STAT 656: Bayesian Data Analysis
Fall 2024
Homework 2

Juanwu Lu*

```
library("bayesplot")
library("ggplot2")
library("patchwork")
library("rstan")
bayesplot_theme_set(theme_default(base_size = 12, base_family = "sans"))
```

Synthetic Data

The file `hw2_synthetic.csv` is a dataset of count-valued measurements $\boldsymbol{y} = \{y_1, \ldots, y_n\}$, 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 $\boldsymbol{x} = \{x_1, \ldots, x_n\}$'s as $\boldsymbol{x}$. 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 $f(x_i, \beta) = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \beta_3 x_{i,1}^2 + \beta_4 x_{i,2}^2 + \beta_5 x_{i,1} x_{i,2}$.

1. (25 points) 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 β is, and how you arrived at your conclusion, including any useful figures (especially of the posterior distribution). You can use Stan.

   Solution:

   ```
   # Read the data
   if (file.exists("data/hw2_synthetic.csv")) {
     data <- read.csv("data/hw2_synthetic.csv", header = TRUE)
   } else {
     stop("FileNotFound: data file not found at 'data/hw2_synthetic.csv'.")
   }
   summary(data)
   ```

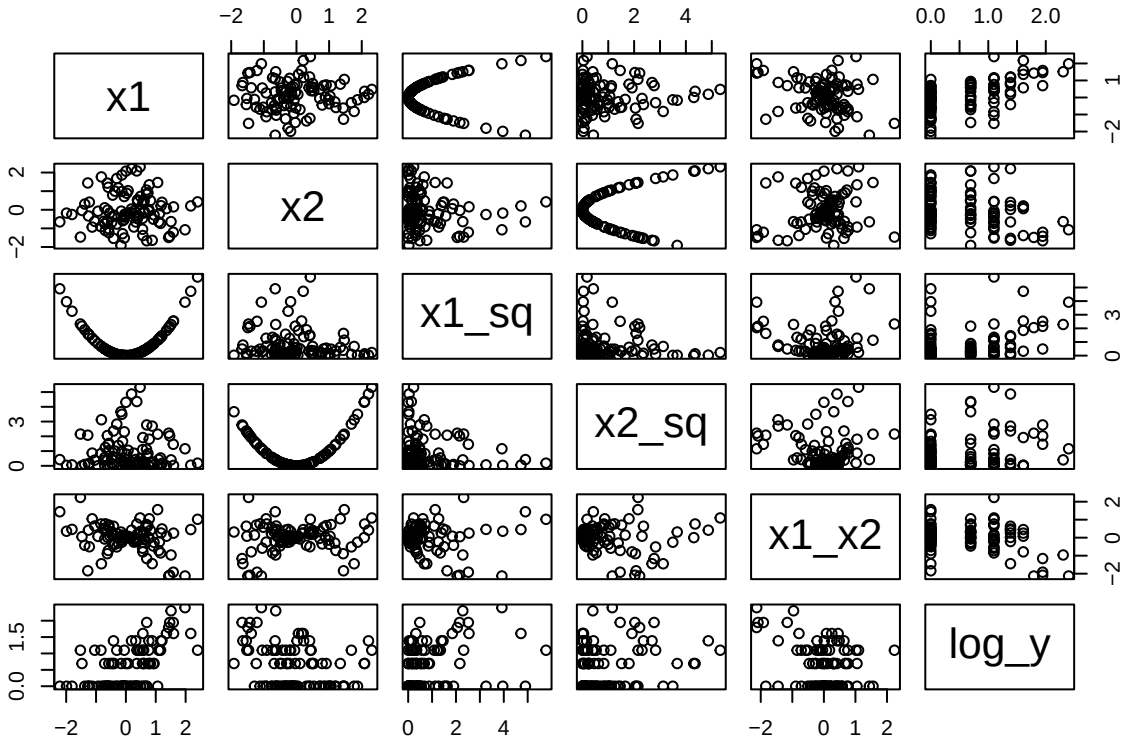
   ```
   ##        x1                 x2                  y
   ##  Min.   :-2.2147   Min.   :-1.91436   Min.   : 0.00
   ##  1st Qu.:-0.4942   1st Qu.:-0.65105   1st Qu.: 0.00
   ##  Median : 0.1139   Median :-0.17722   Median : 1.00
   ##  Mean   : 0.1089   Mean   :-0.03781   Mean   : 1.29
   ##  3rd Qu.: 0.6915   3rd Qu.: 0.50090   3rd Qu.: 2.00
   ##  Max.   : 2.4016   Max.   : 2.30798   Max.   :10.00
   ```

