BDA - Assignment 2

Anonymous

6 10 2019

Contents

Used libraries:

```
library(dplyr)
library(ggplot2)
library(aaltobda)
data("bioassay")
```

Exercise 1

Note about result accuracy and significant digits: I will use as an estimate the MC-errors to decide what amount of digits to use in the results. For the MC-errors, I will use 'rough' approximates for sample sizes of the orders 1000, 10000 which result in factors: 0.1 and 0.01. Those factors are multiplied by standard errors/dev's to give estimates for sampling deviations.(I will use them roughly)

a)

Construct a bivariate normal distribution as prior distribution for (α, β). The marginal distributions are $\alpha \sim N(0, 2^2), \beta \sim N(10, 10^2)$ with correlation corr$(\alpha, \beta) = 0.5$. Report the mean and covariance of the bivariate normal distribution.

This is pretty straight forward, so I presume it needs no elaborations. In any case, I make a simple function for forming the covariance matrix:

```
make_cov <- function(alpha_var, beta_var, corr) {
  tmp <- corr * sqrt(alpha_var * beta_var)
  matrix(c(alpha_var, tmp, tmp, beta_var), nrow = 2, byrow = T)

}
```

And then use it to calculate the covariance I'll be using in the following tasks:

```
s <- make_cov(4, 100, .5)
s
```

```
##      [,1] [,2]
## [1,]    4   10
## [2,]   10  100
```

And means:

```
mu <- c(0,10)
mu
```

```
## [1]  0 10
```

```
mu+2
```

```
## [1]  2 12
```

b)

Implement a function in R for computing the logarithm of the density of the prior distribution in a) for arbitrary values of α, β

Again, a simple function for priors. Most of the work is done by *dmvnorm* as long as the data is properly formatted within the function:

```
p_log_prior <- function(alpha, beta) {

  #Here the two input vectors are bound columnwise, and put in a matrix
  m <- cbind(alpha, beta) %>% as.matrix(.,nrow=length(alpha[,1]))
  # dmvnorm seemed to operate on matrices in a reasonable way
  dmvnorm(m, mu, s, log = T)

  #Note: Using vectors directly seemed to cause problems, and only
  # first pair was used. This didn't cause any problems with the test
  # cases, but that might have been just a lucky break.
}
```

A quick test with given test samples:

```
alpha <- 3
beta <- 9

p_log_prior(alpha, beta)
```

```
## [1] -6.296435
```

c)

Implement a function in R for computing the logarithm of the density of the posterior for arbitrary values of α, β and data x, y and n.

Again, just a simple function using ready made bioassaylp and earlier p_log_prior. And since densities are logged, instead of product prior times likelihood, I sum their logarithms:

```
p_log_posterior <- function(alpha, beta, x = bioassay$x, y = bioassay$y, n = bioassay$n) {

  p_log_prior(alpha, beta) + bioassaylp(alpha, beta, x,y,n)

}
```

And test again:

```
p_log_posterior(alpha, beta)
```

```
## [1] -15.78798
```

Works...

