eDNAjoint demo: Endangered tidewater gobies

First we will load the package:

```r
# 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:

```r
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.

```r
names(goby_data)
```

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

Count data

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

```r
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.

```r
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):

```r
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".

```r
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  6  6 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:

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

```
## [1] 0
```

```
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:

```r
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.

```r
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.

```r
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:

```r
# 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:

```r
# 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 -37.6      31.8  
```

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 posterior

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:

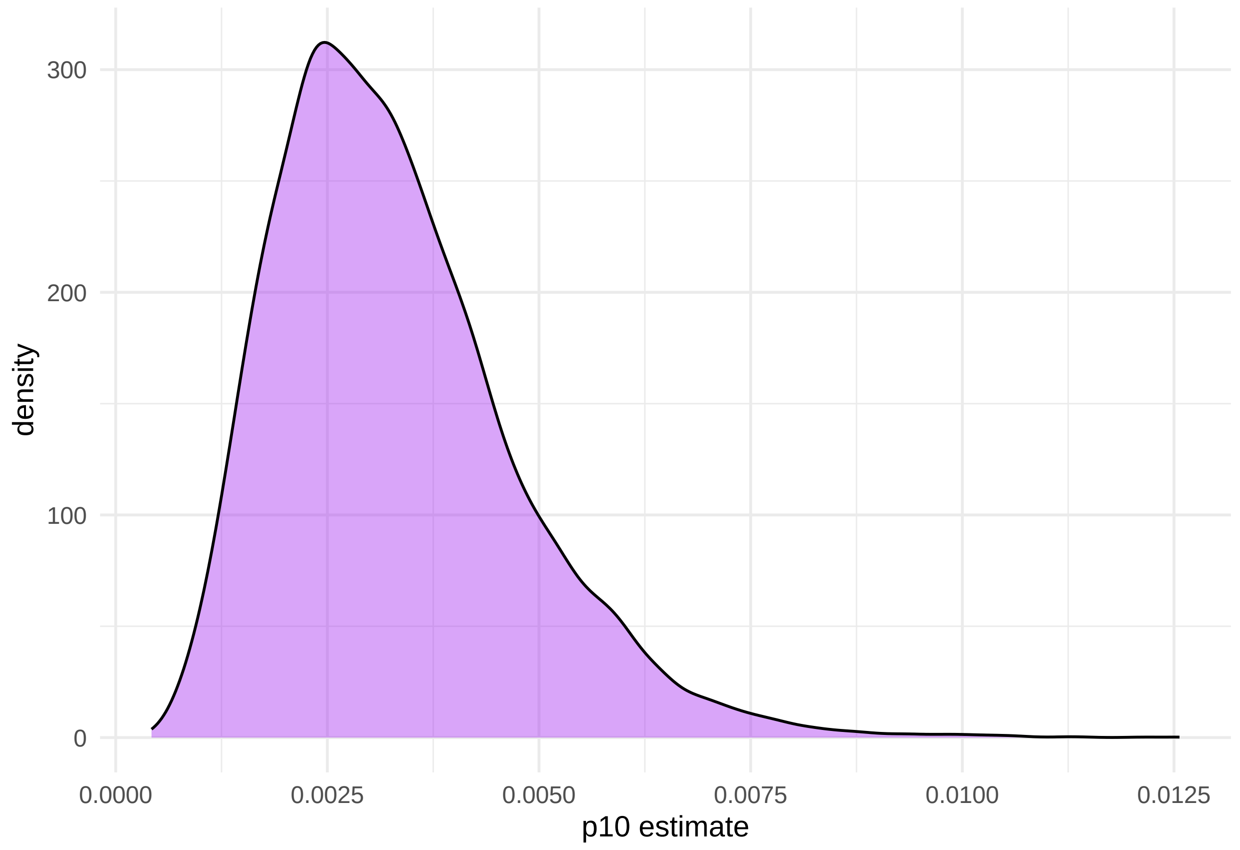
```
joint_summarize(goby_fit_cov1$model, par = "p10")
```

```
##      mean se_mean    sd  2.5% 97.5%    n_eff Rhat
## p10 0.003       0 0.001 0.001 0.007 13727.57    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:

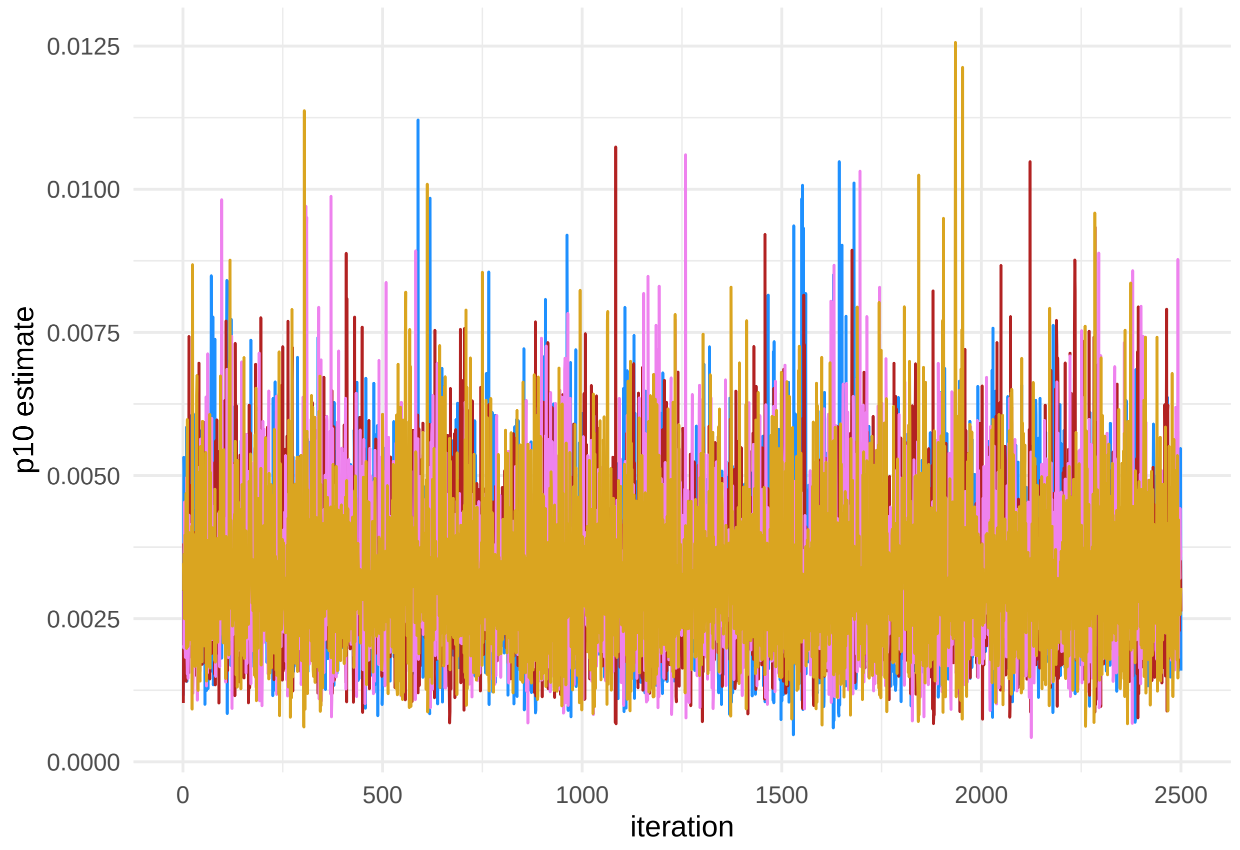
```
ggplot() +
  geom_density(aes(x = as.matrix(goby_fit_cov1$model)[, "p10"]),
               fill = "purple", alpha = 0.4) +
  labs(x = "p10 estimate", y = "density") +
  theme_minimal()
```



