STAT 656 HW 2

Satoshi Ido (ID: 34788706)

2022-10-03

```
library("ggplot2")
library("tidyverse")
library("MASS")
library("fs")
library("moments")
library("rstan")
library("bayesplot")
library("StanHeaders")
library("knitr")
opts_chunk$set(tidy = TRUE)
```

Synthetic data

The file `hw2 synthetic.csv` is a dataset of count-valued measurements $y = \{y_1, \ldots, y_n\}$, with $y_i \in \{0, 1, \ldots\}$. Each output y_i has an associated $x_i = (x_{i,1}, x_{i,2}) \in \mathbb{R}^2$, and write $x = \{x_1, \ldots, x_n\}'$ as x. We model y_i as

$$y_i|\beta \sim \text{Poisson}(e^{f(x_i,\beta)})$$

Here, the exponential is to ensure the Poisson rate is always positive, and the function $f(x_i, \beta) = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \beta_3 x_{i,1}^2 + \beta_4 x_{i,2}^2 + \beta_5 x_{i,1} x_{i,2}$.

```
df <- read.csv("/Users/satoshiido/Documents/statistical-analysis/656/hw/hw2/data/hw2_synthetic.csv")
head(df)
```

```
      x1          x2 y
```

1 -0.6264538 -0.62036668 0 2 0.1836433 0.04211587 0 3 -0.8356286 -0.91092165 2 4 1.5952808 0.15802877 4 5 0.3295078 -0.65458464 3 6 -0.8204684 1.76728727 0

```
# Create the data for stan simulation of a poisson regression model
df$x1_sq <- df$x1^2
df$x2_sq <- df$x2^2
df$x1_x2 <- df$x1 * df$x2
df$offset <- 1
```

1. With the provided data, perform a Bayesian analysis on the parameters of the model above to decide which terms in the expression for f(x) you think are important. State clearly what your prior over beta is, and how you arrived at your conclusion, including any useful figures (especially of the posterior distribution). You can use Stan.

Summary of the data

```
summary(df)
```

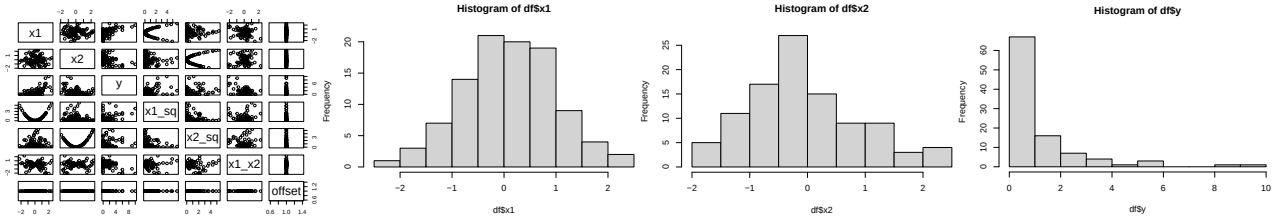
```
##        x1                 x2                  y              x1_sq
##  Min.   :-2.2147   Min.   :-1.91436   Min.   : 0.00   Min.   :0.000001
##  1st Qu.:-0.4942   1st Qu.:-0.65105   1st Qu.: 0.00   1st Qu.:0.110286
##  Median : 0.1139   Median :-0.17722   Median : 1.00   Median :0.352745
##  Mean   : 0.1089   Mean   :-0.03781   Mean   : 1.29   Mean   :0.810551
##  3rd Qu.: 0.6915   3rd Qu.: 0.50090   3rd Qu.: 2.00   3rd Qu.:1.100208
##  Max.   : 2.4016   Max.   : 2.30798   Max.   :10.00   Max.   :5.767768
##      x2_sq              x1_x2              offset
##  Min.   :0.000303   Min.   :-2.131566   Min.   :1
##  1st Qu.:0.093130   1st Qu.:-0.248636   1st Qu.:1
##  Median :0.395067   Median : 0.021236   Median :1
##  Mean   :0.909786   Mean   :-0.004964   Mean   :1
##  3rd Qu.:1.280045   3rd Qu.: 0.392933   3rd Qu.:1
##  Max.   :5.326764   Max.   : 2.235453   Max.   :1
```

```
# sd of x1 and x2
paste0("sd of x1:", sd(df$x1), " sd of x2: ", sd(df$x2))
```

```
## [1] "sd of x1:0.898199359660041 sd of x2: 0.957879069588428"
```

Histogram of data

```
plot(df)
hist(df$x1)
hist(df$x2)
hist(df$y)
```



I have used Stan to perform the Bayesian analysis.
For prior distribution of beta, I have little prior knowledge of the data and model. Also, as we can see by plotting the `x1` and `x2` histogram, they seem to follow normal distribution. Therefore, I set the prior as normal distribution with a large standard deviation, N(0, 100).

We assume $y = \{y_1, \ldots, y_n\}$, with $y_i \in \{0, 1, \ldots\}$. Each output y_i has an associated $x_i = (x_{i,1}, x_{i,2}) \in \mathbb{R}^2$, and write $x = \{x_1, \ldots, x_n\}'$ as x. We model y_i as $y_i|\beta \sim \text{Poisson}(e^{f(x_i,\beta)})$ where the exponential is to ensure the Poisson rate is always positive

The Regression model which I used for the analysis is as below.
Regression model:

$$f(x_i, \beta) = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \beta_3 x_{i,1}^2 + \beta_4 x_{i,2}^2 + \beta_5 x_{i,1} x_{i,2} \tag{1}$$

```
poissonreg_normal_code = "
// Poisson model with normal prior for beta

// Data are things you observe/condition on
data {
  int<lower=1> N; // number of data items
  int<lower=1> p; // Number of beta parameters (predictors)
```

```
  real<lower=0> pr_sd; // std dev of the prior
  matrix[N, p] x; // matrix of predictors
  // real offset[N]; // offset
  int<lower=0> y[N]; // count outcome (output vector)
}

parameters {
  vector[p] beta; // Parameters to estimate
}

// useful to avoid repeating calculations
// note that stan will return values of these variables for each MCMC sample
transformed parameters {
  vector[N] mu = exp(x * beta); // exp of linear predictor
}

// The actual Bayesian model goes here
// I set normal dist as a prior for beta
model {
  beta ~ normal(0, pr_sd); // priors // Note: beta is p-dim
  y ~ poisson(mu); // likelihood
}

// Generate quantities of interest (e.g. posterior predictions)
generated quantities {
  real log_lik[N]; // log likelihood for each observation
  int<lower=0> y_rep[N];
  for (i in 1:N) {
    log_lik[i] = poisson_lpmf(y[i] | mu[i]); // likelihood
    // generate a new predicted data point y_rep[i] that is consistent with the current sample of the pa
    y_rep[i] = poisson_rng(mu[i]);
  }
}
"

# Build the model before sampling
poissonreg_normal <- stan_model(model_code = poissonreg_normal_code)
```

```
# Create the data for stan simulation
X <- df[, c("x1", "x2", "x1_sq", "x2_sq", "x1_x2", "offset")]
y <- df$y
```

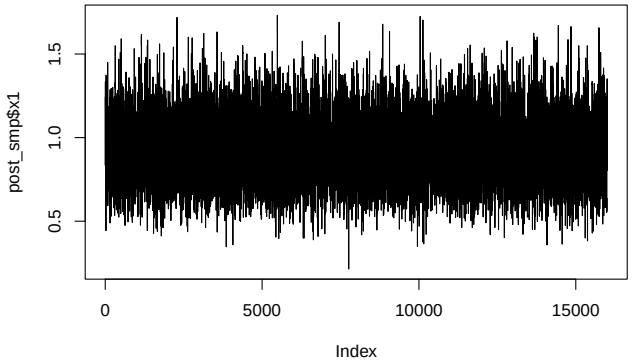
```
# Fit the model to the data and obtain posterior samples
poissonreg_data <- list(N = nrow(X), p = ncol(X), pr_sd = 100, x = X, y = y)
nfit <- rstan::sampling(
  object = poissonreg_normal,
  data = poissonreg_data,
  iter = 10000,
  warmup = 2000,
  chains = 2
)
```

2. Having decided which terms in f are important, keep only those and discard the rest, resulting in a possibly simpler model. Now perform a Bayesian analysis over the parameters of this model.

The posterior distribution of the parameters are as below.
After considering the outputs of the model and fact that the parameter `x1_sq` and `x1_x2` are not significant in the model, I decided to remove these parameters from the model and run the model again. As the result, x_1 and x_2 still seem to be significant in the model, while x_2^2 seems not significant. There is a possibility that the x_2 and x_2^2 are correlated, and have a multicollinearity problem. Yet, it does not seem to be serious problem in this case, since the x_2 and x_2^2 are not highly correlated. Hence, I decided to keep the x_2 in the model.

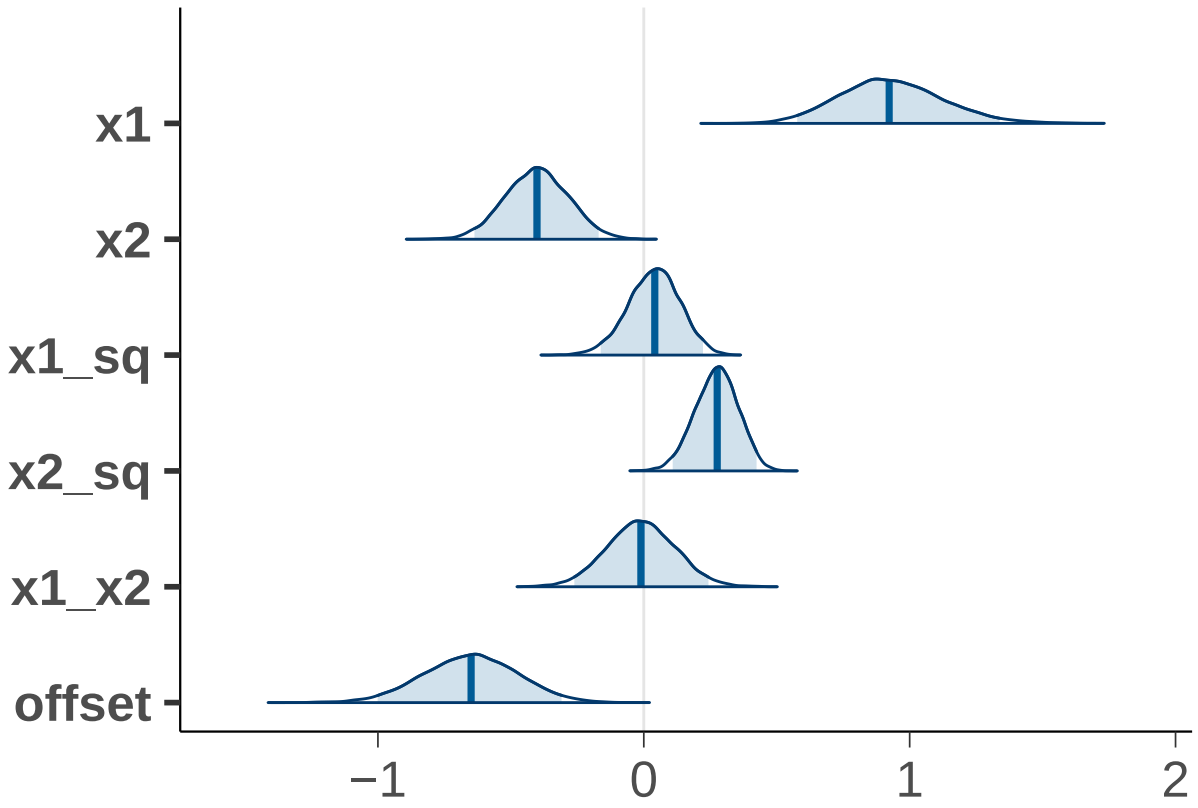
```
# Extract the posterior samples for further analysis
post_smp <- as.data.frame(nfit)[, c(1, 2, 3, 4, 5, 6)]
colnames(post_smp) <- colnames(X)
```

```
plot(post_smp$x1, type = "l")
```