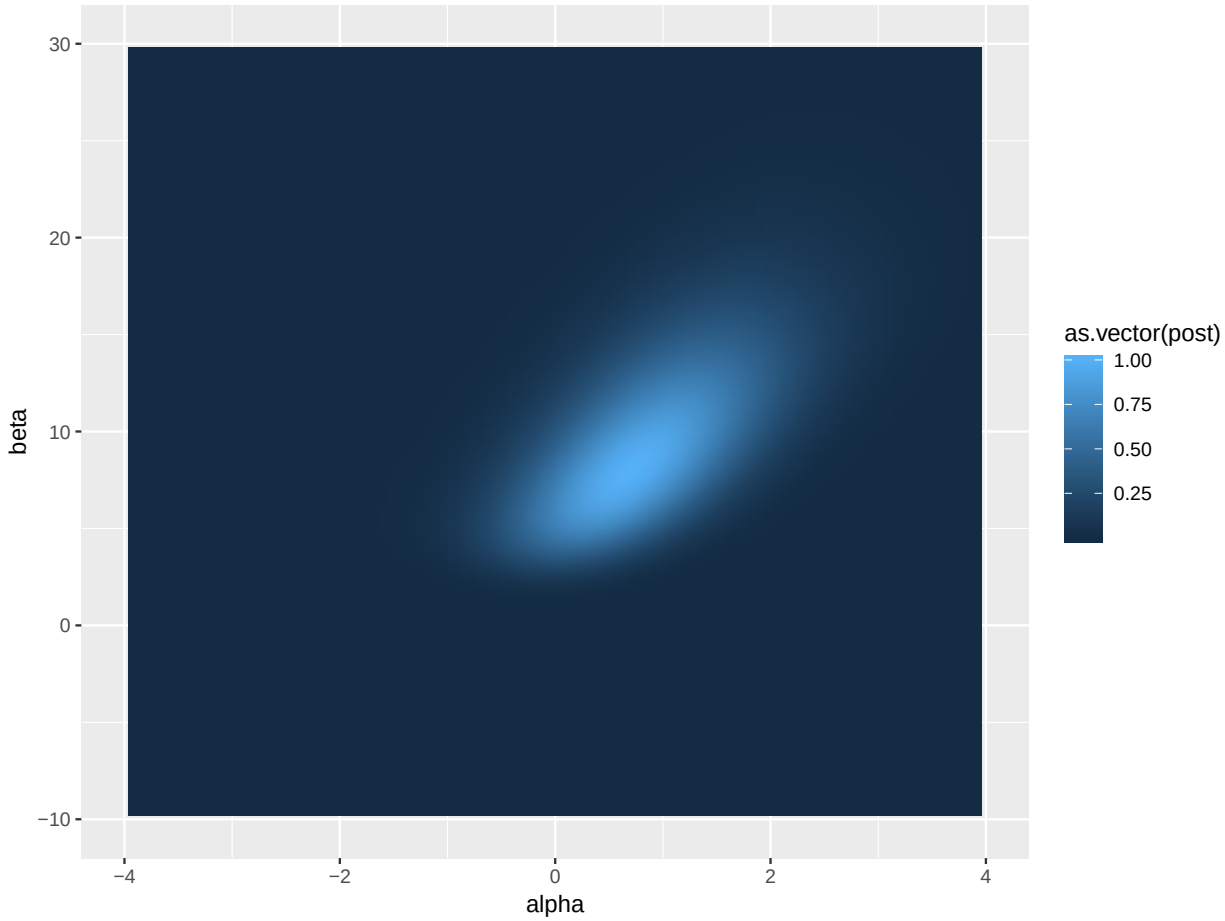
d)

Plot the posterior density in a grid of points ($\alpha \in [-4, 4], \beta \in [-10, 30]$)) using the bioassay_posterior_density_plot function from the aaltobda package. Internally, it uses the

p_log_posterior function you implemented in c).

I guess there is not much to comment here. Maybe just as a side note, I will be using the same parameter domain for later samplings and plots (just to have a point of comparison):

```
bioassay_posterior_density_plot(c(-4,4),c(-10,30), bioassay$x, bioassay$y, bioassay$n)
```



(Already here it seems to show $Pr(\beta < 0) \sim 0$)

e)

Implement two functions in: 1) A function for computing the log importance ratios (log importance weights) for draws from the prior distribution in 1a) when the target distribution is the posterior distribution. 2) A function for exponentiating the log importance ratios and normalizing them to sum to one.

Again, not much to explain. Just note the way logarithms are used:

```
# 1
log_importance_weights <- function(alpha, beta) {

  # log of value ratios is difference of log values
  p_log_posterior(alpha, beta) - p_log_prior(alpha, beta)

}

# 2
```

```
normalized_importance_weights <- function(alpha, beta) {

  # First exponentiate, then normalize
  (log_importance_weights(alpha, beta) %>% exp()) / sum(log_importance_weights(alpha, beta) %>% exp())

}
```

Again, a quick test run:

```
alpha <- c(1.896, -3.6, 0.374, 0.964, -3.123, -1.581)
beta <- c(24.76, 20.04, 6.15, 18.65, 8.16, 17.4)

log_importance_weights(alpha, beta) %>% round(., digits=2)
```

```
## [1]  -8.95 -23.47  -6.02  -8.13 -16.61 -14.57
```

```
normalized_importance_weights(alpha, beta) %>% round(., digits = 3)
```

```
## [1] 0.045 0.000 0.852 0.103 0.000 0.000
```

Seems to work.

f)

Sample draws of α, β from the prior distribution from 1a). Implement a function for computing the posterior mean using importance sampling, and report the obtained mean.

