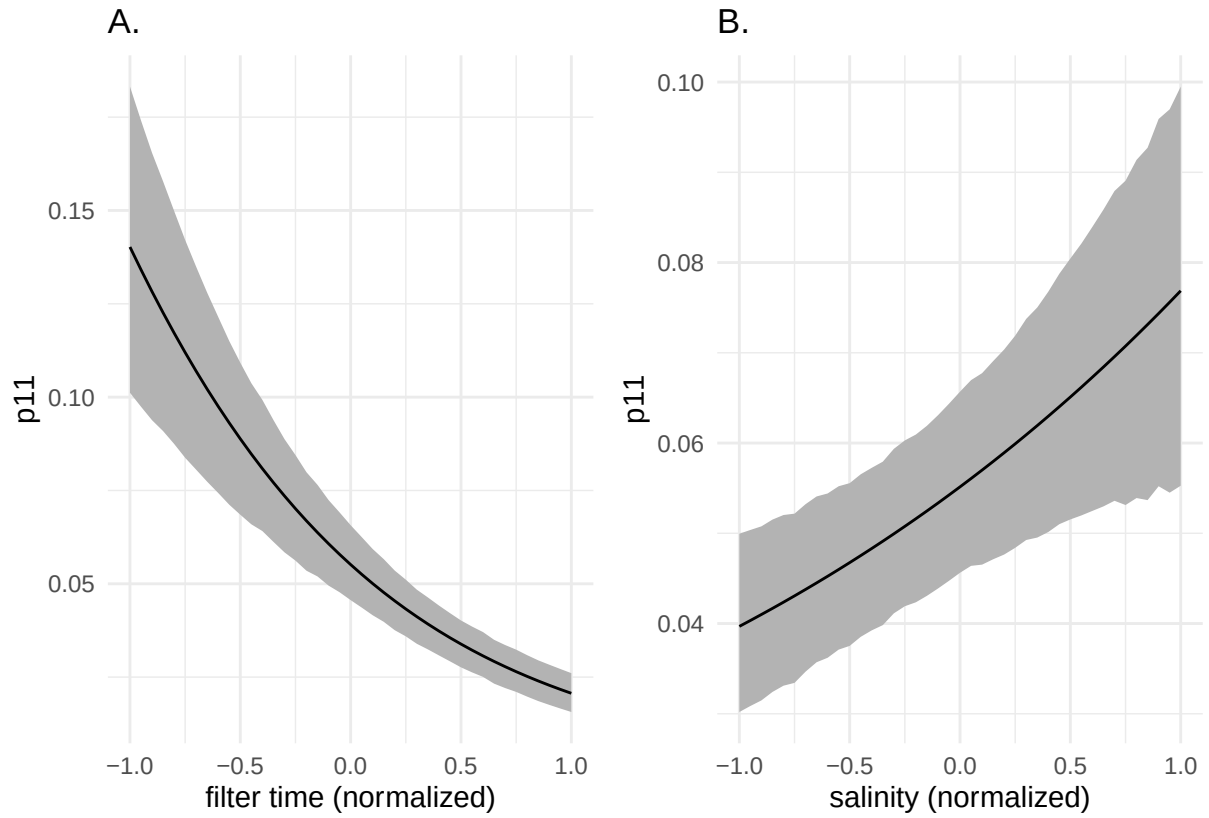
`alpha[2]` is the regression coefficient for filter time. Since this is positive, this means that at sites where the filter time is longer, the sensitivity of eDNA is lower.

`alpha[3]` is the regression coefficient for salinity. Since this is negative, this means that at sites with higher salinity, the sensitivity of eDNA is higher.