I have checked the histogram of the posterior distribution of the parameters, and it seems that `x1_sq` and `x1_x2` are not significant in the model. Therefore, we try to compare the model with and without `x1_sq` and `x1_x2` to see which model is better.

```
mcmc_areas(post_smp, pars = colnames(X), prob = 0.95)
```



model without x1_sq and x1_x2

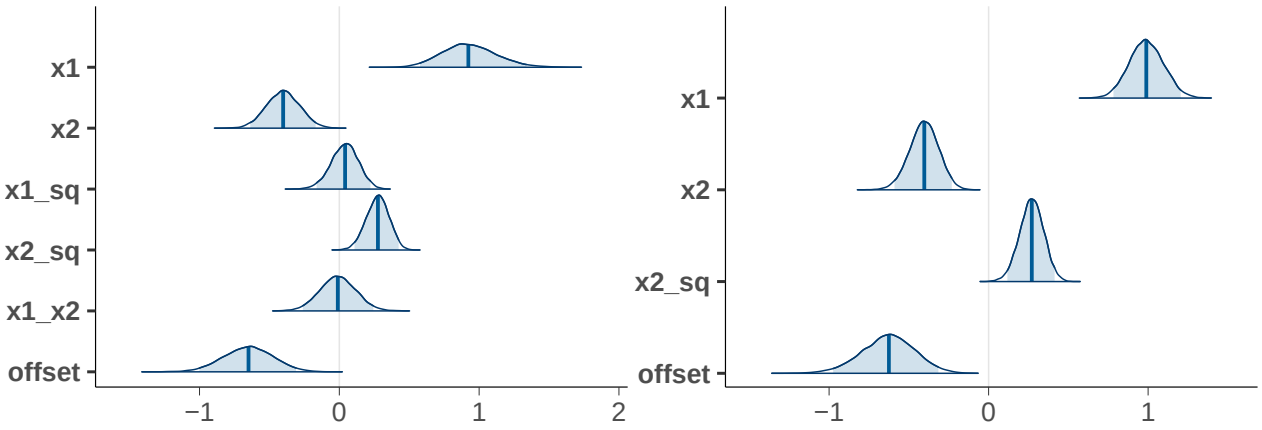
```
# Create the data for stan simulation
X2 <- df[, c("x1", "x2", "x2_sq", "offset")]
y2 <- df$y
```

```
# Fit the model to the data and obtain posterior samples
poissonreg_data2 <- list(N = nrow(X2), p = ncol(X2), pr_sd = 100, x = X2, y = y2)
nfit2 <- rstan::sampling(
  object = poissonreg_normal,
  data = poissonreg_data2,
  iter = 10000,
  warmup = 2000,
  chains = 2
)
```

```
# Extract the posterior samples for further analysis
post_smp2 <- as.data.frame(nfit2)[, c(1, 2, 3, 4)]
colnames(post_smp2) <- colnames(X2)
```

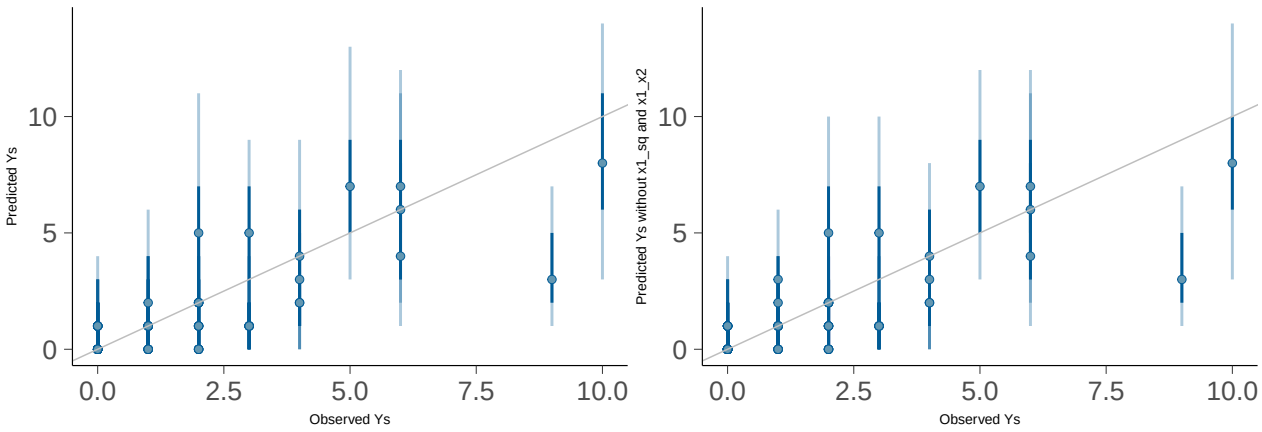
Apparently, without x1_sq and x1_x2, the posterior distribution of the parameters look more significant.

```
# Plot comparison of the two models
mcmc_areas(post_smp, pars = colnames(X), prob = 0.95)
mcmc_areas(post_smp2, pars = colnames(X2), prob = 0.95)
```



We can check interval estimates for each predicted observations in both models.

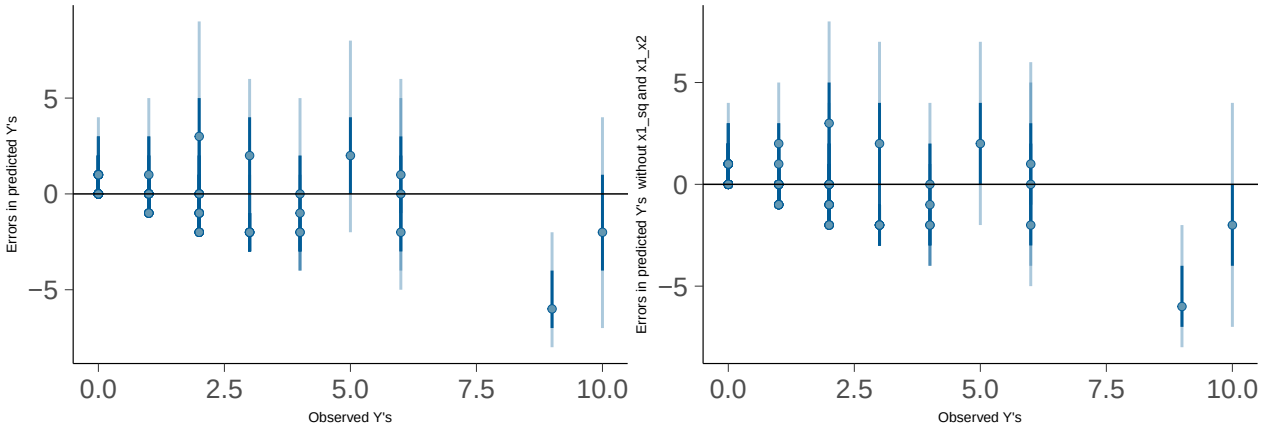
```
# Plot the posterior distribution of the parameters of the base model
y_rep <- extract(nfit, "y_rep")$y_rep

## Interval estimates for each predicted observations (columns) in y_pred
ppd_intervals(y_rep, x = y) +
  geom_abline(intercept = 0, slope = 1, color = "grey") +
  ggplot2::labs(y = "Predicted Ys", x = "Observed Ys") +
  theme(axis.title.x = element_text(size = 10),
        axis.title.y = element_text(size = 10))

# Plot the posterior distribution of the parameters of the model without x1_sq and x1_x2
y_rep2 <- extract(nfit2, "y_rep")$y_rep

## Interval estimates for each predicted observations (columns) in y_pred2
ppd_intervals(y_rep2, x = y2) +
  geom_abline(intercept = 0, slope = 1, color = "grey") +
  ggplot2::labs(y = "Predicted Ys without x1_sq and x1_x2", x = "Observed Ys") +
  theme(axis.title.x = element_text(size = 10),
        axis.title.y = element_text(size = 10))
```



We can check an error in the predicted Y's for both models as well.

```
ppd_intervals(t(t(y_rep) - y), x = y) +
  geom_abline(intercept = 0, slope = 0) +
  ggplot2::labs(y = "Errors in predicted Y's", x = "Observed Y's") +
  theme(axis.title.x = element_text(size = 10),
```

```
        axis.title.y = element_text(size = 10))

ppd_intervals(t(t(y_rep2) - y2), x = y2) +
  geom_abline(intercept = 0, slope = 0) +
  ggplot2::labs(y = "Errors in predicted Y's  without x1_sq and x1_x2", x = "Observed Y's") +
  theme(axis.title.x = element_text(size = 10),
        axis.title.y = element_text(size = 10))
```



The second model looks better because the variance of predictions is smaller.

3. Perform posterior predictive checks for both models, being sure to explain what you are doing. Which model do you think fits the data better?