First the function. I am using the already implemented *normalized* weights, and it's a simple calculation of weighted sums:

```
posterior_mean <- function(alpha, beta) {

  # First the normalized weights
  w <- normalized_importance_weights(alpha, beta)

  # Then the weighted sums
  c( sum(w*alpha), sum(w*beta) )

}
```

A quick test run:

```
posterior_mean(alpha, beta) %>% round(.,digits = 3)
```

```
## [1] 0.503 8.275
```

Then the actual run; First a sample of size 5000:

```
sam_from_prior <- rmvnorm(5000, mu, s)
```

Then the means:

```
posterior_mean(sam_from_prior[,1], sam_from_prior[,2])
```

```
## [1]  0.9835984 10.5469959
```

And since the sampling causes additional uncertainty/variance:

```
res_mu <- posterior_mean(sam_from_prior[,1], sam_from_prior[,2]) %>% round(., digits = 1)
res_mu
```

```
## [1]  1.0 10.5
```

So **Result**: Posterior mean is roughly: `1, 10.5`

g)

Using the importance ratios, compute the effective sample size S_{eff} and report it. If S_{eff} is less than 1000, repeat f) with more draws.

The effective sample size can be calculated as the inverse of sum of squared weights. First the function:

```
s_eff <- function(alpha, beta) {
  #using normalized weights
  w <- normalized_importance_weights(alpha, beta)
  1/sum(w^2)

}
```

Just for the lols, let's try with the test case first:

```
s_eff(alpha, beta)
```

```
## [1] 1.354205
```

Seems to be alright. And then let's check the effective size for the real sample:

```
res_eff <- s_eff(sam_from_prior[,1], sam_from_prior[,2])
res_eff
```

```
## [1] 1332.73
```

So, the **resulting** effective sample size was 1332. And luckily (...), it seems to be over 1000. So, no need to repeat the process.

h)

Use importance resampling without replacement to obtain a posterior sample of size 1000 of α, β and plot a scatterplot of the obtained posterior sample.

First we need to pull a proper sample from prior. I will use the earlier sample size (albeit not the same sample). I will also check that my effective samplesize is greater than needed amount of samples (since I am pulling without replacement). Then I will take *normalized* weights for the sample, and use those samples as probabilities for indices of the prior sample (this way I am not losing the joint probability of alpha and beta). And finally the posterior sample can be spliced with the indices from the prior sample. Here's the implementation:

```
#prior sample
sam <- rmvnorm(5000, mu, s)
# S_eff check
print(paste('Effective sample size: ',s_eff(sam[,1],sam[,2]) %>% round(.,digits=0)))
```

```
## [1] "Effective sample size:  1371"
```

```
# weights
sam_w <- normalized_importance_weights(sam[,1],sam[,2])
# indices with corresponding weights are sampled (sample(...) would normalize the weights if they were
ind <- sample(1:length(sam[,1]),1000, replace=F, prob = sam_w)

# Now splicing the prior samples with indices (this is our posterior sample)
ab <- sam[ind,]
```

```r
# A quick check if there are any points in negative beta space
neg_betas <- sum(ab[,2]<0)
print(paste('Negative betas: ',neg_betas))
```

```
## [1] "Negative betas:  0"
```

```r
# Overall probability weight in the negative beta space
print(paste('Overall probability weight in the negative beta space',sam_w[which(sam[,2]<0)] %>% sum()))
```