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:

```r
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 15045.90    1
## mu[2]     0.008   0.000 0.008   0.000   0.029 14237.78    1
## mu[3]     0.010   0.000 0.010   0.000   0.037 15373.85    1
## mu[4]    16.694   0.009 1.258  14.289  19.242 21872.51    1
## mu[5]     2.052   0.002 0.283   1.534   2.650 16754.79    1
## mu[6]     0.035   0.000 0.039   0.001   0.140 14651.91    1
## mu[7]     0.254   0.001 0.082   0.122   0.443 14440.27    1
## mu[8]     0.653   0.006 0.679   0.015   2.480 14475.40    1
## mu[9]     0.057   0.001 0.060   0.001   0.217 13910.80    1
## mu[10]    1.161   0.001 0.180   0.838   1.543 15915.62    1
## mu[11]   26.877   0.008 1.117  24.784  29.110 19134.52    1
## mu[12]    0.016   0.000 0.016   0.000   0.059 12882.30    1
## mu[13]    0.006   0.000 0.006   0.000   0.024 15219.90    1
## mu[14]    0.055   0.000 0.058   0.001   0.215 16672.98    1
## mu[15]    0.030   0.000 0.021   0.005   0.085 14629.92    1
## mu[16]    0.027   0.000 0.022   0.002   0.083 12036.59    1
## mu[17]    0.014   0.000 0.014   0.000   0.052 15098.67    1
## mu[18]    0.029   0.000 0.030   0.001   0.111 16190.91    1
## mu[19]    0.094   0.001 0.096   0.002   0.349 14827.26    1
## mu[20]    0.267   0.001 0.097   0.114   0.489 16728.57    1
## mu[21]    0.142   0.001 0.141   0.004   0.518 16921.51    1
## mu[22]   0.412   0.001 0.187   0.131   0.852 16746.49    1
## mu[23]  90.794   0.024 3.143  84.690  97.140 17682.30    1
## mu[24]   1.677   0.003 0.320   1.118   2.360 15787.26    1
## mu[25]  20.863   0.012 1.600  17.868  24.077 17155.26    1
## mu[26]   1.829   0.002 0.302   1.282   2.464 19351.61    1
## mu[27]   3.247   0.004 0.528   2.308   4.364 16379.45    1
## mu[28]   8.109   0.005 0.761   6.680   9.698 19737.29    1
## mu[29]   2.439   0.004 0.408   1.713   3.296 12896.47    1
## mu[30]  15.115   0.015 2.170  11.256  19.659 21955.60    1
## mu[31]  94.553   0.029 4.348  86.201 103.327 22057.60    1
## mu[32]   0.028   0.000 0.029   0.001   0.104 14756.88    1
## mu[33]   0.056   0.000 0.060   0.001   0.220 14865.52    1
## mu[34]   0.396   0.002 0.211   0.089   0.909 13449.08    1
## mu[35]   0.081   0.001 0.087   0.002   0.310 15089.82    1
## mu[36]   0.006   0.000 0.007   0.000   0.025 13464.41    1
## mu[37]   0.011   0.000 0.011   0.000   0.041 13139.91    1
## mu[38] 170.503   0.043 5.876 159.238 182.239 18267.41    1
## mu[39]   0.006   0.000 0.006   0.000   0.023 13819.88    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  0.348  0.737 10905.97    1
## alpha[2]   1.022   0.001 0.118  0.782  1.249 10907.63    1
## alpha[3]  -0.350   0.001 0.107 -0.556 -0.141 12323.10    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.

Compare detection rates

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

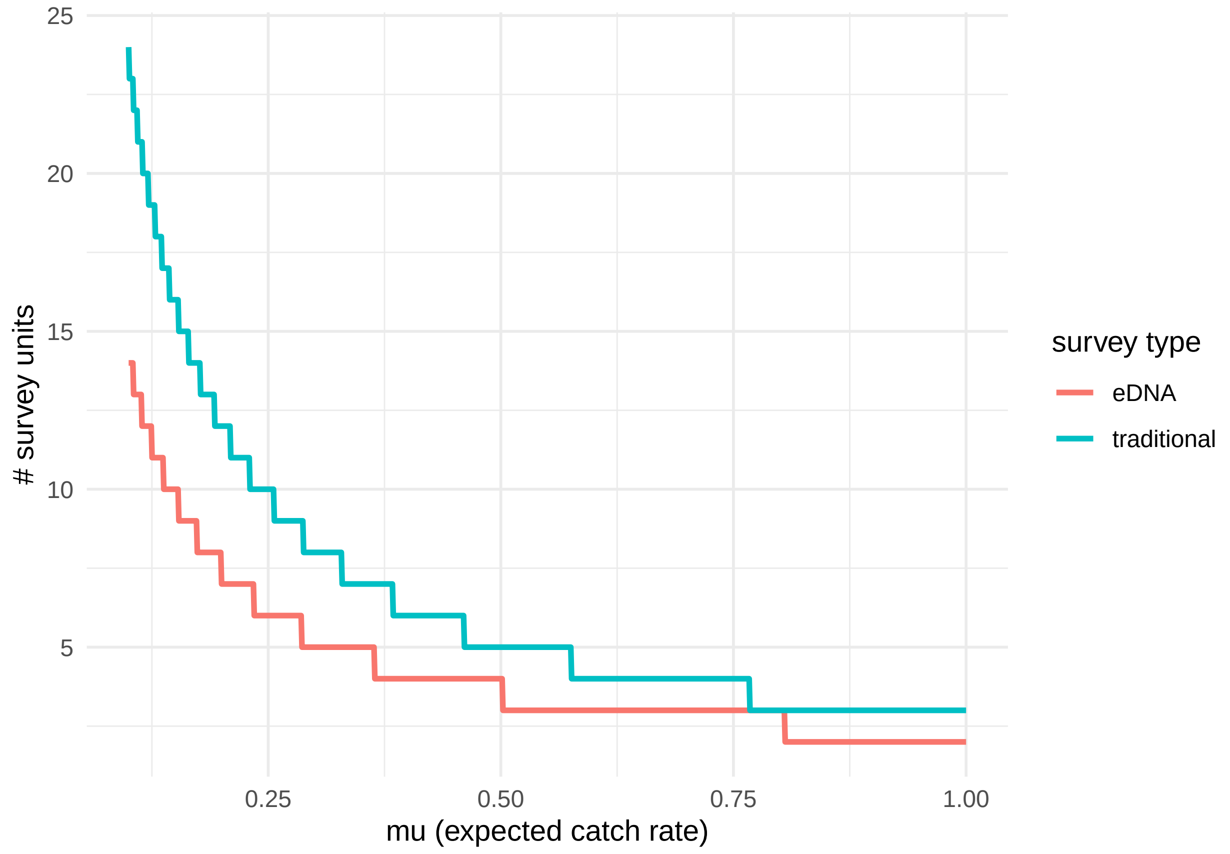
Here, detecting presence refers to producing at least one true positive eDNA detection or catching at least one individual in a traditional survey.

This is calculated for an average site:

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

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