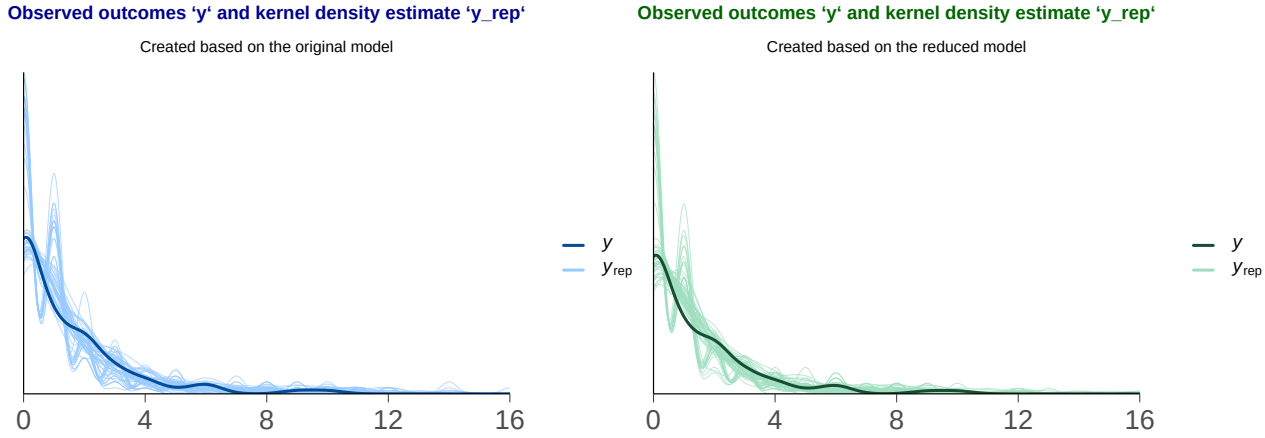
We have already been through a posterior predictive checking. The idea behind posterior predictive checking is simple: if a model is a good fit then we should be able to use it to generate data that looks a lot like the data we observed.
To generate the data used for posterior predictive checks, we simulate from the posterior predictive distribution. This is the distribution of the outcome variable implied by a model after using the observed data `y` (a vector of N outcome values) to update our beliefs about unknown model parameters θ.
The posterior predictive distribution for observation $y^{\sim}$ can be written as

$$p(y^{\sim}|y) = \int p(y^{\sim}|\theta)p(\theta|y)d\theta.$$

We have checked some of the posterior predictive distribution of the parameters of the models above.
In addition to the plots above, we can also check and compare the posterior predictive distributions of the conversion for each model by plotting some histograms and high density estimates.

```
# Plot the posterior predictive distribution of the parameters of the base model
posterior_samples <- rstan::extract(nfit, permuted = TRUE)
y_rep_samples <- posterior_samples$y_rep
# Each row corresponds to a posterior sample (= 16000).
# Each column corresponds to a data point (i.e., each of the N data points I have) (= 100).
# dim(y_rep_samples)

color_scheme_set("brightblue")
ppc_dens_overlay(y, y_rep_samples[1:50, ]) +
  labs(title = "Observed outcomes `y` and kernel density estimate `y_rep`", subtitle = "Created based o
  theme(
    plot.title = element_text(
```

```
      size = 13,
      face = "bold",
      color = "darkblue",
      hjust = 0.5),
    plot.subtitle = element_text(size = 11, hjust = 0.5)
    )

# Plot the posterior predictive distribution of the parameters of the base model
posterior_samples2 <- rstan::extract(nfit2, permuted = TRUE)
y_rep_samples2 <- posterior_samples2$y_rep

color_scheme_set("green")
ppc_dens_overlay(y2, y_rep_samples2[1:50, ]) +
  labs(title = "Observed outcomes `y` and kernel density estimate `y_rep`", subtitle = "Created based o
  theme(
    plot.title = element_text(
      size = 13,
      face = "bold",
      color = "darkgreen",
      hjust = 0.5),
    plot.subtitle = element_text(size = 11, hjust = 0.5)
  )
```



Plots above gave us a hint but they are not clear. Hence, we can also compare the models using Leave-One-Out Cross-Validation method (LOO-CV).
Here, the key output metrics are:

- elpd_diff: The difference in expected log pointwise predictive density. Positive values indicate that the model on the row is expected to make better predictions (on average) than the model/column used for comparison.

- se_diff: The standard error of the difference.

Reduced model has a better elpd_diff (= + 3.0), meaning the model is expected to predict new data better than the original model based on LOO-CV as the metric of comparison.

```
# Compute LOO-CV for the first model
loo_all <- loo(nfit)

# Exclude x1_sq and x1_x2 from the model
X2 <- X[, !colnames(X) %in% c("x1_sq", "x1_x2")]
```

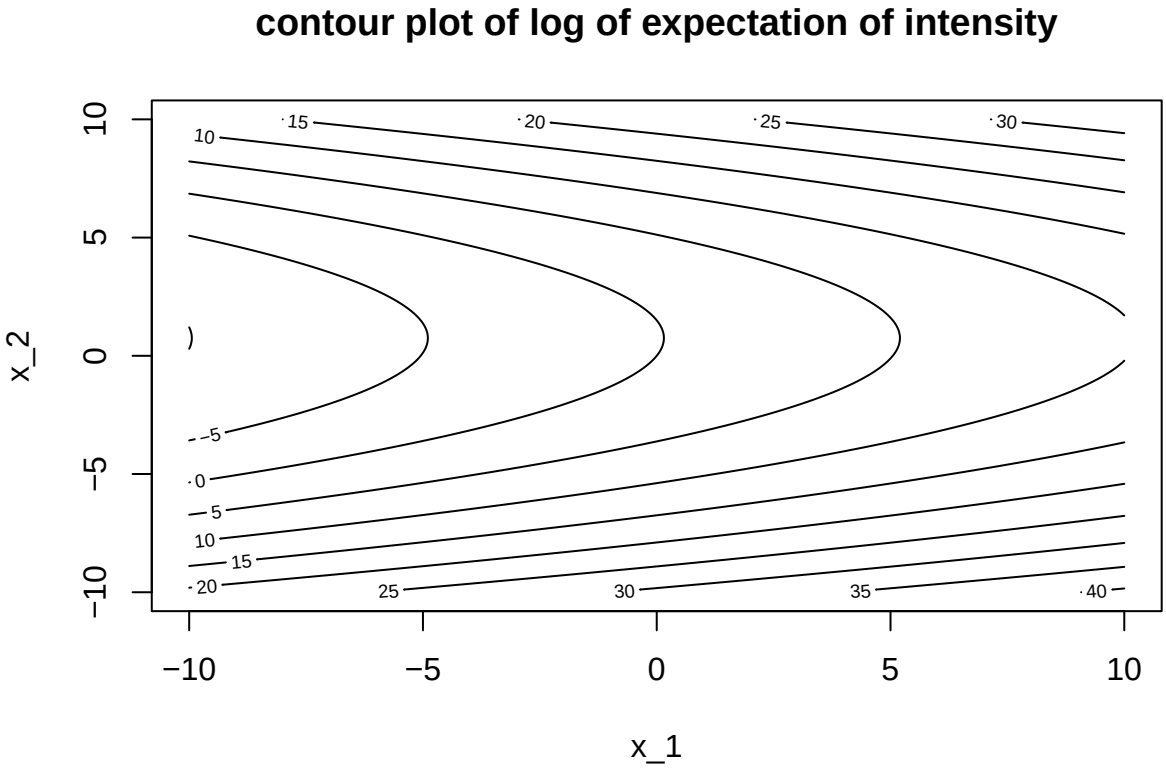
```r
poissonreg2_data <- list(N = nrow(X2), p = ncol(X2), x = X2, y = y2, pr_sd = 100)

poissonreg2 <- rstan::sampling(
  object = poissonreg_normal,
  data = poissonreg2_data,
  iter = 10000,
  warmup = 2000,
  chains = 2
)

# Compute LOO-CV for the second model
loo2 <- loo(poissonreg2)

# Compare the models
loo::loo_compare(loo_all, loo2)
```

```
##        elpd_diff se_diff
## model2  0.0       0.0
## model1 -2.9       1.3
```

4. Use the results from the second model to create a contour plot showing the log average Poisson intensity as a function of x. In other words, plot $logE_{p(\beta|x,y)}[exp(f(x,\beta))]$ as a function of the two components of x (you can restrict the component ranges from -10 to +10).

I create the function and display a contour plot of the log average Poisson intensity as a function of x.

```r
# Define a grid of concentration settings
x1_seq <- seq(-10, 10, length.out = 1000)
x2_seq <- seq(-10, 10, length.out = 1000)

# Function to compute the log of the expectation
log_exp_func <- function(x1, x2, beta){
  mu_predictive <- mean(beta[,1]) * x1 + mean(beta[, 2]) * x2 + mean(beta[, 3]) * x2^2
  expected_val <- mean(exp(mu_predictive))
  return(log(expected_val))
}

# we take a function that works on scalars and vectorize it so that it can work on vectors in an elemen
intensity <- outer(x1_seq, x2_seq, Vectorize(function(x1, x2) log_exp_func(x1, x2, post_smp2)))

# Now, plot the results with contour
contour(
  x1_seq, x2_seq, intensity, nlevels=10, xlab = "x_1", ylab = "x_2",
  main = "contour plot of log of expectation of intensity"
)
```



Applied problem

First design selection

My first approach is to run the eperiments with 24 wells initially, followed by 48 wells in the second experiments. The reason for this is that I want to see if the 24 wells are enough to get the information about the mean and variance of the population so that I can set a reasonable prior for the second experiment. I will assign pairs of concentrations for the two chemical modulators to each well in the manner as below. The main focus is to see how the effect of the one modulator changes depending on the concentration of the other modulator. Since I have little idea of the effect of the modulators, I will assign the concentrations of the modulators somewhat randomly.

```
design1 <- matrix(nrow = 24, ncol = 2)
# assign the concentration of the modulators randomly
## set a seed to reproduce the same result
set.seed(49)
modA <- rep(seq(0, 75, by = 15), each = 4)
modB <- rep(seq(0, 30, by = 10), times = 6)
design1 <- cbind(modA, modB)
# write a table and save it to csv
write.table(design1, file = "/Users/satoshiido/Documents/statistical-analysis/656/hw/hw2/outputs/design
```