And we can go a little farther than this and use the posterior samples and model structure to see the relationship between the value of the covariate and of a true eDNA detection.

```
# get posterior samples of site-level covariate coefficients (alpha)
samples_alpha <- as.data.frame(rstan::extract(goby_fit_cov1$model,
                                              pars = "alpha")$alpha)
colnames(samples_alpha) <- c("alpha1", "alpha2", "alpha3")
# calculate p11, given mu = 0.1 at a range of covariate values
cov_val <- seq(-1, 1, 0.05)
```

```r
mu <- 0.1
# filter time
p11_filtertime <- as.data.frame(matrix(NA, nrow = length(cov_val), ncol = 4))
colnames(p11_filtertime) <- c("filtertime_z", "p11_mean", "p11_low", "p11_high")
for (i in seq_along(1:length(cov_val))) {
  beta <- as.matrix(samples_alpha) %*% c(1, cov_val[i], 0)
  p11 <- mu / (mu + exp(beta))
  p11_filtertime[i, ] <- as.vector(c(cov_val[i], mean(p11),
  ci(p11, method = "HDI")[2:3]))
}

# salinity
p11_salinity <- as.data.frame(matrix(NA, nrow = length(cov_val), ncol = 4))
colnames(p11_salinity) <- c("salinity_z", "p11_mean", "p11_low", "p11_high")
for (i in seq_along(1:length(cov_val))) {
  beta <- as.matrix(samples_alpha) %*% c(1, 0, cov_val[i])
  p11 <- mu / (mu + exp(beta))
  p11_salinity[i, ] <- as.vector(c(cov_val[i], mean(p11),
  ci(p11, method = "HDI")[2:3]))
}
# plot
filtertime_plot <- ggplot(data = p11_filtertime) +
  geom_ribbon(aes(x = filtertime_z,
                  ymin = p11_low, ymax = p11_high), fill = "grey70") +
  geom_line(aes(x = filtertime_z, y = p11_mean)) +
  labs(x = "filter time (normalized)", y = "p11") +
  theme_minimal() +
  ggtitle("A.")

salinity_plot <- ggplot(data = p11_salinity) +
  geom_ribbon(aes(x = salinity_z,
                  ymin = p11_low, ymax = p11_high), fill = "grey70") +
  geom_line(aes(x = salinity_z, y = p11_mean)) +
  labs(x = "salinity (normalized)", y = "p11") +
  theme_minimal() +
  ggtitle("B.")

filtertime_plot + salinity_plot + plot_layout(nrow = 1)
```



Compare detection rates

To compare the relative sensitivity of eDNA and traditional sampling, we can use `detection_calculate()` to find the units of survey effort necessary to detect presence of the species.

Here, detecting presence refers to producing at least one true positive eDNA detection or catching at least one individual in a traditional survey.

This is calculated for an average site:

$$\beta_i = \alpha_1 + \alpha_2 \times FilterTime_i + \alpha_3 \times Salinity_i$$

$$\beta_i = \alpha_1 + \alpha_2 \times 0 + \alpha_3 \times 0$$

```
detection_calculate(goby_fit_cov1$model, # model fit
                    mu = c(0.1, 0.5, 1), # expected catch rate
                    cov_val = c(0, 0), # mean covariate values
                    probability = 0.9) # probability of detection
```

```
##       mu n_traditional n_eDNA
## [1,] 0.1            24     14
## [2,] 0.5             5      4
## [3,] 1.0             3      2
```

How does this change if we do not calculate for a site with mean covariate values?

This is calculating the effort necessary to detect presence for a site with a filter time 0.5 z-scores above the mean:

$$\beta_i = \alpha_1 + \alpha_2 \times 0.5 + \alpha_3 \times 0$$

```
detection_calculate(goby_fit_cov1$model,
                    mu = c(0.1, 0.5, 1),
                    cov_val = c(0.5, 0), # filter time 0.5 z-scores above the mean
                    probability = 0.9)
```

```
##       mu n_traditional n_eDNA
## [1,] 0.1            24     23
## [2,] 0.5             5      5
## [3,] 1.0             3      3
```

Now let's do the same for salinity:

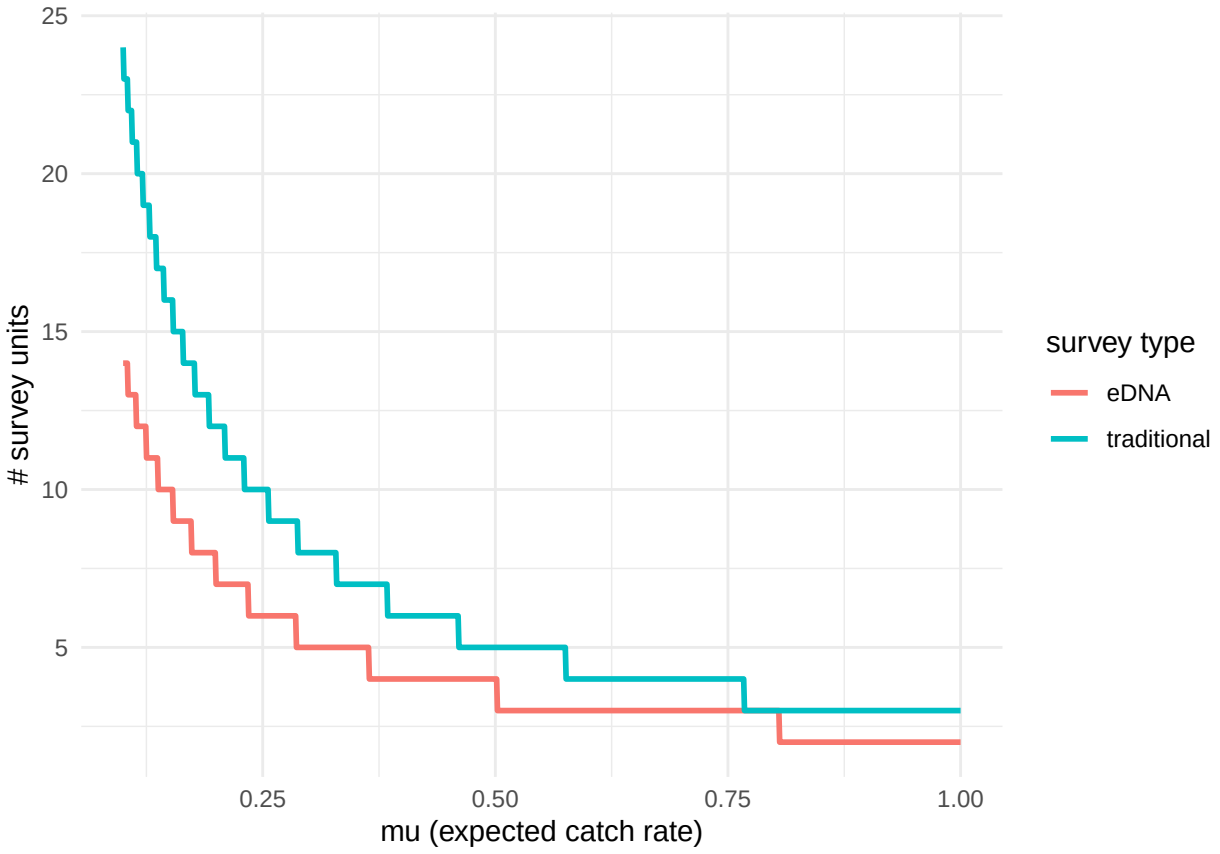
$$\beta_i = \alpha_1 + \alpha_2 \times 0 + \alpha_3 \times 0.5$$

```
detection_calculate(goby_fit_cov1$model,
                    mu = c(0.1, 0.5, 1),
                    cov_val = c(0, 0.5), # filter time 0.5 z-scores above the mean
                    probability = 0.9)
```

```
##       mu n_traditional n_eDNA
## [1,] 0.1            24     12
## [2,] 0.5             5      3
## [3,] 1.0             3      2
```