   Since the model is a Poisson regression, the logarithm of the response variable is assumed to be a linear combination of the kernel features. The visualization below shows the relationship between the

*College of Engineering, Purdue University, West Lafayette, IN, USA

logarithm of the response variable and the kernel features. It is shown that $\log y$ is roughly positively correlated with x_1 and $x_1 x_2$, and roughly negatively correlated with x_2, x_1^2, and x_2^2.

```
# Preprocess data to create the kernel terms
data$x1_sq <- data$x1^2
data$x2_sq <- data$x2^2
data$x1_x2 <- data$x1 * data$x2
data$log_y <- log(data$y + 1)
data <- data[, c("x1", "x2", "x1_sq", "x2_sq", "x1_x2", "y", "log_y")]
plot(
  data[, c("x1", "x2", "x1_sq", "x2_sq", "x1_x2", "log_y")]
)
```



Therefore, the weights β_i can be both positive and negative. Given no prior knowledge on the features, I choose a non-informative prior for the weights, *i.e.,* the isotropic normal distribution: $\mathcal{N}(0, 10^2\mathbf{I})$. The Stan model is implemented as follows:

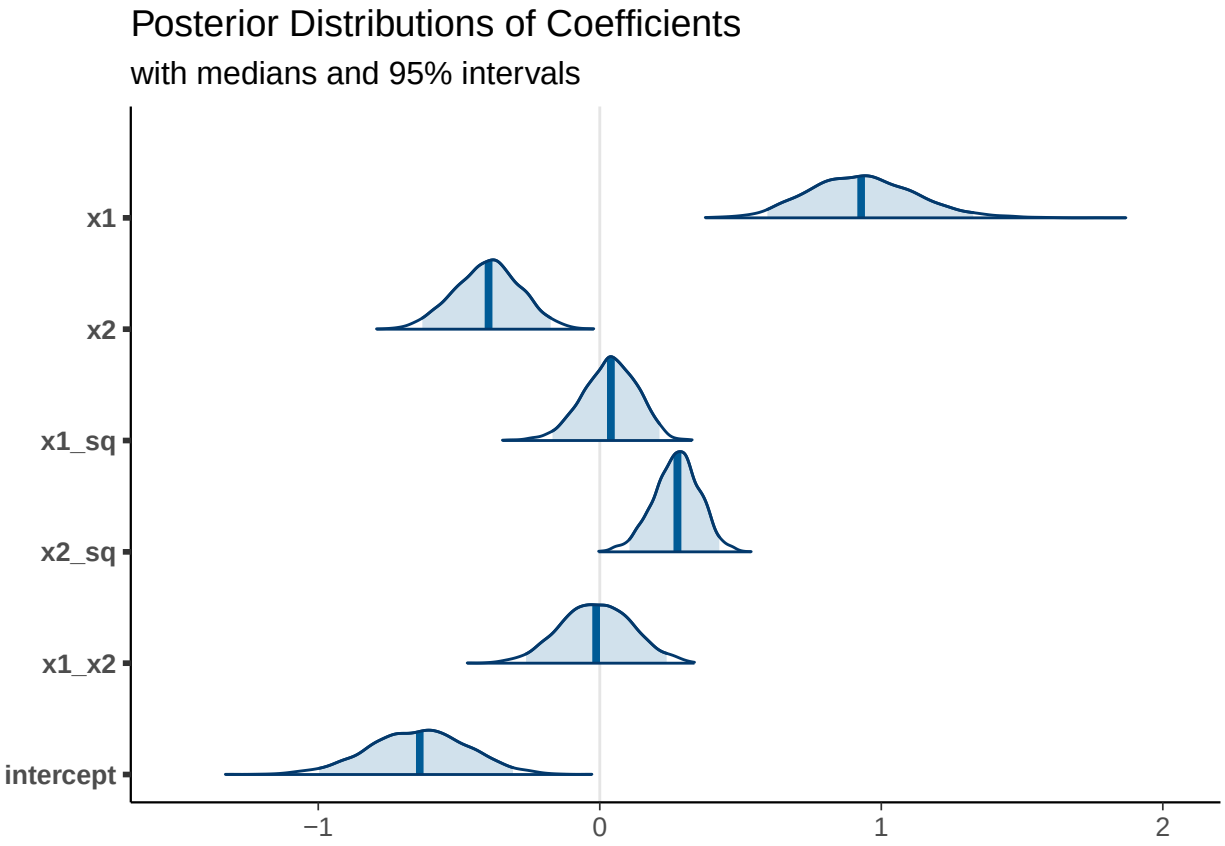
```
linreg_poisson_code <- "
    data {
        int<lower=0> n;          // Number of observations
        int<lower=0> k;          // Number of features
        real<lower=0> pr_std;    // Prior coefficients standard deviation
        matrix[n, k] x;          // Observation matrix
        int<lower=0> y[n];                  // Integer response vector
    }

    parameters {
        vector[k] beta;          // Coefficients
    }

    transformed parameters {
        vector[n] lambda = exp(x * beta);    // Poisson rate
```

```
    }

    model {
        beta ~ normal(0, pr_std);           // Prior on the coefficients
        y ~ poisson(lambda);                // Poisson emission
    }

    generated quantities {
        real y_hat[n];
        y_hat = poisson_rng(lambda);
    }
"
linreg_poisson_model <- stan_model(
  model_name = "poisson_regression",
  model_code = linreg_poisson_code
)
x <- data[, c("x1", "x2", "x1_sq", "x2_sq", "x1_x2")]
x[, "intercept"] <- 1
y <- data$y
reg_data <- list(n = nrow(x), k = ncol(x), pr_std = 10.0, x = x, y = y)
nfit <- sampling(
  linreg_poisson_model,
  data = reg_data,
  iter = 5000,
  warmup = 2000,
  chains = 1,
  seed = 42,
  show_messages = FALSE,
)
```