Second design selection

Based on the result of the first experiment, I will take a Bayesian approach to run the second experiment with 48 wells. Mainly, I will take the following five steps to create the second experiment design. * Build Bayesian Model: Perform Bayesian inference on the first 24 runs to estimate the model parameters. * Create Contour Plot: Use the estimated parameters to create a contour plot of the posterior predictive mean of conversion as a function of concentration settings for modulator A and modulator B. * Optimization: Find the concentration

settings that maximize the posterior predictive mean of conversion using an optimization algorithm. * Posterior Predictive Distribution: Calculate the posterior predictive distribution of the concentration settings that yield maximum conversion. * Posterior Predictive Distribution of Conversion: Construct the posterior predictive distribution of the conversion corresponding to the maximum conversion concentration settings.

```
# resulf of the first experiment
design1_result <- read.table("/Users/satoshiido/Documents/statistical-analysis/656/hw/hw2/data/rslt1.cs
colnames(design1_result) <- c("A", "B", "y")
design1_result
```

```
##     A  B           y
## 1   0  0 -0.16111064
## 2   0 10 -0.07848086
## 3   0 20 -0.53289365
## 4   0 30 -1.85996326
## 5  15  0  0.03215158
## 6  15 10  1.72423283
## 7  15 20  2.46785113
## 8  15 30  1.29173204
## 9  30  0 -1.63989596
## 10 30 10  1.36423535
## 11 30 20  1.41115862
## 12 30 30  0.79123690
## 13 45  0 -3.03085521
## 14 45 10  1.03320638
## 15 45 20  2.18227189
## 16 45 30 -0.42219770
## 17 60  0 -1.98515998
## 18 60 10  2.38282290
## 19 60 20  3.57714517
## 20 60 30 -0.55883519
## 21 75  0 -1.99071184
## 22 75 10  2.42820357
## 23 75 20  3.29022743
## 24 75 30 -3.12989196
```

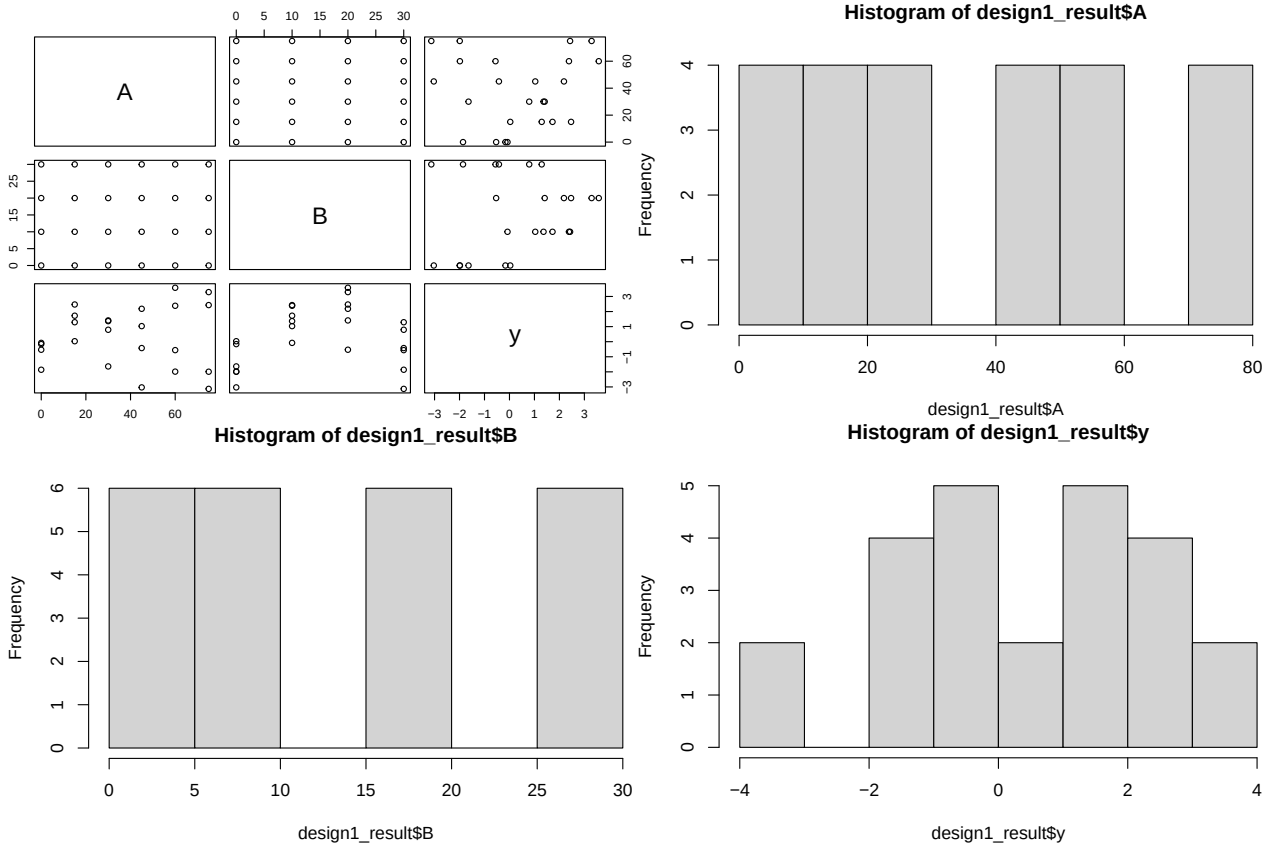
```
summary(design1_result)
```

```
##        A               B              y
##  Min.   : 0.0   Min.   : 0.0   Min.   :-3.1299
##  1st Qu.:15.0   1st Qu.: 7.5   1st Qu.:-0.8291
##  Median :37.5   Median :15.0   Median : 0.4117
##  Mean   :37.5   Mean   :15.0   Mean   : 0.3578
##  3rd Qu.:60.0   3rd Qu.:22.5   3rd Qu.: 1.8387
##  Max.   :75.0   Max.   :30.0   Max.   : 3.5771
```

```
# sd of A and B
paste0("sd of A:", sd(design1_result$A), " | ", " sd of B: ", sd(design1_result$B), " | ", " sd of y: "
```

```
## [1] "sd of A:26.1683512796594 |  sd of B: 11.4208048144032 |  sd of y: 1.94496363035641"
```

```
plot(design1_result)
hist(design1_result$A)
hist(design1_result$B)
hist(design1_result$y)
```




```
# mean of each level in modulator A
design1_result %>%
  group_by(A) %>%
    summarise_at(vars(y), list(name = mean))
```

```
## # A tibble: 6 x 2
##       A    name
##   <int>   <dbl>
## 1     0 -0.658
## 2    15  1.38
## 3    30  0.482
## 4    45 -0.0594
## 5    60  0.854
## 6    75  0.149
```

```
# mean of each level in modulator B
design1_result %>%
  group_by(B) %>%
    summarise_at(vars(y), list(name = mean))
```

```
## # A tibble: 4 x 2
##       B   name
##   <int>  <dbl>
## 1     0 -1.46
## 2    10  1.48
## 3    20  2.07
## 4    30 -0.648
```

As we can see the result of the first design and its scatter plot, the modulators are correlated with each other and have some effect on the conversion.
For instance, when the value of Modulator B are either 0 or 30, the conversion became lower, meanwhile it became higher when Modulator B are either 10, 20.
Also, as we can see from the average of each level in modulator A, there seems to be a tipping point around 60.
Therefore I set `Inverse-Gamma` distribution $\sim (\alpha = 1, \beta = 1)$ as a prior for the parameters of the modulators, and use the following model to estimate the parameters of the modulators., and use the following model to estimate the parameters of the modulators.

Upon runnning the model, I assume $y = \{y_1, \ldots, y_n\}$, with $y_i \in \mathbb{R}$. Each output y_i has an associated $x_i = (x_{i,1}, x_{i,2}) \in \mathbb{R}^2$, and write $x = \{x_1, \ldots, x_n\}'$ as x.

We model y_i as

$$y_i|\beta \sim \text{Normal}(f(x_i, \beta), \sigma)$$

Stan code for the model simulation

```
reg_half_cauchy_code = "
// regression model with normal prior for beta

// Data are things you observe/condition on
data {

  int<lower=1> N; // number of data items
  int<lower=1> p; // Number of beta parameters (predictors)
  matrix[N, p] x; // Matrix of predictors
  real<lower=0> pr_sd; // Prior standard deviation for beta
  real y[N]; // output vector
}

parameters {

  vector[p] beta; // Parameters to estimate
  real<lower=0> sigma; // Standard deviation
}

// useful to avoid repeating calculations
// note that stan will return values of these variables for each MCMC sample
transformed parameters {

  vector[N] mu = x * beta; // Linear predictor
}

// The actual Bayesian model goes here
// I set normal dist as a prior for beta
model {

  // priors
  beta ~ normal(0, pr_sd); // Note: beta is p-dim
  sigma ~ cauchy(0, 5); // Half-Cauchy prior for sigma with scale parameter 5 (adjust as needed)
  y ~ normal(mu, sigma); // likelihood
}

// Generate quantities of interest
generated quantities {
```

```
  real log_lik[N]; // log likelihood for each observation
  real y_rep[N];
  for (i in 1:N) {
    y_rep[i] = normal_rng(mu[i], sigma);
    log_lik[i] = normal_lpdf(y[i] | mu[i], sigma); // compute log likelihood for each observation
  }
}
"

# build the model before sampling
reg_half_cauchy <- stan_model(model_code = reg_half_cauchy_code)
```