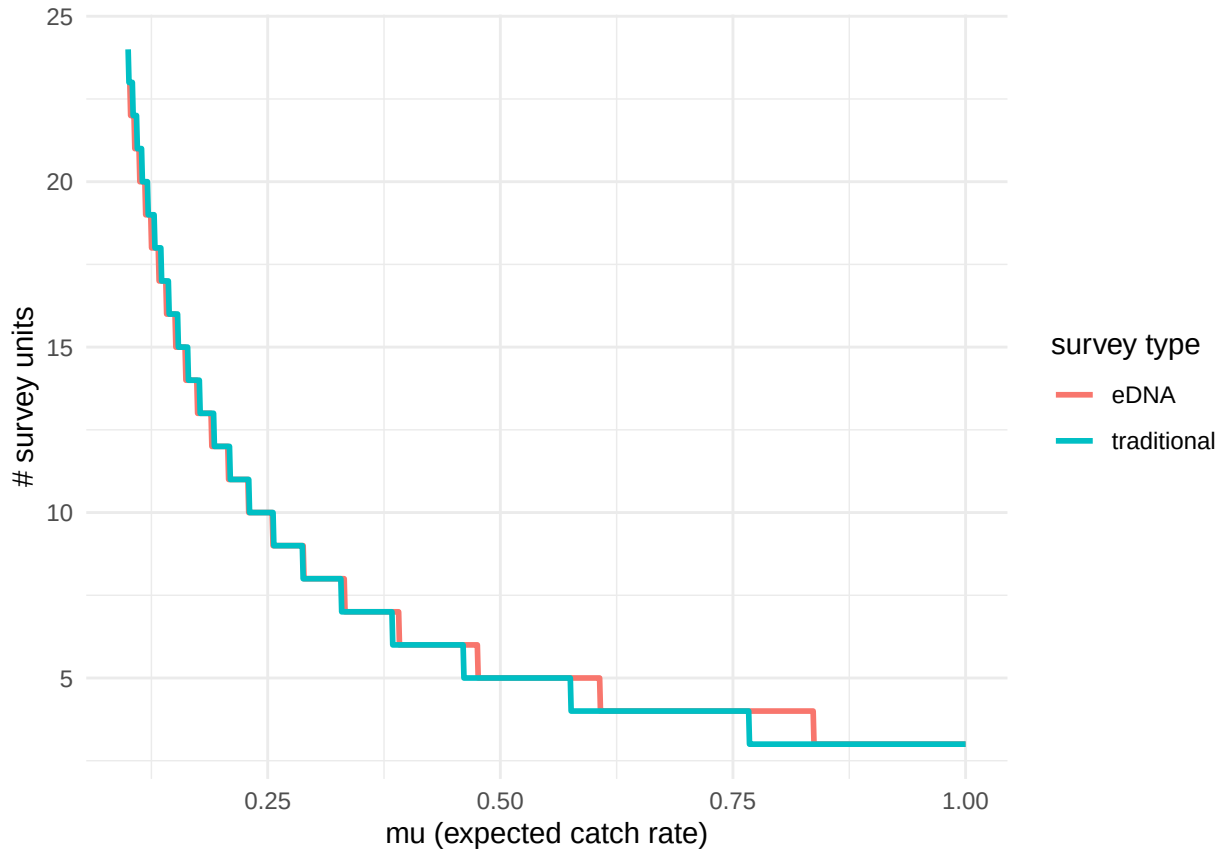
We can plot these comparisons as well with `detection_plot()`:

```
detection_plot(goby_fit_cov1$model,
               mu_min = 0.1, mu_max = 1, # min and max mu
               cov_val = c(0, 0), # mean covariate values
               probability = 0.9)
```



And now plot for a site with filter time 0.5 z-scores above the mean:

```
detection_plot(goby_fit_cov1$model,
               mu_min = 0.1, mu_max = 1, # min and max mu
               cov_val = c(0.5, 0), # high salinity
               probability = 0.9)
```



Prior sensitivity analysis

When we first fit our model, we used the default prior distribution for $p10$, the probability of a false positive eDNA detection:

```
# run the joint model with two covariates
goby_fit_cov1 <- joint_model(
  data = goby_data,
  cov = c("Filter_time", "Salinity"),
  family = "poisson",
  p10_priors = c(1, 20), # specify prior distribution for p10
  q = FALSE,
  multicore = TRUE
)
```