```r
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)
```

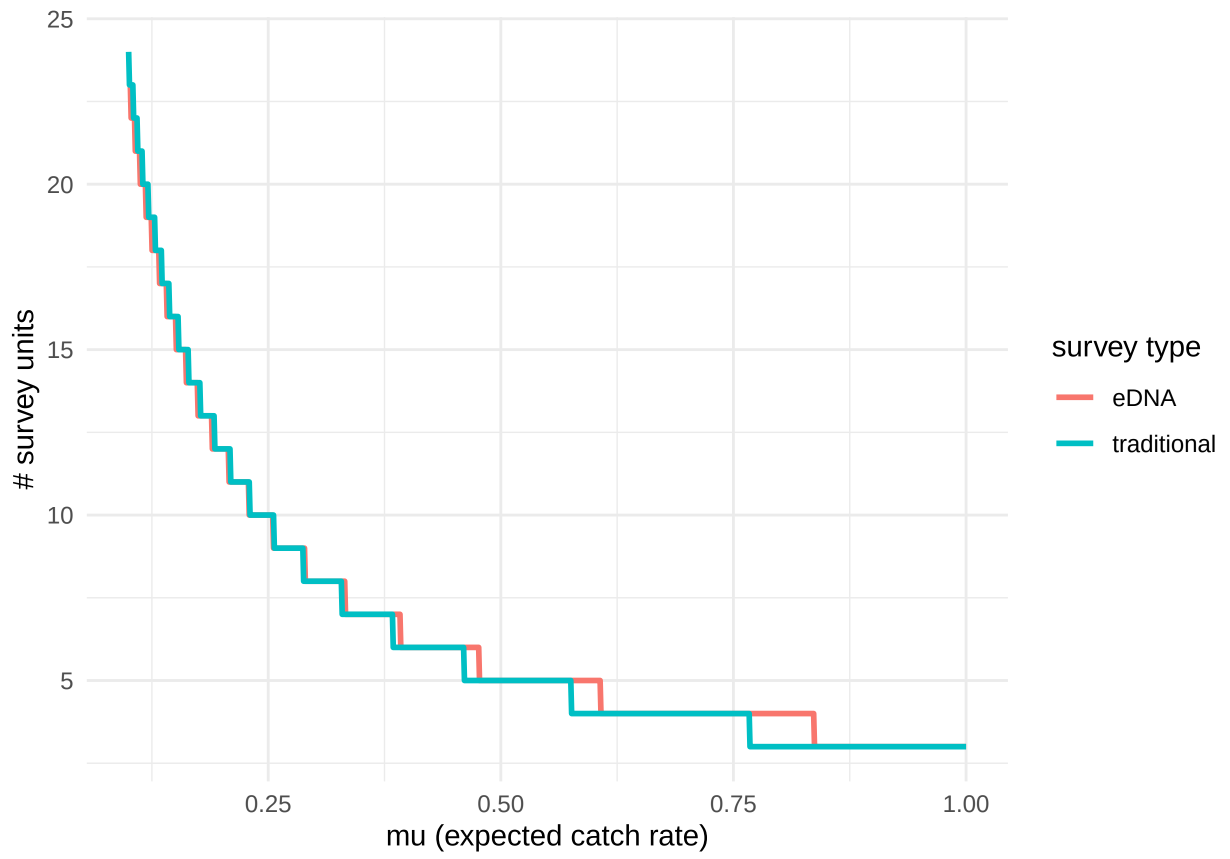


How does this change if we do not calculate for a site with mean covariate values?

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

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

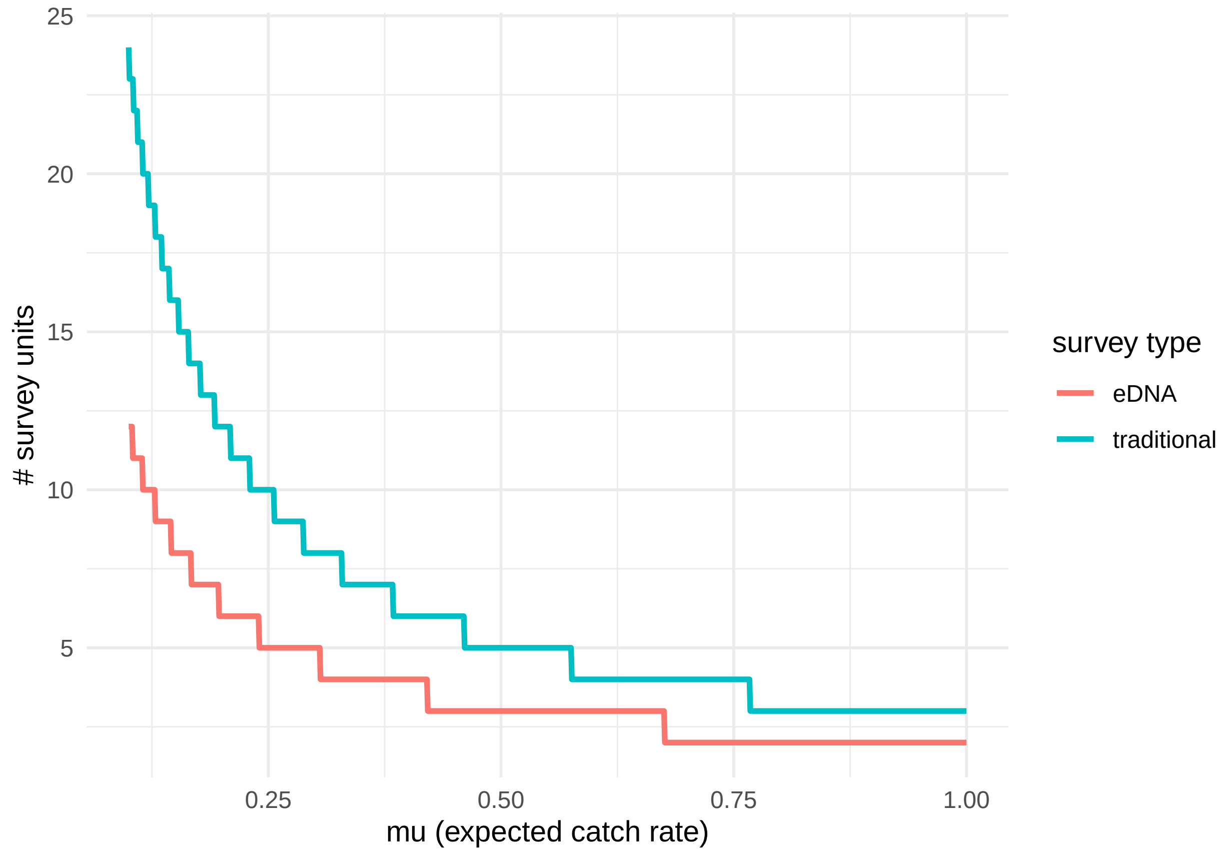
```r
detection_plot(goby_fit_cov1$model,
               mu_min = 0.1, mu_max = 1, # min and max mu
               cov_val = c(0.5, 0), # high filter time
               probability = 0.9)
```



Now let's do the same for salinity:

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

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