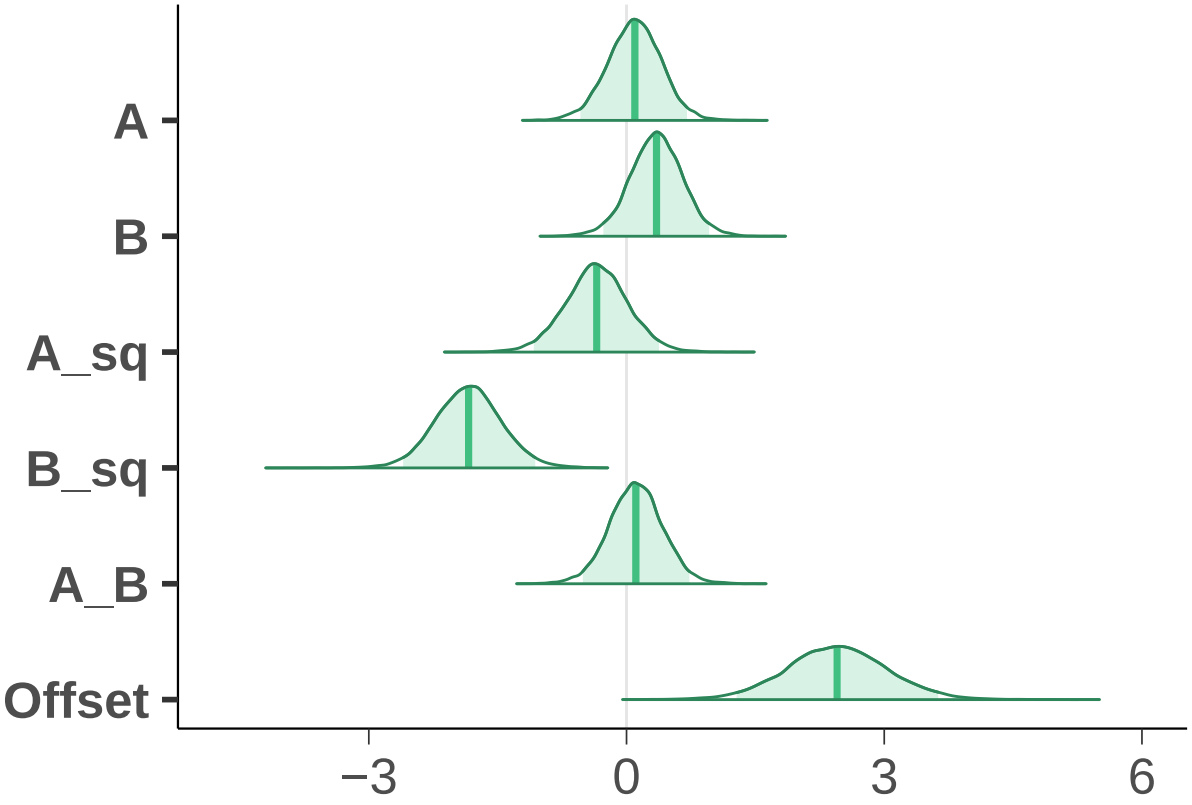
```
# standardize the data
design1_result$A <- (design1_result$A - mean(design1_result$A)) / sd(design1_result$A)
design1_result$B <- (design1_result$B - mean(design1_result$B)) / sd(design1_result$B)

# create the data for a regression model
design1_result$A_sq <- design1_result$A^2
design1_result$B_sq <- design1_result$B^2
design1_result$A_B <- design1_result$A * design1_result$B

# create the data for stan simulation
X <- design1_result[, c("A", "B", "A_sq", "B_sq", "A_B")]
X[, "Offset"] <- 1
y <- design1_result$y
```

```
# compile the model
reg1_data <- list(N = nrow(X), p = ncol(X), x = X, y = y, pr_sd = 100)
reg_fit <- rstan::sampling(
  object = reg_half_cauchy,
  data = reg1_data,
  iter = 10000,
  warmup = 2000,
  chains = 2,
  seed = 9
)
```

```
# save the model
saveRDS(reg_fit, file = "/Users/satoshiido/Documents/statistical-analysis/656/hw/hw2/outputs/reg_fit.rds
# read the model
reg_fit <- readRDS(file = "/Users/satoshiido/Documents/statistical-analysis/656/hw/hw2/outputs/reg_fit.
```

Check the result of the model

```
post_smp_reg <- as.data.frame(reg_fit)
# rename the columns
colnames(post_smp_reg)[1:6] <- colnames(X)
mcmc_areas(post_smp_reg[, 1:6], pars = colnames(X)[1:6], prob = 0.95)
```



Bayesian model selection
Reduced model has a better elpd_diff (= + 2.8), meaning the model is expected to predict new data better than the original model based on LOO-CV as the metric of comparison.

```
# Compute LOO-CV for the first model
loo_all <- loo(reg_fit)

# Exclude A_sq and A_B from the model
X_non <- X[, !colnames(X) %in% c("A_sq", "A_B")]
reg1_data_non <- list(N = nrow(X_non), p = ncol(X_non), x = X_non, y = y, pr_sd = 100)

reg_fit_non <- rstan::sampling(
  object = reg_half_cauchy,
  data = reg1_data_non,
  iter = 10000,
  warmup = 2000,
  chains = 2,
  seed = 9
)

# Compute LOO-CV for the second model
loo2 <- loo(reg_fit_non)

# Compare the models
loo::loo_compare(loo_all, loo2)
```

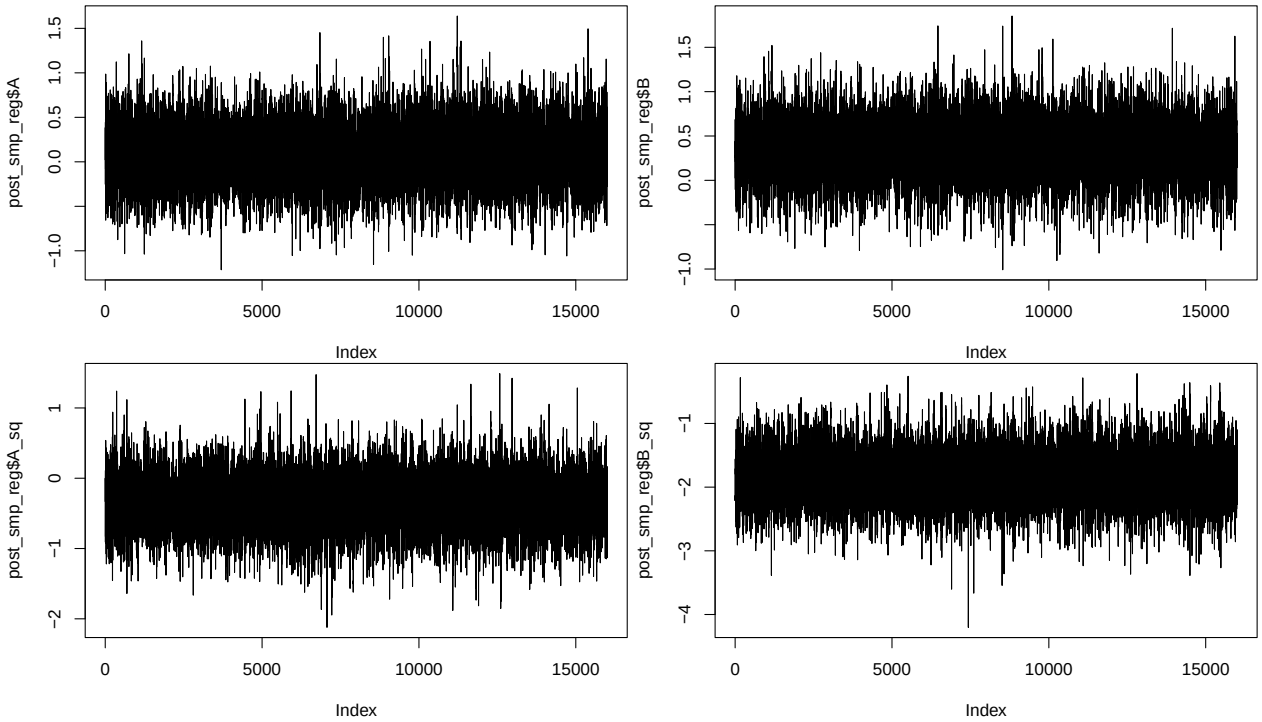
```
##        elpd_diff se_diff
## model2  0.0       0.0
## model1 -2.5       1.3
```

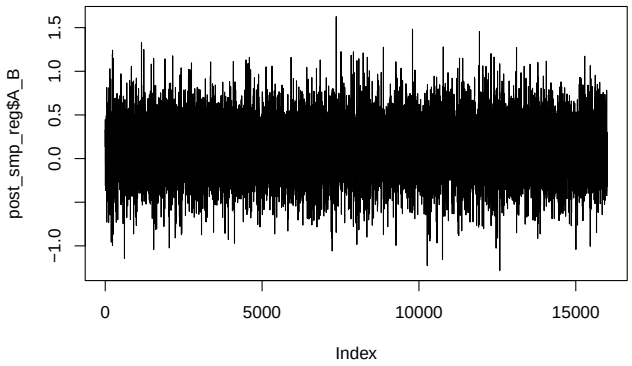
Create a contour plot of the posterior predictive mean of conversion as a function of concentration settings

```
# extract the posterior samples for further analysis
# post_smp_reg <- as.data.frame(rstan::extract(reg_fit, permuted = TRUE))

# Extract the summary of the fit
fit_summary <- summary(reg_fit)$summary
# Extract the means of beta[1] to beta[5]
beta_means <- fit_summary[grep("beta", rownames(fit_summary)), "mean"]
# Print the extracted means
print(beta_means)
```

```
##     beta[1]     beta[2]     beta[3]     beta[4]     beta[5]     beta[6]
##  0.09335011  0.34761615 -0.34735433 -1.83775648  0.11104711  2.45076850
```