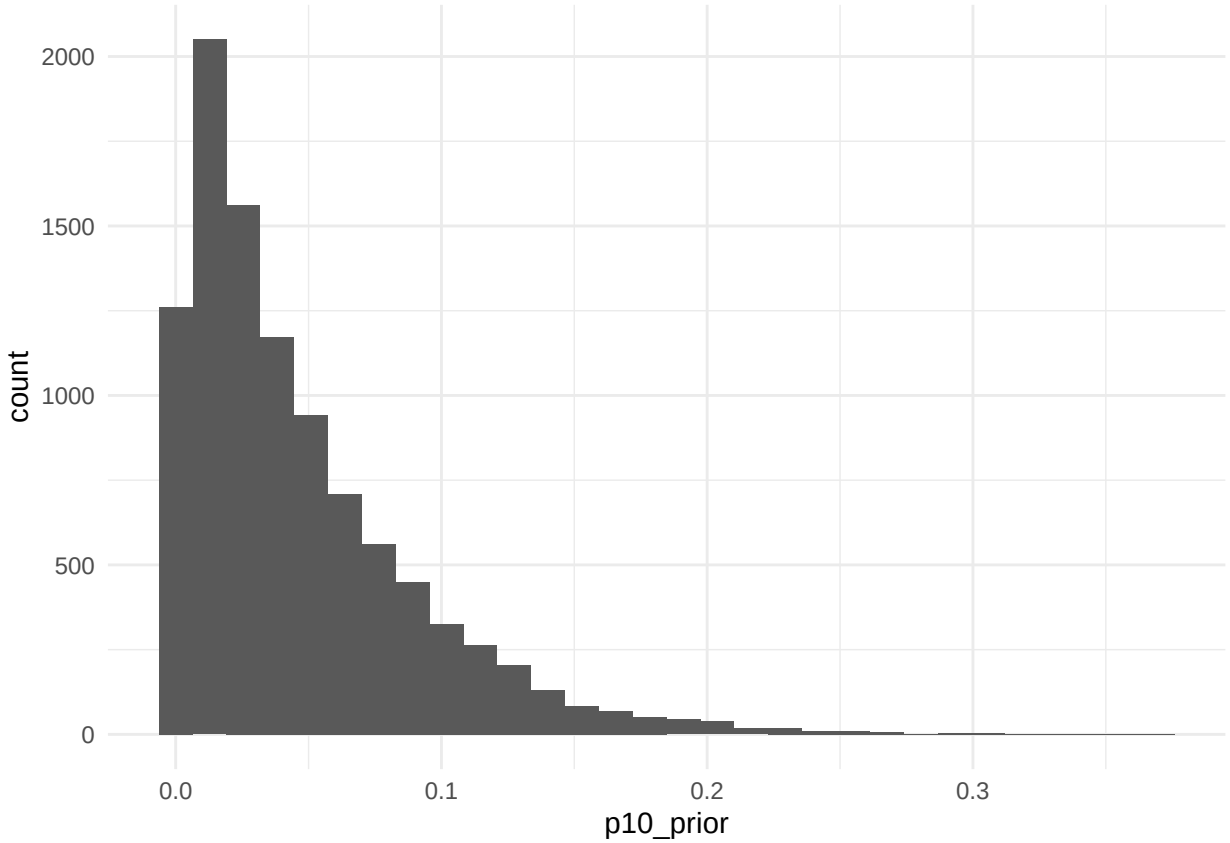
```
## Refer to the eDNAjoint guide for visualization tips:  https://ednajoint.netlify.app/tips#visualizati
```

These are the shape parameters of a beta distribution that reflects our prior belief about the probability of a false positive. And this prior belief is updated with our data to give use our posterior estimate.

The default parameters give us a prior for p10 that is relatively uninformative:

```r
p10_prior <- rbeta(10000, shape1 = 1, shape2 = 20)

ggplot() +
  geom_histogram(aes(x = p10_prior)) +
  theme_minimal()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



With this prior, 99% of the probability density is less than 0.2.