The visualization below shows the posterior distributions of the coefficients $\boldsymbol{\beta} = \{\beta_0, \ldots, \beta_5\}$ with 95% intervals. Based on the results, the coefficients of x_1^2, and $x_1 x_2$ are close to zero while the coefficients of x_1, x_2, and x_2^2 are different from zero. Meanwhile, the uncertainty of the coefficient posterior for x_1 are significantly larger than the others. Based on the result:

- the terms x_1 and x_2^2 are positively correlated with the response variable,
- the term x_2 is negatively correlated with the response variable, and
- the terms x_1^2 and $x_1 x_2$ are not important in the expression for $f(x)$. Therefore, we can simpilify the model by discarding the terms x_1^2 and $x_1 x_2$.

```
samples <- as.data.frame(nfit)
colnames(samples)[seq_len(ncol(x))] <- colnames(x)
mcmc_areas(
  samples[, seq_len(ncol(x))],
  pars = colnames(x),
  prob = 0.95,
) + labs(
  title = "Posterior Distributions of Coefficients",
  subtitle = "with medians and 95% intervals"
)
```



2. (25 points) 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. Note that you are using the data twice, once to select the model and next to fit it, but we will not worry about that. Compare the posteriors for both models.

Solution:

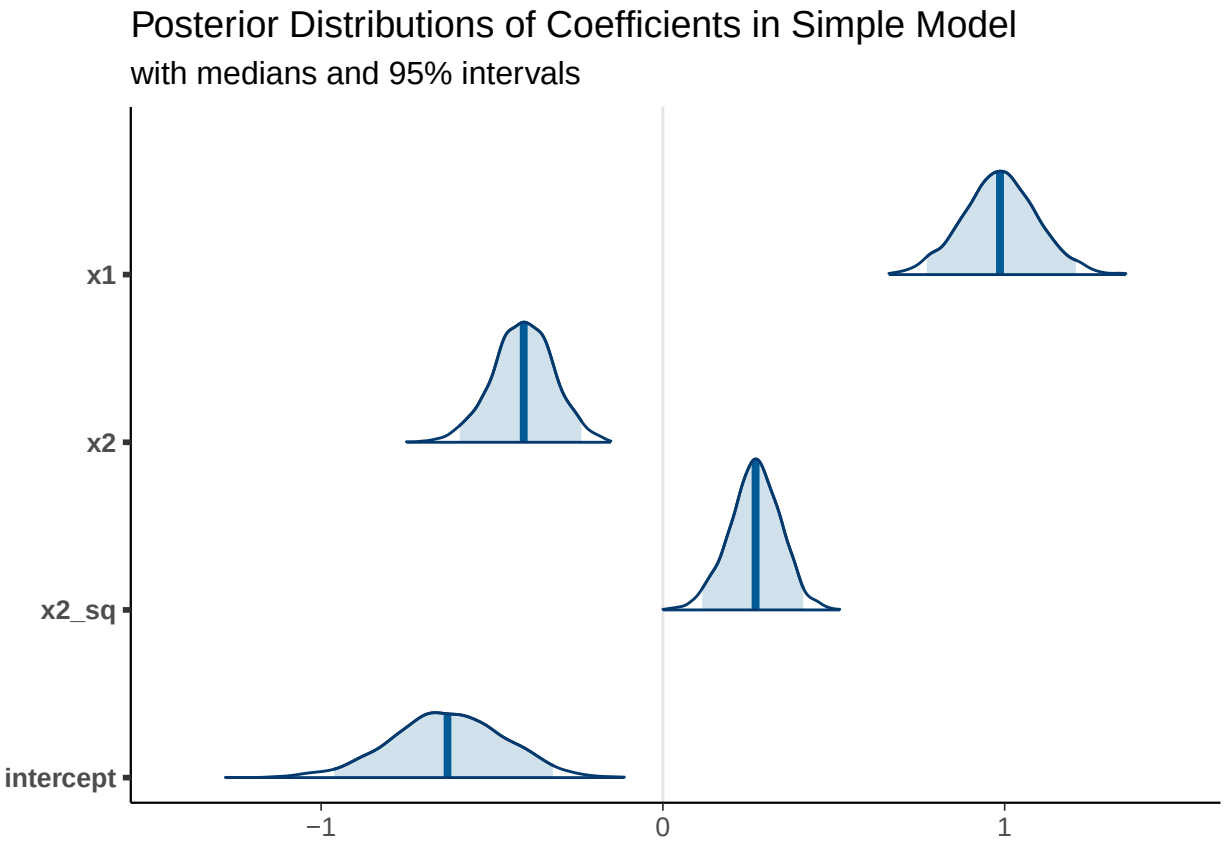
Based on the analysis in the previous question, the terms x_1, x_2, and x_2^2 are kept in the model. The new stan model is implemented as follows:

```
x_simple <- x[, c("x1", "x2", "x2_sq", "intercept")]
reg_data_simple <- list(
  n = nrow(x_simple),
  k = ncol(x_simple),
  pr_std = 10.0,
  x = x_simple,
  y = y
)
nfit_simple <- sampling(
  linreg_poisson_model,
  data = reg_data_simple,
  iter = 5000,
  warmup = 2000,
  chains = 1,
  seed = 42,
  show_messages = FALSE,
)
```

The visualization below shows the posterior distributions of the coefficients $\boldsymbol{\beta}' = \{\beta_0, \beta_1, \beta_2, \beta_4\}$ with

95% intervals. The results show that the coefficients of x_1, x_2, and x_2^2 are all significantly different from zero (*i.e.*, with zero outside its 95% posterior interval).

```
samples_simple <- as.data.frame(nfit_simple)
colnames(samples_simple)[seq_len(ncol(x_simple))] <- colnames(x_simple)
mcmc_areas(
  samples_simple[, seq_len(ncol(x_simple))],
  pars = colnames(x_simple),
  prob = 0.95,
) + labs(
  title = "Posterior Distributions of Coefficients in Simple Model",
  subtitle = "with medians and 95% intervals"
)
```