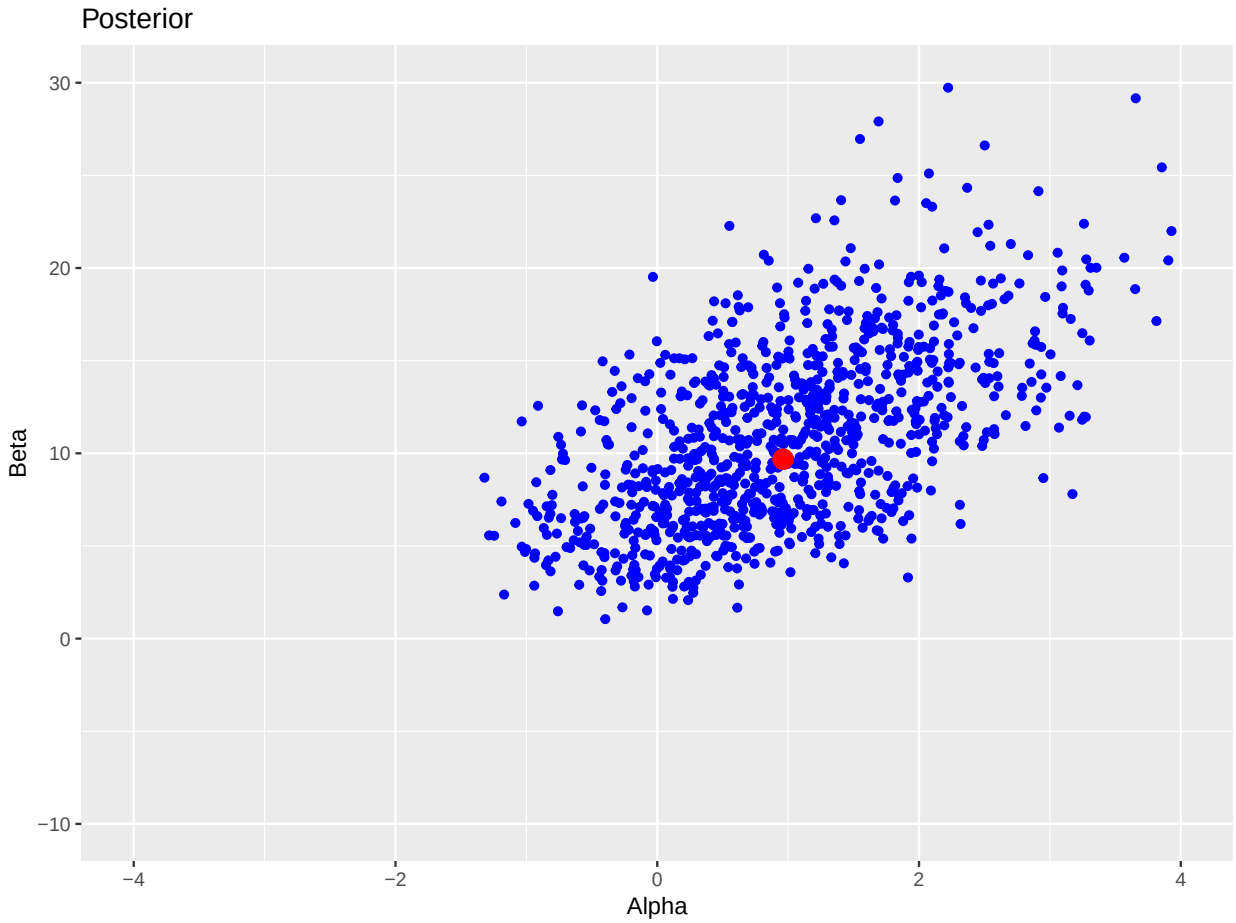
```
## [1] "Overall probability weight in the negative beta space 2.63724873034969e-06"
```

So it seems there is hardly any mass in the $\beta < 0$ area, and in our sample there are exactly 0 negative betas.

Now the posterior plot. Even though it does leave an abundance of empty space, I am plotting with same domain as the earlier aaltobda::bioassay_poster... -function, to see how they compare. So please, take my choise of x,y -limits into acount when grading if the model plot looks different. I am also inserting the posterior mean as a (big) red dot:

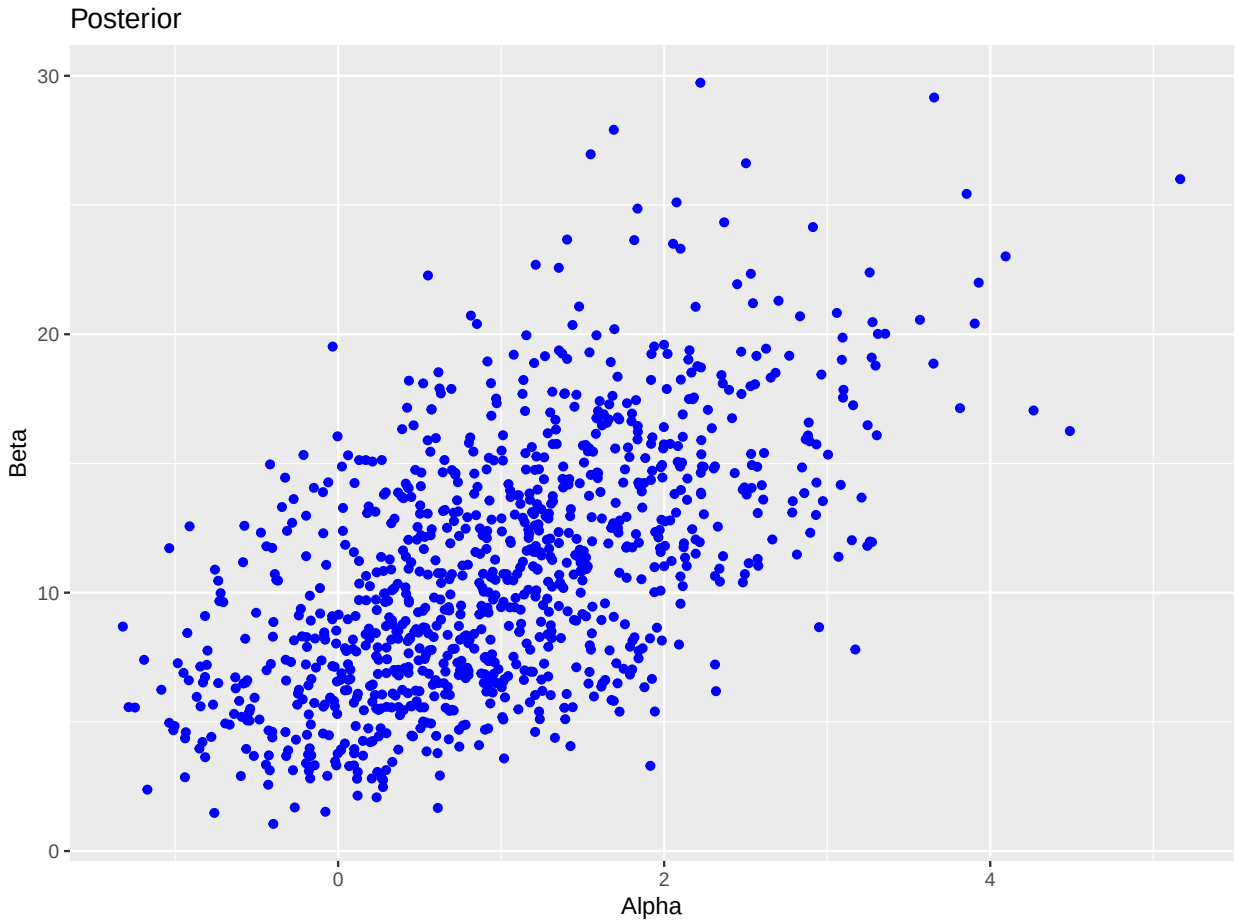
```r
pos_mu <- posterior_mean(ab[,1],ab[,2])

ggplot() +
  geom_point(aes(x=ab[,1], y=ab[,2]), color='blue') +
  xlab('Alpha') +
  ylab('Beta') +
  ggtitle('Posterior') +
  geom_point(aes(x=pos_mu[1], y=pos_mu[2]), color='red', size=4) +
  xlim(c(-4,4)) +
  ylim(c(-10,30))
```



It does look reasonable, and similar to the earlier plot. Just to be safe, here's a simple plot with narrower limits:

```
ggplot() +
  geom_point(aes(x=ab[,1], y=ab[,2]), color='blue') +
  xlab('Alpha') +
  ylab('Beta') +
  ggtitle('Posterior')
```



i)

Using the posterior sample obtained via importance resampling, report an estimate for p($\beta >$ 0 |x, n, y), that is, the probability that the drug is harmful.

As I mostly did this part already in the previous task, I can say, based on my sample, it is highly likely that the drug is harmfull. Just to be precise, the probability that $\beta > 0$ is:

```
mean(ab[,2]>0)
```

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

So in other words, *in my sample* I had `0` negative values for β, and the probability mass in the $\beta < 0$ was extremely low (see previous task). That being said, there is still a non-zero probability for pulling a negative beta from the posterior.

j)

Using the posterior sample obtained via importance resampling, draw a histogram of the draws from the posterior distribution of the LD50 conditional on $\beta > 0$

First one should filter the negative beta values (although I had none in any of my samples - so far). This is mostly a fail-safe for the case if there should be a negative beta in that very time I'm finally compiling my document...