```r
1 - qbeta(0.2, shape1 = 1, shape2 = 20)
```

```
## [1] 0.9889048
```

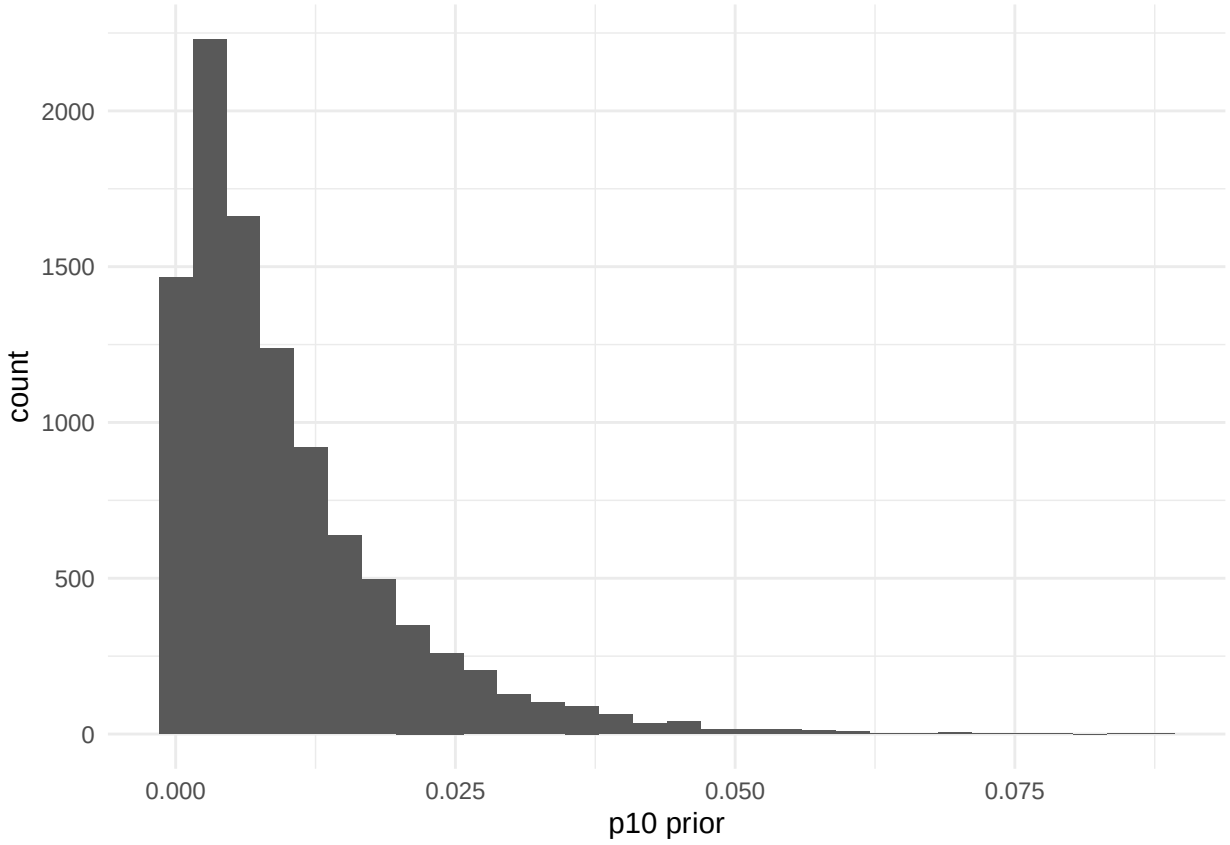
We could keep this prior relatively uninformative, or we could use data to create the prior, perhaps from negative controls.

For example, let's say you used 25 negative qPCR controls during eDNA data processing. You can assume that the probability of a false positive is less than $1/25 = 0.04$. To reflect this data, you could create a prior where the $probability(p10 > 0.04)$ is low:

```r
p10_negative_control <- rbeta(n = 10000, shape1 = 1, shape2 = 100)

ggplot() +
```

```
  geom_histogram(aes(x = p10_negative_control)) +
  labs(x = "p10 prior", y = "count") +
  theme_minimal()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



With an informative prior, it is good practice to do a prior sensitivity analysis. Here, we are testing how sensitive our posterior inference is to the choice of prior.

If the posterior is not sensitive to the prior, this means that the posterior is more driven by the information in the data, rather than by the prior.

So we could run our model again with different priors:

```
fit_prior2 <- joint_model(data = goby_data, family = "poisson",
                          cov = c("Filter_time", "Salinity"),
                          p10_priors = c(1, 15), # new prior
                          q = FALSE, verbose = FALSE,
                          multicore = TRUE)
```