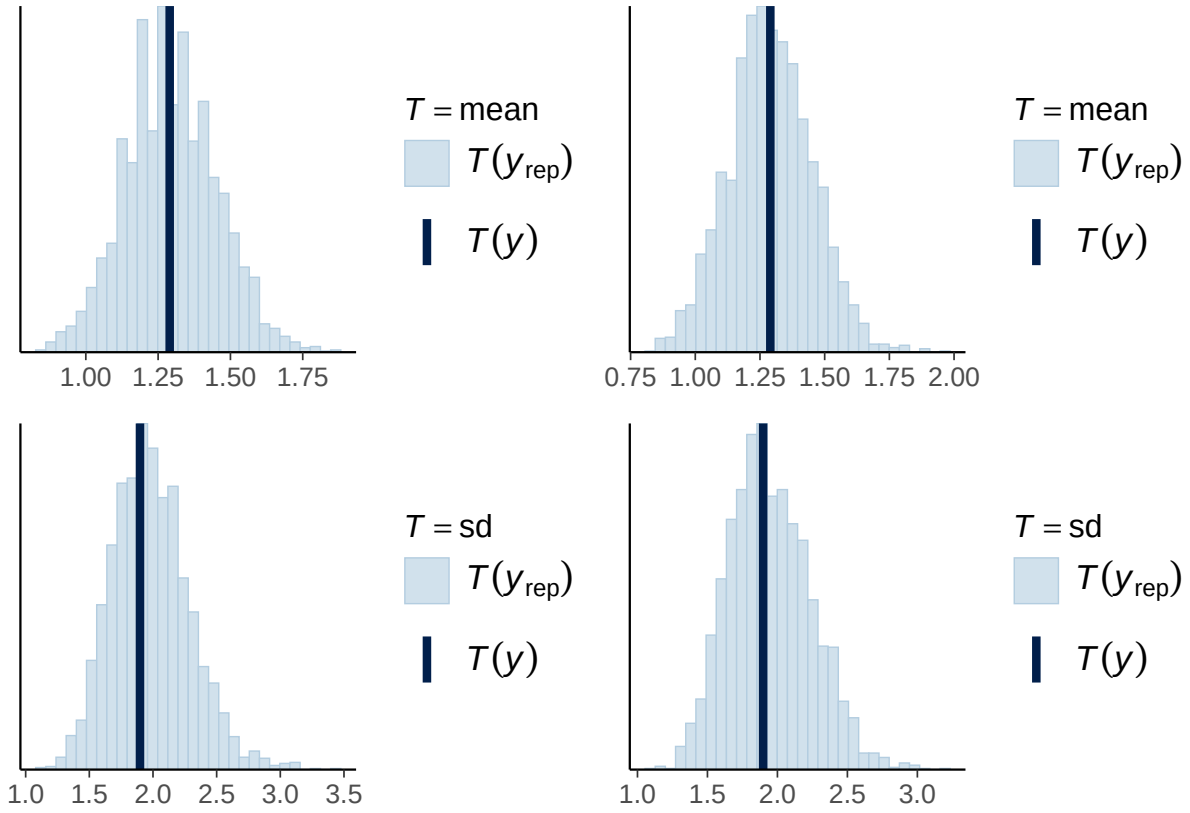
Compared to the model from the previous question, the posterior distributions of the coefficients in this simpler model are more concentrated around its mean, which indicates that the simpler model is more confident about the estimation (*i.e.* with lower posterior uncertainty).

3. (25 points) Perform posterior predictive checks for both models, being sure to explain what you are doing. Which model do you think fits the data better?

   Solution:

   The posterior predictive checks (PPC) are performed to evaluate the goodness-of-fit of the models. The PPC is done by comparing the mean and standard deviation of observed data with ones of the data simulated from the posterior predictive distribution. Figure 3 shows that the simpler model fits data slightly better than the full model for that the average mean and standard deviation of the simulated data are closer to the observed data in the simpler model. Moreover, compared to the original model, posterior predictive samples distribute more symmetric around the ground-truth mean.

```
y_hat <- extract(nfit)$y_hat
y_hat_simple <- extract(nfit_simple)$y_hat
plot1 <- ppc_stat(
  y = data$y, yrep = y_hat
)
plot2 <- ppc_stat(
  y = data$y, yrep = y_hat_simple
)
plot3 <- ppc_stat(
  y = data$y, yrep = y_hat, stat = "sd"
)
plot4 <- ppc_stat(
  y = data$y, yrep = y_hat_simple, stat = "sd"
)
(plot1 + plot2) / (plot3 + plot4)
```