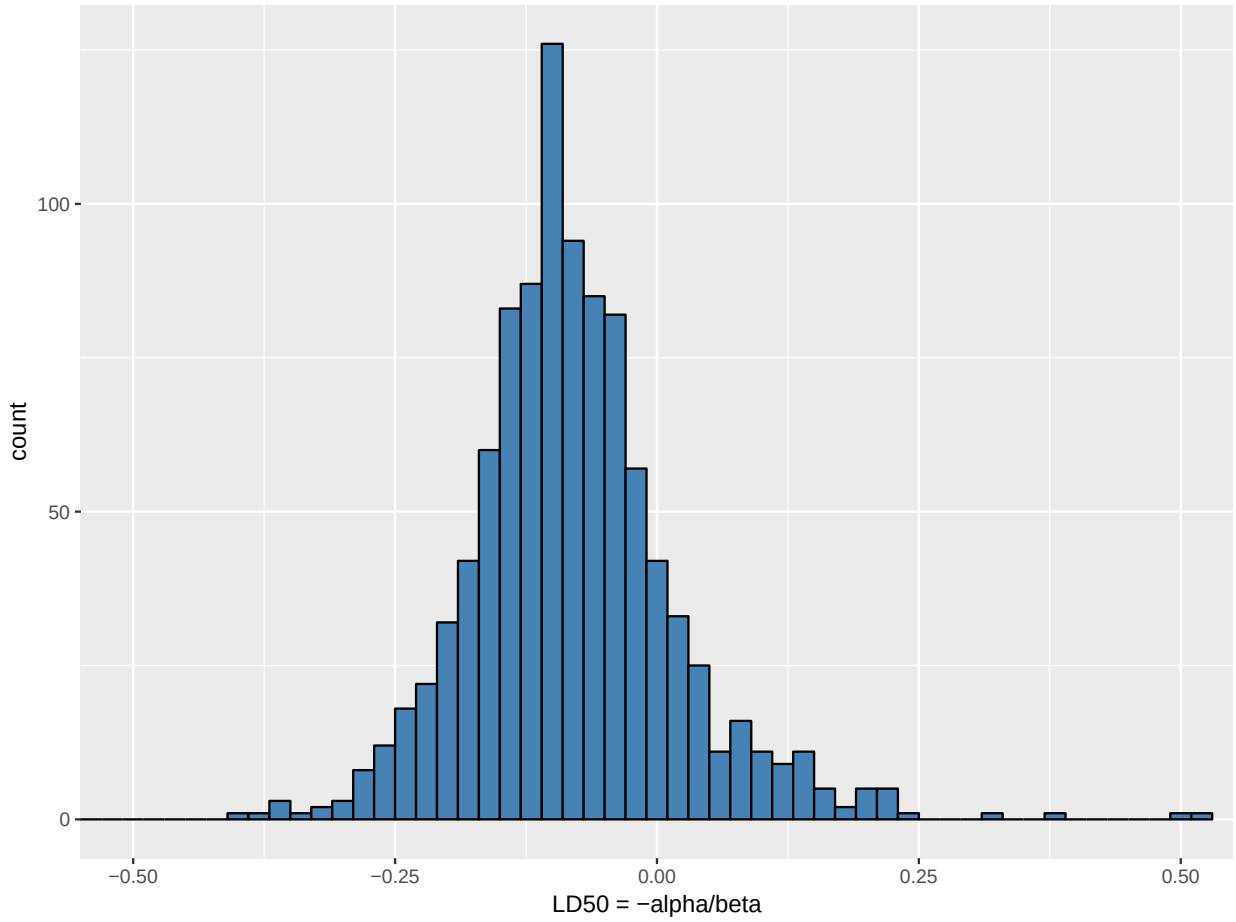
```
filtered_ab <- ab[which(ab[,2]>0),]
```

And then let's compute the LD50's:

```
# negative ratio alpha/beta
ld50 <- -filtered_ab[,1]/filtered_ab[,2]
```

And finally, the histogram:

```
ggplot() +
  geom_histogram(aes(ld50), binwidth = 0.02, fill = 'steelblue', color = 'black') +
  coord_cartesian(xlim = c(-0.5, 0.5)) +
  labs(x = 'LD50 = -alpha/beta')
```



Which does look reasonable. If one should compare it to posterior with uninformative (const) prior, it seems like the mean has slightly moved towards zero.