```r
# 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
)
```

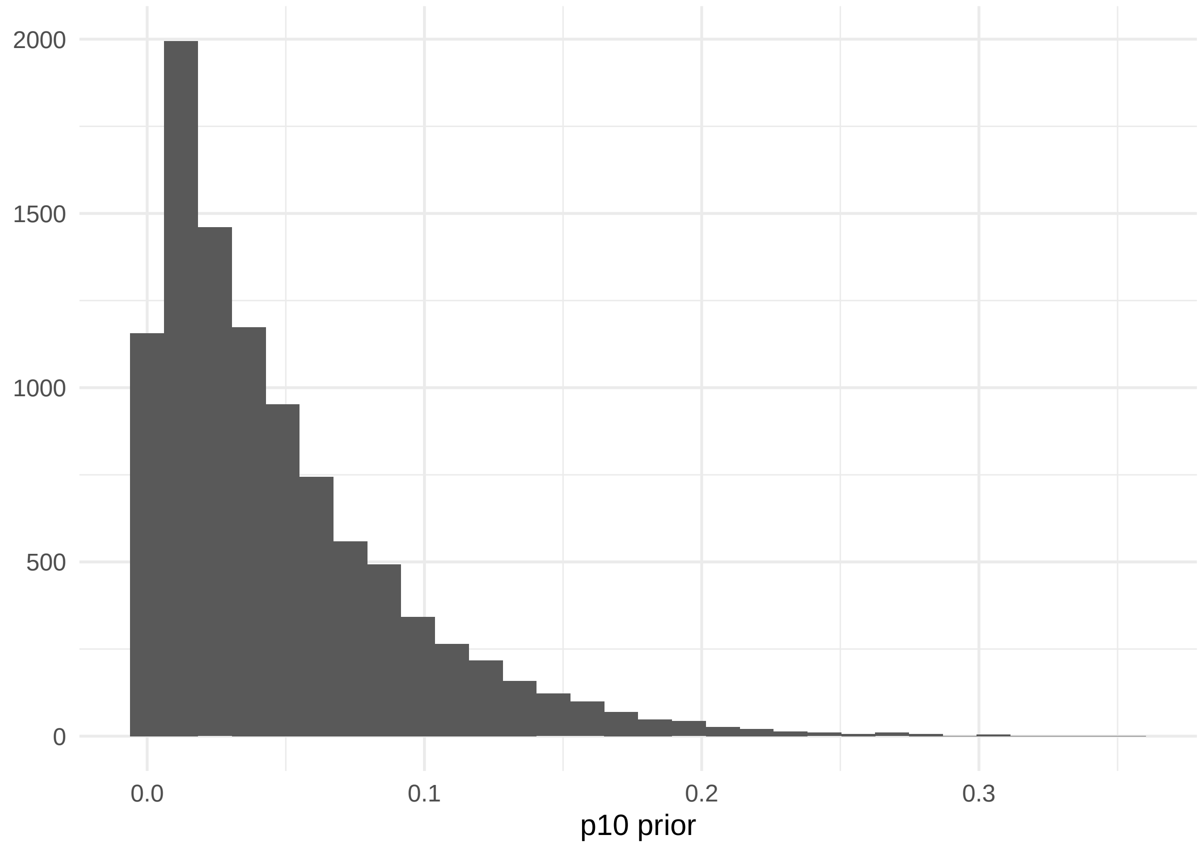
These are the shape parameters of a beta distribution that reflects our prior belief about the probability of a false positive. And this prior belief is updated with our data to give use our posterior estimate.

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)) +
  labs(x = "p10 prior", y = "") +
  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
```