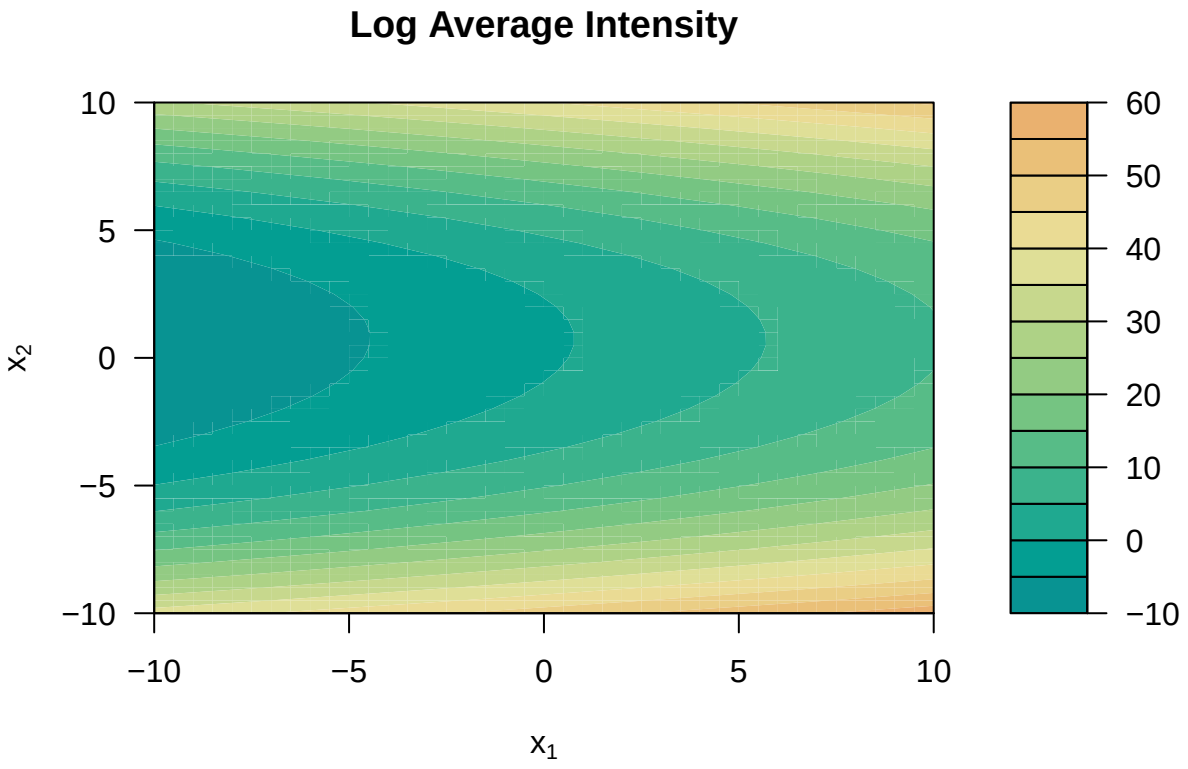
4. (25 points) Use the results from the second model to create a contour plot showing the log average Poisson intensity as a function of x. In other words, plot $\log \mathbb{E}_{p(\beta|\boldsymbol{x},\boldsymbol{y})}[\exp(f(\boldsymbol{x},\boldsymbol{\beta}))]$ as a function of the two components of x (you can restrict the component ranges from -10 to $+10$).

Solution:

The code below generates the contour plot as requested.

```
log_avg_intensity <- function(x1, x2, weights) {
  intensity <- weights[, 1] * x1 +
    weights[, 2] * x2 +
    weights[, 3] * x2^2 +
    weights[, 4]
```

```r
  out <- log(mean(exp(intensity)))
  return(out)
}

x1 <- seq(-10, 10, 0.5)
x2 <- seq(-10, 10, 0.5)
grid <- expand.grid(x1 = x1, x2 = x2)
weight_samples <- samples_simple[, 1:4]
z <- mapply(
  function(x1, x2) log_avg_intensity(x1, x2, weight_samples),
  grid$x1,
  grid$x2
)
z <- matrix(z, nrow = length(x1), ncol = length(x2))
col <- hcl.colors(20, "Temps")
filled.contour(
  x1,
  x2,
  z,
  col = col,
  xlab = expression("x"[1]),
  ylab = expression("x"[2]),
  main = "Log Average Intensity"
)
```



Applied Problem

Scientists at the Notlem lab are researching the conversion of stem cells into pancreatic β-cells to treat patients with type 1 diabetes. They recently developed two new chemical modulators for improved conversion, and wish to identify their **concentration settings that maximize conversion**.