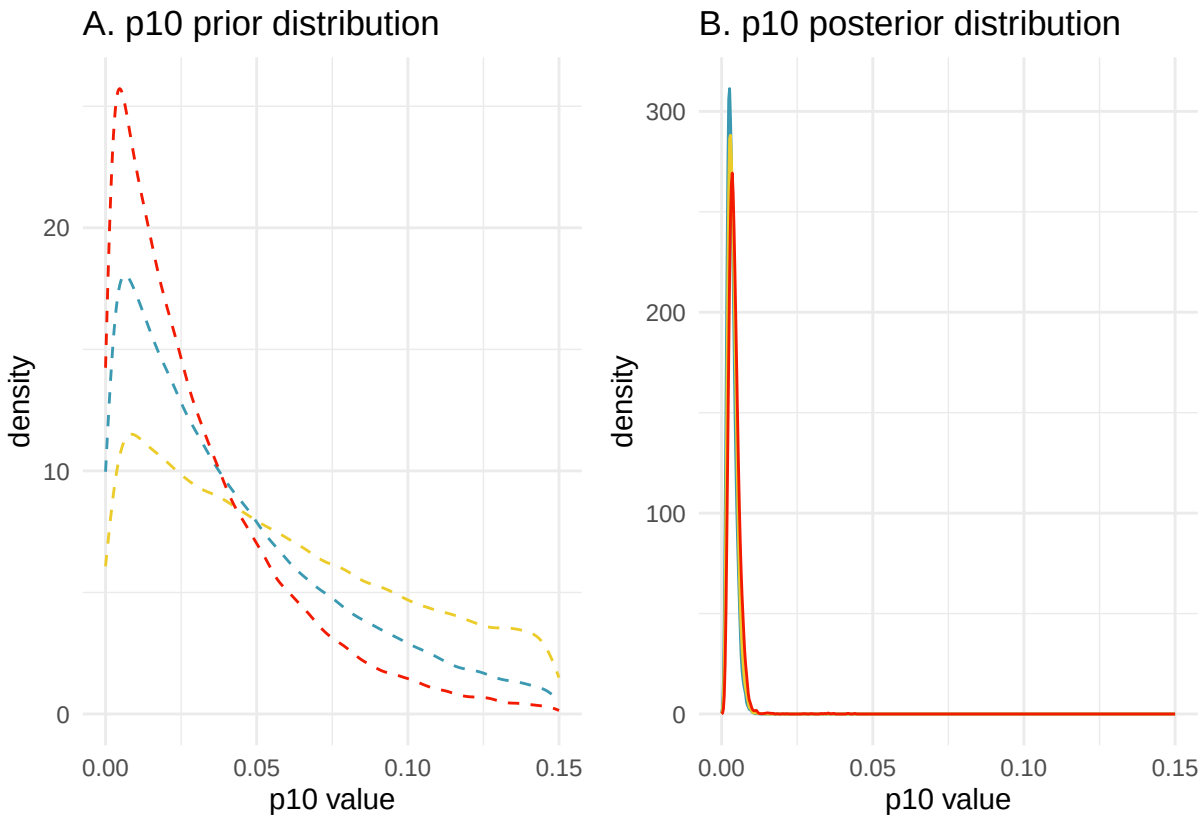
```
## Refer to the eDNAjoint guide for visualization tips:  https://ednajoint.netlify.app/tips#visualizati
```

```
fit_prior3 <- joint_model(data = goby_data, family = "poisson",
                          cov = c("Filter_time", "Salinity"),
                          p10_priors = c(1, 10), # new prior
```

```
                          q = FALSE, verbose = FALSE,
                          multicore = TRUE)
```

```
## Warning: There were 21 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: There were 2479 transitions after warmup that exceeded the maximum treedepth. Increase max_

## https://mc-stan.org/misc/warnings.html#maximum-treedepth-exceeded

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

## Warning: The largest R-hat is 1.15, indicating chains have not mixed.
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#r-hat

## 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

## Refer to the eDNAjoint guide for visualization tips:  https://ednajoint.netlify.app/tips#visualizati
```

```
# get samples
samples_20 <- rstan::extract(goby_fit_cov1$model, pars = "p10")$p10
samples_10 <- rstan::extract(fit_prior2$model, pars = "p10")$p10
samples_30 <- rstan::extract(fit_prior3$model, pars = "p10")$p10

# plot
prior_plot <- ggplot() +
  geom_density(aes(x = rbeta(100000, 1, 20)), color = wes_palette("Zissou1")[1],
               linetype = "dashed") + # blue
  geom_density(aes(x = rbeta(100000, 1, 10)), color = wes_palette("Zissou1")[3],
               linetype = "dashed") + # yellow
  geom_density(aes(x = rbeta(100000, 1, 30)), color = wes_palette("Zissou1")[5],
               linetype = "dashed") + # red
  scale_x_continuous(limits = c(0, 0.15)) +
  labs(x = "p10 value", y = "density") +
  ggtitle("A. p10 prior distribution") +
  theme_minimal()

posterior_plot <- ggplot() +
  geom_density(aes(x = samples_20), color = wes_palette("Zissou1")[1]) +
  geom_density(aes(x = samples_10), color = wes_palette("Zissou1")[3]) +
  geom_density(aes(x = samples_30), color = wes_palette("Zissou1")[5]) +
  scale_x_continuous(limits = c(0, 0.15)) +
  labs(x = "p10 value", y = "density") +
  ggtitle("B. p10 posterior distribution") +
  theme_minimal()

prior_plot + posterior_plot + plot_layout(ncol = 2)
```


A. p10 prior distribution
B. p10 posterior distribution
density
p10 value
0
10
20
100
200
300
0.00
0.05
0.10
0.15