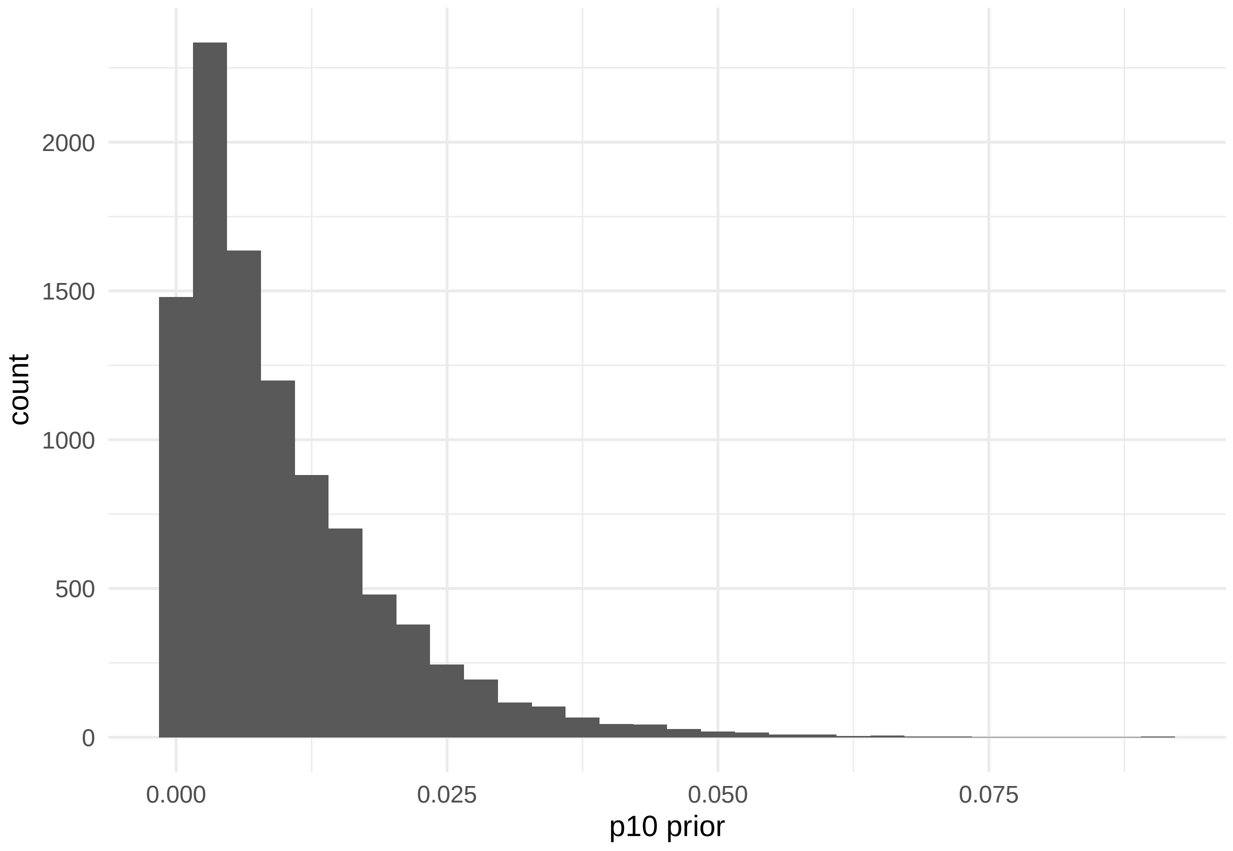
We could keep this prior relatively uninformative, or we could use data to create the prior, perhaps from negative controls.