The scientists also state that:

- their resource constraints limit the *number of plates to* 3. Since each plate has 24 wells, you can get **at most** 72 **measurements**.
- their time constraints limit the *number of experimental runs to* 2. Thus you can tell the scientists 24 pairs of concentrations followed by 48, or you can tell them 48 pairs followed by 24. In either case, you can wait for their measurements for the first set before choosing the concentrations for the second.
- you should only consider concentrations between 0 **and** 80 **for the first chemical modulator**, and between 0 **and** 30 **for the second chemical modulator**, because anything outside this region is not thought to improve conversion. and that
- **polynomial models of conversion** as a function of chemical modulators' concentrations (*e.g.,* $f(x, \beta)$ *like the previous question, but possibly with cubic or even higher terms*) have been shown to enjoy some measure of success for this type of problem in previous literature. (Unlike the earlier question, here the measurements are **real valued**, so you don't need the exp and Poisson parts, just Gaussian noise.)

Your task in this problem is to:

- tell the scientists the specific pairs of concentrations to run in the study, explaining your thinking
- analyze the resulting (cumulative) data to build a Bayesian regression model of stem cell conversion as a function of the two chemical modulators' concentrations,
- use the model to create a **contour plot** of the posterior predictive mean of conversion as a function of concentration settings,
- use the model to calculate the **posterior predictive distribution** of the concentration settings that yield maximum conversion, and finally
- construct the posterior predictive distribution of the conversion corresponding to *a specific point prediction of the concentration* that yields **maximum conversion**.

Solution:

In this problem, we formulate the Gaussian linear regression model. Denote the conversion rate as $\boldsymbol{y} = \{y_1, \ldots, y_n\}$, and the concentration of the pair of chemical modulators as $\boldsymbol{x}_i = (x_{i,1}, x_{i,2})$, the model is formulated as

$$y_i|\beta \sim \mathcal{N}(f(\boldsymbol{x}_i, \beta), \sigma^2).$$

Since we have already known that the concentration of the first chemical modulator is between 0 and 80, and the concentration of the second chemical modulator is between 0 and 30, it is reasonable to first scale the two features to the range of $[0, 1]$. Then, we can expand the features through polynomial expansion.

```
# Read data
if (file.exists("data/hw2_init_result.csv")) {
  df_1 <- read.csv("data/hw2_init_result.csv", header = FALSE, sep = " ")
} else {
  stop("FileNotFound: data not found at 'data/hw2_init_result.csv'.")
}
colnames(df_1) <- c("x1", "x2", "y")
summary(df_1)
```

```
##        x1                x2               y
##  Min.   : 0.08648   Min.   : 0.0000   Min.   :0.645
##  1st Qu.:22.43609   1st Qu.: 0.6563   1st Qu.:2.213
##  Median :40.97810   Median :15.3864   Median :2.784
##  Mean   :40.60699   Mean   :15.2178   Mean   :2.884
```

```
##  3rd Qu.:61.10781   3rd Qu.:29.4811   3rd Qu.:3.627
##  Max.   :78.62772   Max.   :30.6563   Max.   :4.806
```

```r
# Preprocess the features through normalization
df_1$x1 <- df_1$x1 / 80
df_1$x2 <- df_1$x2 / 30

# Preprocess the features through polynomial expansion
df_1$x1_sq <- df_1$x1^2
df_1$x2_sq <- df_1$x2^2
df_1$x1_x2 <- df_1$x1 * df_1$x2
df_1$x1_cb <- df_1$x1^3
df_1$x2_cb <- df_1$x2^3
df_1$x1_x2_sq <- df_1$x1 * df_1$x2^2
df_1$x1_sq_x2 <- df_1$x1^2 * df_1$x2
cols <- c(
  "x1",
  "x2",
  "x1_sq",
  "x2_sq",
  "x1_x2",
  "x1_cb",
  "x2_cb",
  "x1_x2_sq",
  "x1_sq_x2",
  "y"
)
```