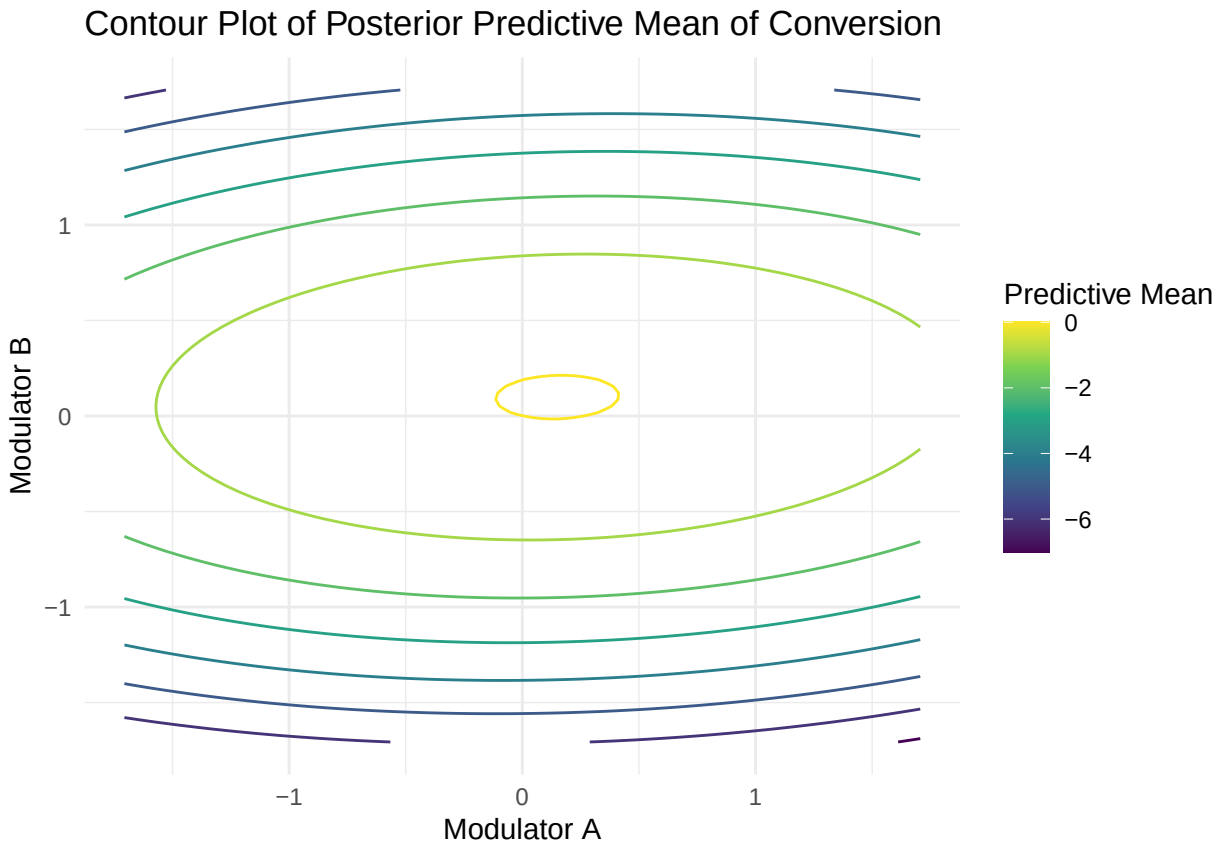
```
# Extract them individually
beta_1_mean <- fit_summary["beta[1]", "mean"]
beta_2_mean <- fit_summary["beta[2]", "mean"]
beta_3_mean <- fit_summary["beta[3]", "mean"]
beta_4_mean <- fit_summary["beta[4]", "mean"]
beta_5_mean <- fit_summary["beta[5]", "mean"]

# rename the columns
colnames(post_smp_reg)[1:5] <- colnames(X)

# plot the posterior distribution of the parameters
plot(post_smp_reg$A, type = "l")
plot(post_smp_reg$B, type = "l")
plot(post_smp_reg$A_sq, type = "l")
plot(post_smp_reg$B_sq, type = "l")
plot(post_smp_reg$A_B, type = "l")
```





Once we have the posterior distributions of the parameters, we can use them to generate the posterior predictive distribution of conversion for any given pair of concentrations. Here, I created a contour plot of the posterior predictive mean of conversion as a function of concentration settings to visualize the relationship between the modulators and conversion.

```r
# Define a grid of concentration settings
modulator_A_seq <- seq(0, 75, length.out = 100)
modulator_B_seq <- seq(0, 30, length.out = 100)

# standardize the data
modulator_A_seq_standardized <- (modulator_A_seq - mean(modulator_A_seq)) / sd(modulator_A_seq)
modulator_B_seq_standardized <- (modulator_B_seq - mean(modulator_B_seq)) / sd(modulator_B_seq)

grid <- expand.grid(modulator_A = modulator_A_seq_standardized, modulator_B = modulator_B_seq_standardi

# Compute the posterior predictive mean of conversion for each pair of concentration settings
predictive_means <- apply(grid, 1, function(row) {
  modulator_A <- row["modulator_A"]
  modulator_B <- row["modulator_B"]

  # Compute the predictive mean using the model equation and the posterior samples
  mean_predictive <- mean(post_smp_reg$A * modulator_A +
                          post_smp_reg$B * modulator_B +
                          post_smp_reg$A_sq * modulator_A^2 +
                          post_smp_reg$B_sq * modulator_B^2 +
                          post_smp_reg$A_B * (modulator_A * modulator_B))
  return(mean_predictive)
})

# Add the computed predictive means to the grid
grid$predictive_mean <- predictive_means

# Plot the contour
ggplot(grid, aes(x = modulator_A, y = modulator_B, z = predictive_mean)) +
  geom_contour(aes(color = ..level..)) +
  scale_color_viridis_c() +
  labs(title = "Contour Plot of Posterior Predictive Mean of Conversion",
       x = "Modulator A",
       y = "Modulator B",
       color = "Predictive Mean") +
  theme_minimal()
```



Based on the countour plot, I decided to mostly include the data points which are located within the light green area where the conversion is high. The range of Modulator A is [34.20324, 46.29136] and the range of Modulator B is [9.725183, 22.033089].
I will use these ranges to create the second experiment design.

```
# Set the range for Modulator A
range_A_standardized <- c(-0.15, 0.4)
# Set the range for Modulator B
range_B_standardized <- c(-0.6, 0.4)

# De-standardize the range for Modulator A
range_A_original <- (range_A_standardized * sd(modulator_A_seq)) + mean(modulator_A_seq)
# De-standardize the range for Modulator B
range_B_original <- (range_B_standardized * sd(modulator_B_seq)) + mean(modulator_B_seq)

# randomly select the values of the modulators within the range
set.seed(46)
rangeA <- runif(48, min = 34.20324, max = 46.29136)
rangeB <- runif(48, min = 9.725183, max = 22.033089)

# Create a DataFrame
design2 <- data.frame(A = rangeA, B = rangeB)
# write a table and save it to csv
write.table(design2, file = "/Users/satoshiido/Documents/statistical-analysis/656/hw/hw2/outputs/design1
```

Third design selection

Post experimental analysis After the second experiment, I have a total of 72 wells (24 wells from the first experiment and 48 wells from the second experiment). I use the results both from the first experiment and the second experiment to build a Bayesian regression model. Based on the model, I create the posterior predictive distribution of the concentration settings that yield maximum conversion and the distribution of the conversion corresponding to a specific point prediction of the concentration.

```
# resulf of the first experiment
design2_result <- read.table("/Users/satoshiido/Documents/statistical-analysis/656/hw/hw2/data/rslt2.cs
colnames(design2_result) <- c("A", "B", "y")
design2_result
```

```
##           A         B         y
## 1  36.43167 14.234562 1.2106188
## 2  37.14504 11.403157 1.0175362
## 3  41.26267 12.116437 1.2803667
## 4  38.38125 20.813234 2.8354485
## 5  37.02130 13.983985 1.9780103
## 6  42.12194 15.694542 2.1685976
## 7  44.98199 17.990600 1.3811351
## 8  41.37615 17.601736 1.3620833
## 9  44.82587 10.742672 1.5649440
## 10 45.66939 11.302058 1.6201618
## 11 37.42849 19.191965 1.4810199
## 12 35.44806 14.093865 1.1514140
## 13 42.25537 12.646631 1.1183535
## 14 34.92814 17.543906 1.0596195
## 15 43.44457 17.290116 1.6554864
## 16 42.17368 11.065775 1.0729409
## 17 45.75193 13.604550 1.5689524
## 18 35.83337 11.161713 1.9506709
## 19 41.42546 13.799584 1.2860021
## 20 40.75681 16.137796 1.2603309
## 21 43.51787 20.072959 1.5303147
## 22 36.52622 14.462030 0.9851038
## 23 46.12361  9.797949 1.9684172
## 24 36.37072 14.727138 1.2688693
## 25 34.53133 21.695453 2.3773087
## 26 37.58054 14.266104 0.4964904
## 27 38.48951 14.417665 0.5767935
## 28 35.70443 18.675620 1.4221261
## 29 39.24191 12.805504 0.1274957
## 30 39.28627 21.244311 2.8192873
## 31 36.36467 21.809617 2.4392764
## 32 44.22214 21.204542 2.7837256
## 33 40.54805 19.233203 1.6851171
## 34 46.13605 14.090806 0.9117426
## 35 42.81370 18.976224 1.6217187
## 36 37.78682 11.623642 1.7003947
## 37 38.37417 10.830786 1.8212186
## 38 40.61705 17.085486 0.9678136
## 39 42.21109 15.346512 1.9784919
## 40 40.75150 20.847742 2.6762827
```

```
## 41 38.32654 21.166076 1.9626742
## 42 36.96757 20.163179 1.7626139
## 43 36.82789 14.154881 0.8717500
## 44 43.76135 11.180679 1.7785373
## 45 40.19266 20.807527 2.2817213
## 46 34.68403  9.822053 0.6525697
## 47 45.65693  9.927693 1.7756978
## 48 44.65715 12.102590 1.4641594
```

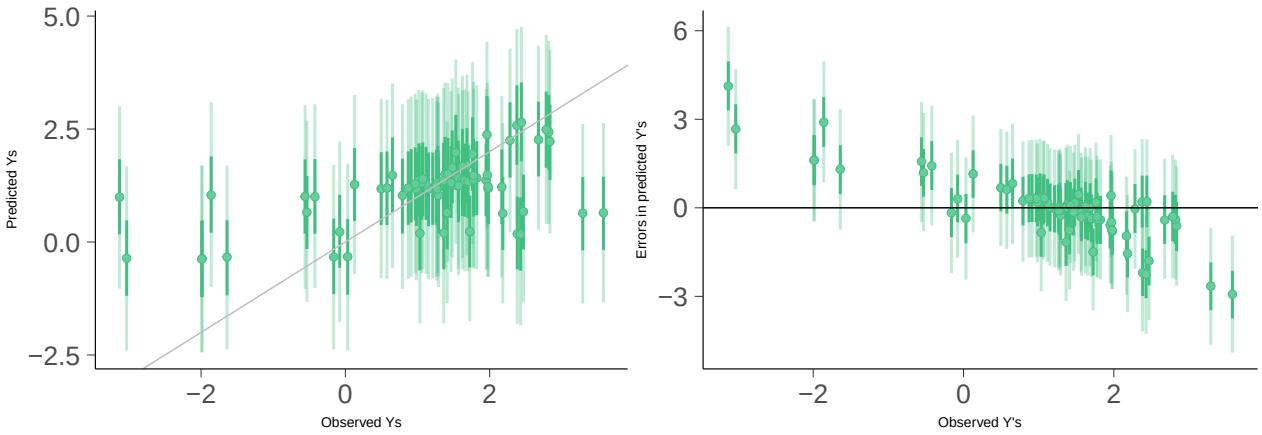
```
# Combine two results
y2 <- c(design1_result$y, design2_result$y)
A2 <- c(design1_result$A, design2_result$A)
B2 <- c(design1_result$B, design2_result$B)

# standardize the data
A2 <- (A2 - mean(A2)) / sd(A2)
B2 <- (B2 - mean(B2)) / sd(B2)

# combine them
X2 <- data.frame(cbind(B2, A2 * B2, B2^2, B2^3))
X2[, "Offset"] <- 1
colnames(X2) <- c("B", "A_B", "B_sq", "B_3", "Offset")
```

```
# compile the model
reg2_data <- list(N = nrow(X2), p = ncol(X2), x = X2, y = y2, pr_sd = 100)
# Fit the model and sampling
reg_fit2 <- rstan::sampling(
  object = reg_half_cauchy,
  data = reg2_data,
  iter = 10000,
  warmup = 2000,
  chains = 2,
  seed = 9
)

post_smp_reg2 <- as.data.frame(reg_fit2)
# rename the columns
colnames(post_smp_reg2)[1:5] <- colnames(X2)
# extract the posterior prediction for further analysis
y_rep <- extract(reg_fit2)$y_rep
```

We can check interval estimates for each predicted observations of the model and an error in the predicted Y's. From the plot, we can see that the model is not really good at predicting the conversion when the conversion is high, yet we still proceed with this model.