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, 25), # new prior
                          q = FALSE, verbose = FALSE,
                          multicore = TRUE)
```

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

And then compare the posteriors of `p10`:

Model 1:

```r
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 13727.57    1
```

Model 2:

```r
joint_summarize(fit_prior2$model, par = "p10")
```

```
##      mean se_mean    sd  2.5% 97.5%  n_eff  Rhat
## p10 0.004   0.001 0.004 0.001 0.009 43.135 1.065
```

Model 3:

```r
joint_summarize(fit_prior3$model, par = "p10")
```

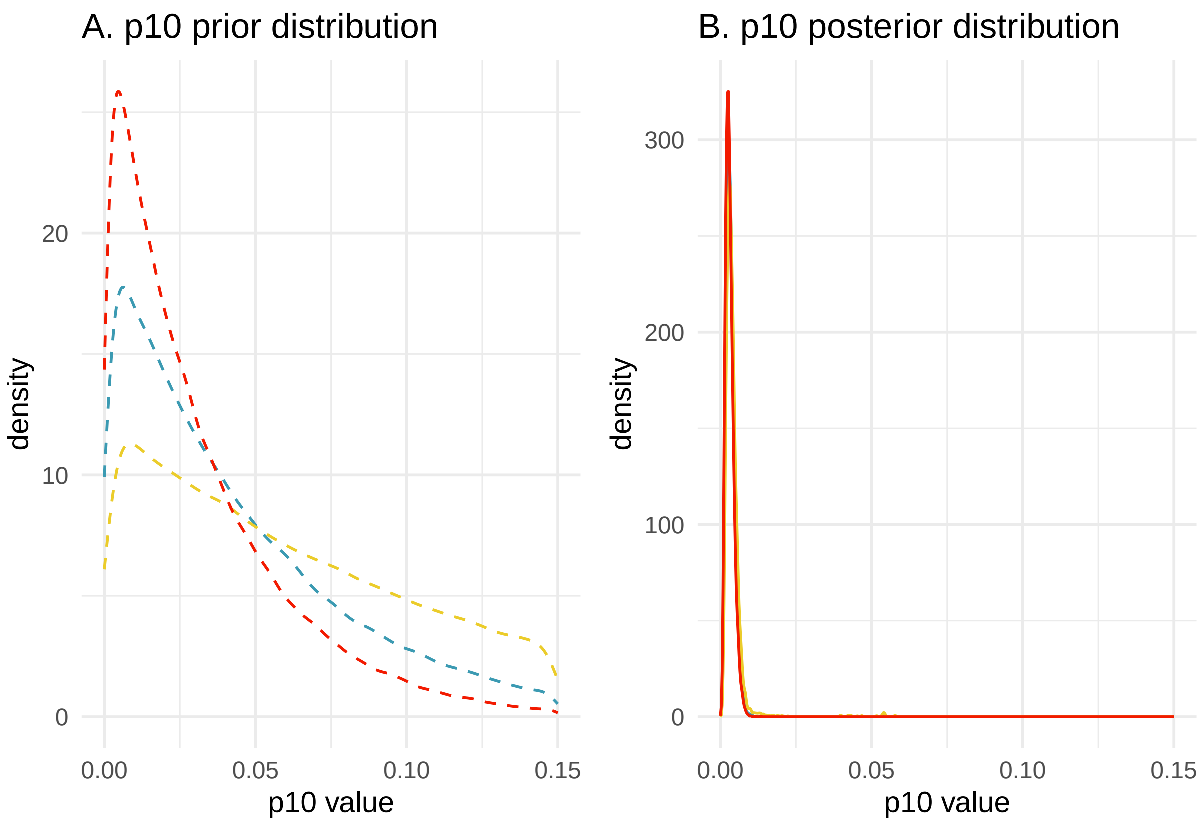
```
##      mean se_mean    sd  2.5% 97.5%    n_eff Rhat
## p10 0.003       0 0.001 0.001 0.006 15580.93    1
```

```r
# 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
density
20
10
0
0.00
0.05
0.10
0.15
p10 value
B. p10 posterior distribution
density
300
200
100
0
0.00
0.05
0.10
0.15
p10 value