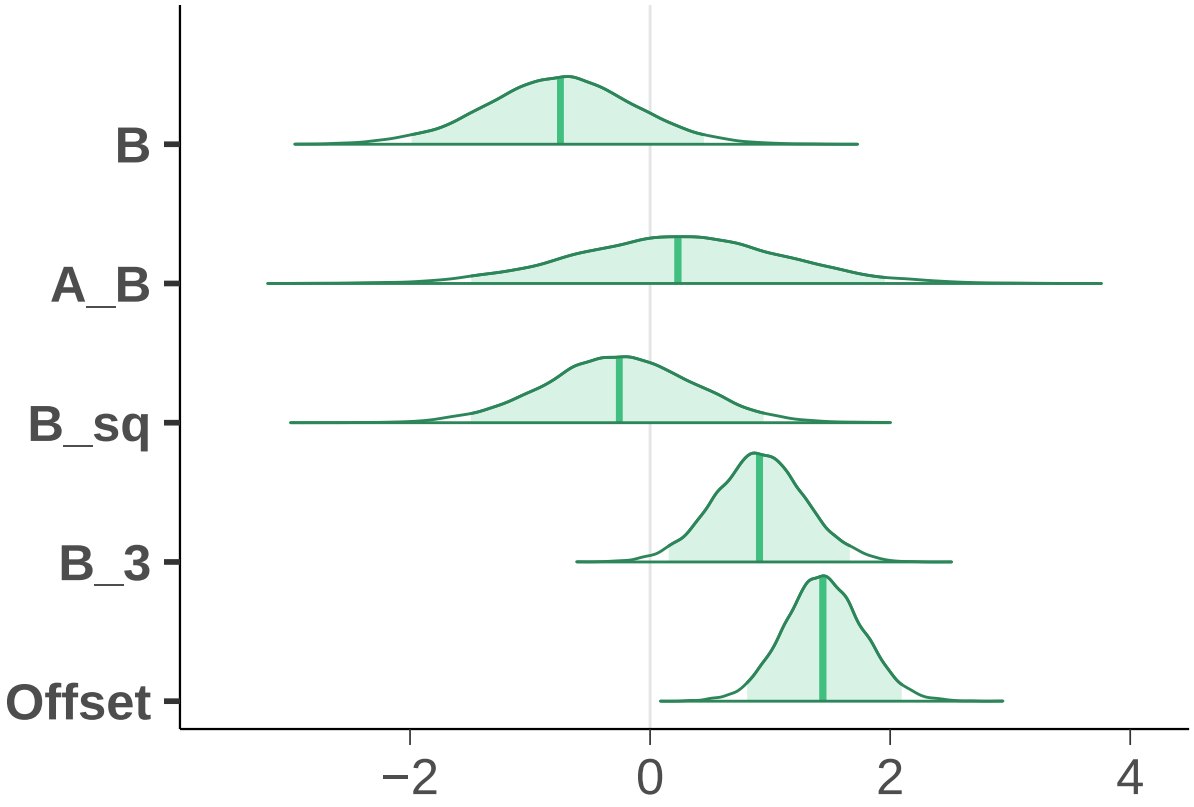
```
## Interval estimates for each predicted observations (columns) in y_pred
ppd_intervals(y_rep, x = y2) +
  geom_abline(intercept = 0, slope = 1, color = "grey") +
  ggplot2::labs(y = "Predicted Ys", x = "Observed Ys") +
  theme(axis.title.x = element_text(size = 10),
        axis.title.y = element_text(size = 10))

ppd_intervals(t(t(y_rep) - y2), x = y2) +
  geom_abline(intercept = 0, slope = 0) +
  ggplot2::labs(y = "Errors in predicted Y's", x = "Observed Y's") +
```

```
  theme(axis.title.x = element_text(size = 10),
        axis.title.y = element_text(size = 10))
```



The regression coefficients was investigated to verify that some of the parameters in the model are significant at a 5% significance level.

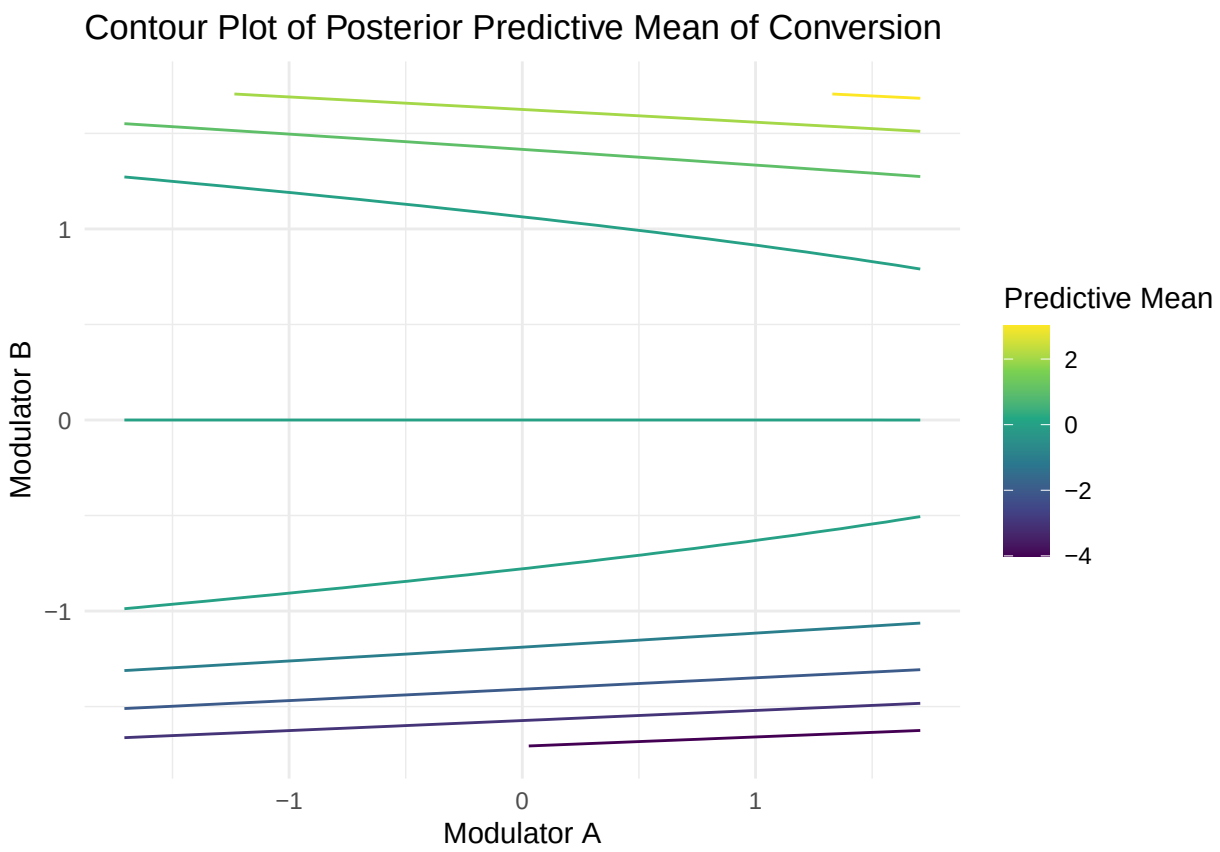
```
mcmc_areas(post_smp_reg2[1:5], pars = colnames(X2)[1:5], prob = 0.95)
```



From the outputs above, $f(x, \beta) = \beta_0 + \beta_1 \times B + \beta_2 \times A\_B + \beta_3 \times B^2 + \beta_4 \times B^3$ as the optimal fitting model, given the available data.

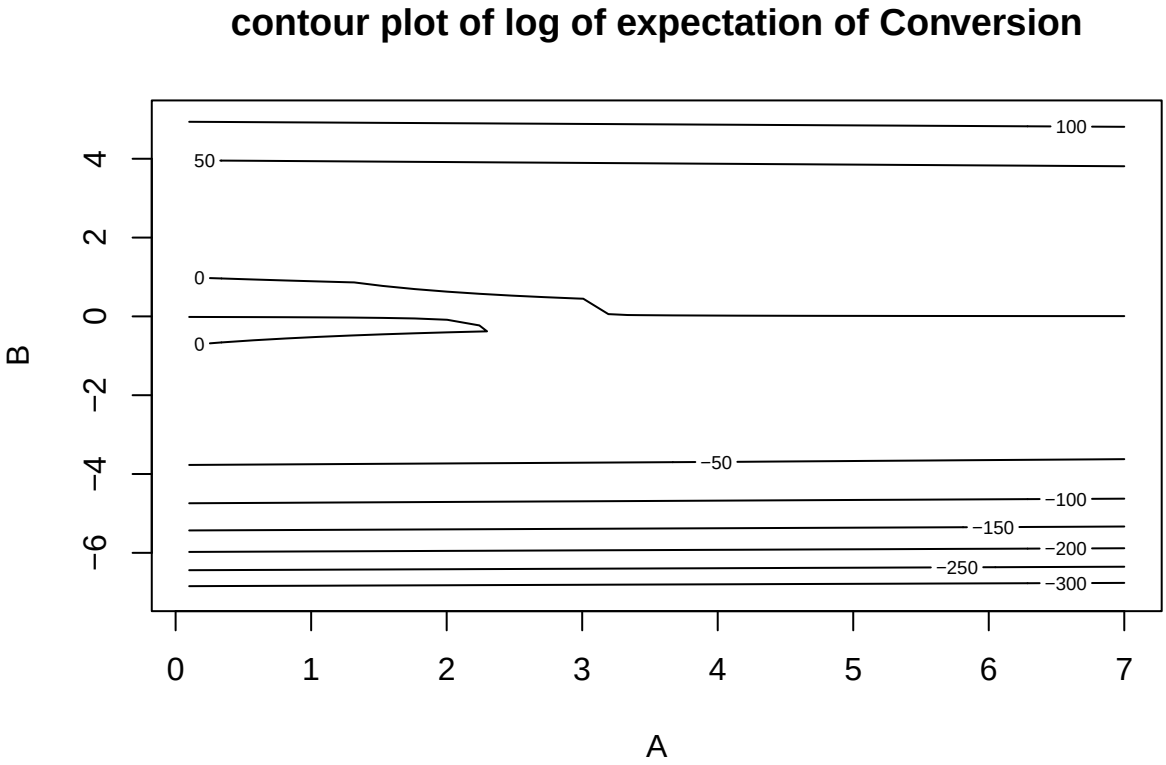
```
# Define a grid of concentration settings
modulator_A_seq <- seq(0, 80, length.out = 100)
modulator_B_seq <- seq(0, 30, length.out = 100)
```

```r
# standardize the data
modulator_A_seq_standardized <- (modulator_A_seq - mean(modulator_A_seq)) / sd(modulator_A_seq)
modulator_B_seq_standardized <- (modulator_B_seq - mean(modulator_B_seq)) / sd(modulator_B_seq)

grid <- expand.grid(modulator_A = modulator_A_seq_standardized, modulator_B = modulator_B_seq_standardi

# Compute the posterior predictive mean of conversion for each pair of concentration settings
predictive_means <- apply(grid, 1, function(row) {
  modulator_A <- row["modulator_A"]
  modulator_B <- row["modulator_B"]

  # Compute the predictive mean using the model equation and the posterior samples
  mean_predictive <- mean(post_smp_reg2$B * modulator_B +
                          post_smp_reg2$A_B * (modulator_A * modulator_B) +
                          post_smp_reg2$B_sq * modulator_B^2 +
                          post_smp_reg2$B_3 * modulator_B^3
                          )
  return(mean_predictive)
})

# Add the computed predictive means to the grid
grid$predictive_mean <- predictive_means

# Plot the contour
ggplot(grid, aes(x = modulator_A, y = modulator_B, z = predictive_mean)) +
  geom_contour(aes(color = ..level..)) +
  scale_color_viridis_c() +
  labs(title = "Contour Plot of Posterior Predictive Mean of Conversion",
       x = "Modulator A",
       y = "Modulator B",
       color = "Predictive Mean") +
  theme_minimal()
```



Now contour plots will be generated to investigate the areas of maximum log of `E[exp(conversion rate)]` according to the chosen model.

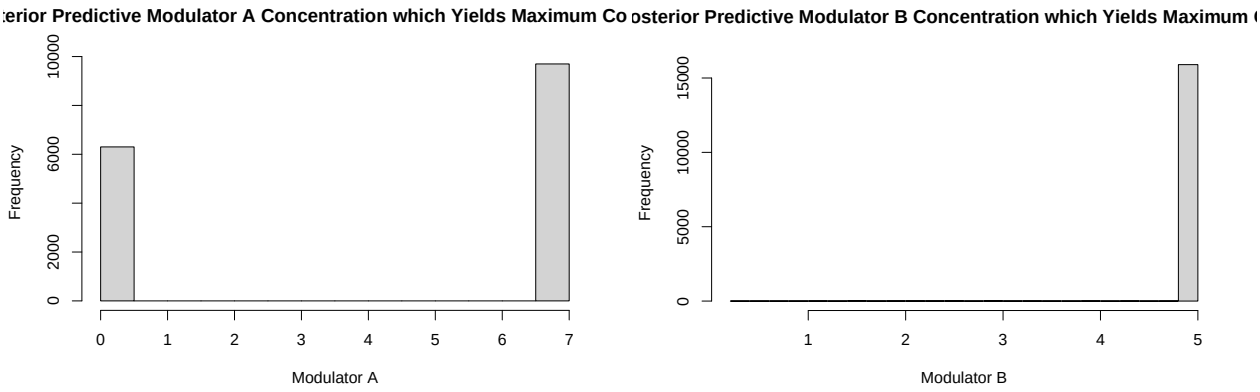
```
beta <- extract(reg_fit2)$beta

# Define a grid of concentration settings
A_seq <- seq(0.1, 7, length.out = 30)
B_seq <- seq(-7, 5, length.out = 30)

# Function to compute the log of the expectation
log_exp_func <- function(A, B, beta){
  mu_predictive <- mean(beta[,1]) * B + mean(beta[, 2]) * A * B + mean(beta[, 3]) * B^2 + mean(beta[, 4
  expected_val <- mean(exp(mu_predictive))
  return(log(expected_val))
}

# we take a function that works on scalars and vectorize it so that it can work on vectors in an elemen
intensity <- outer(A_seq, B_seq, Vectorize(function(A, B) log_exp_func(A, B, post_smp_reg2)))

# Now, plot the results with contour
contour(
  A_seq, B_seq, intensity, nlevels=10, xlab = "A", ylab = "B",
  main = "contour plot of log of expectation of Conversion"
)
```



Based on the Bayesian model which I built above, I create the posterior predictive distribution of the concentration settings that yield maximum conversion.

```
beta <- extract(reg_fit2)$beta

conversion_rate <- function(A, B, index) {
  X <- cbind(B, A * B, B^2, B^3, 1)
  # mean(beta[,1]) * B + mean(beta[, 2]) * A * B + mean(beta[, 3]) * B^2 + mean(beta[, 4]) * B^3
  conversion_rate <- X%*%beta[index, ]
  return(conversion_rate)
}


max_conversion_rates <- function(A_seq, B_seq) {
  rates <- matrix(0, nrow = length(A_seq), ncol = length(B_seq))
  for (i in 1:nrow(beta)) {
    rates <- rates + outer(A_seq, B_seq, conversion_rate, i)
  }
  return(rates / nrow(beta))
}

max_concentration <- function(A_seq, B_seq, index) {
  rates <- outer(A_seq, B_seq, conversion_rate, index)
  m <- which(rates == max(rates), arr.ind = TRUE)[1, ]
  return(c(A_seq[m[1]], B_seq[m[2]])
  )
}

A_seq <- seq(0.1, 7, length.out = 40)
B_seq <- seq(0, 5, length.out = 40)
```

```r
max_concentration_set <- matrix(nrow = nrow(beta), ncol = 2)
for (i in 1:nrow(beta)) {
  max_concentration_set[i, ] <- max_concentration(A_seq, B_seq, i)
}


hist(max_concentration_set[,1] * sd(A2) + mean(A2), breaks=20, main="Posterior Predictive Modulator A C
hist(max_concentration_set[,2] * sd(B2) + mean(B2), breaks = 20, main = "Posterior Predictive Modulator
```

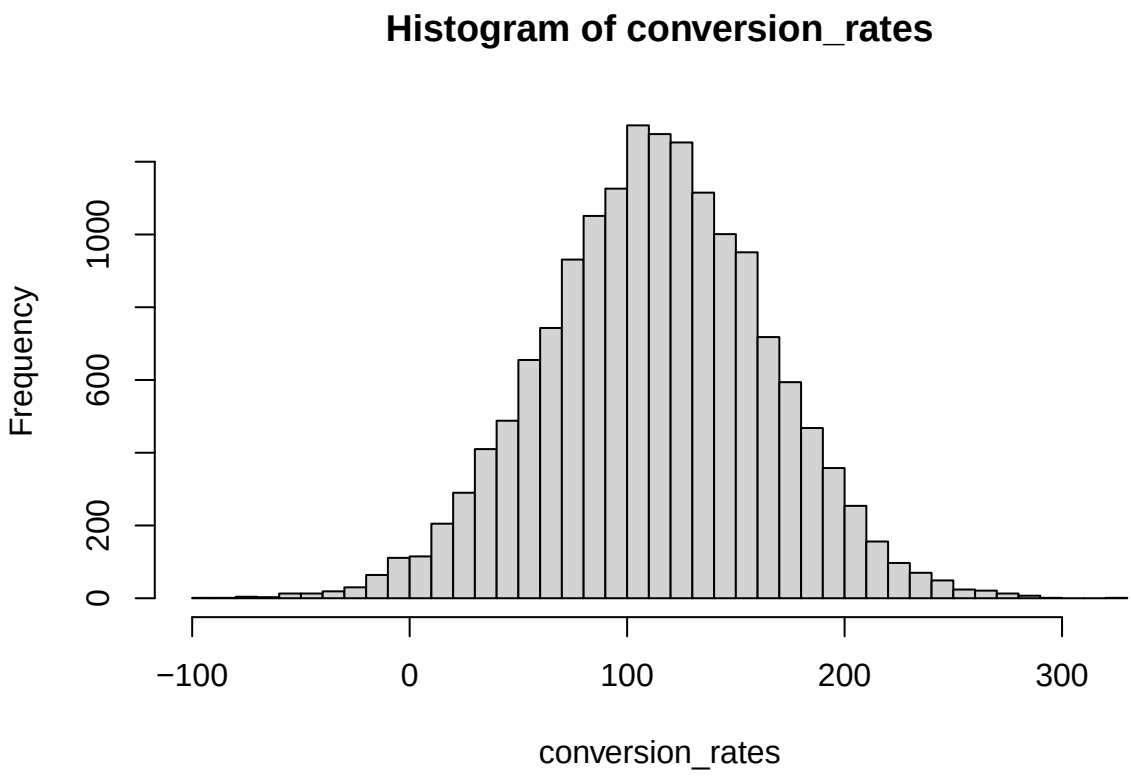


Finally, we plot the distribution of the conversion corresponding to a specific point prediction of the concentration.

```r
avg_rate <- max_conversion_rates(A_seq, B_seq)
index <- which(avg_rate == max(avg_rate), arr.ind = TRUE)[1, ]
A_max <- A_seq[index[1]] * sd(A2) + mean(A2)
B_max <- B_seq[index[2]] * sd(B2) + mean(B2)

conversion_rates <- matrix(nrow = nrow(beta), ncol = 1)
for (i in 1:nrow(beta)) {
  conversion_rates[i] <- conversion_rate((A_max - mean(A2)) / sd(A2), (B_max - mean(B2)) / sd(B2), i)
}
```

The histogram below shows the posterior predictive distribution of the conversion corresponding to the point prediction of concentrations which yields maximum conversion.

```r
hist(conversion_rates, breaks = 30)
```


Histogram of conversion_rates
Frequency
0
200
600
1000
−100
0
100
200
300
conversion_rates