As in the previous question, we choose a non-informative normal prior over the weights, *i.e.,* the isotropic normal distribution: $\mathcal{N}(0, 10^2\mathbf{I})$ and a non-informative half-Cauchy prior over the noise, *i.e.,* $\sigma \sim \text{Cauchy}(0.0, 1.0)$. The Stan model is implemented as follows:

```
linreg_gaussian_code <- "
    data {
        int<lower=0> n;          // Number of observations
        int<lower=0> k;          // Number of features
        real<lower=0> pr_std;    // Prior coefficients standard deviation
        matrix[n, k] x;          // Observation matrix
        vector[n] y;             // Real-valued response vector
    }

    parameters {
        vector[k] beta;          // Coefficients
        real<lower=0> sigma;     // Noise standard deviation
    }

    transformed parameters {
        vector[n] mu = x * beta;     // Mean of the Gaussian
    }

    model {
        beta ~ normal(0, pr_std);    // Prior on the coefficients
        sigma ~ cauchy(0, 1);        // Prior on the noise
        y ~ normal(mu, sigma);       // Gaussian emission
    }
```

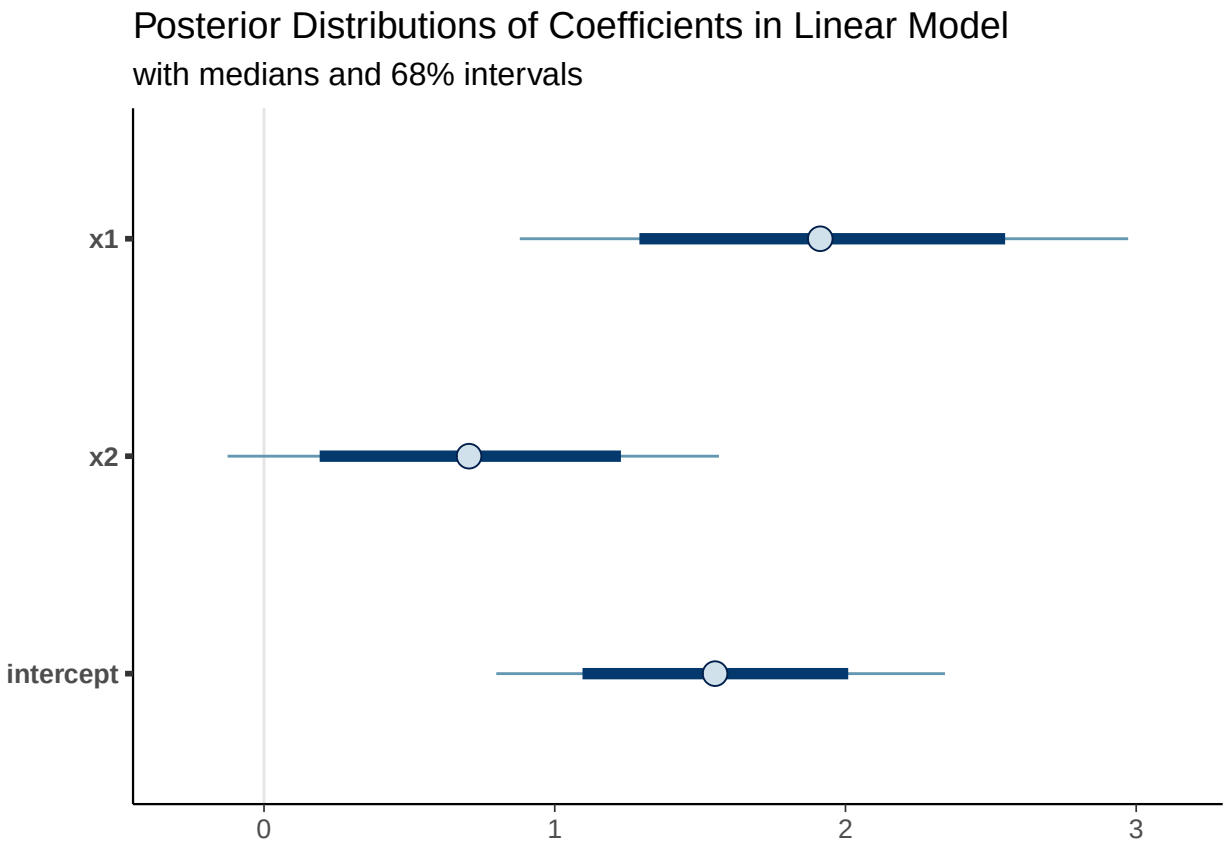
```
    generated quantities {
        real y_hat[n];
        y_hat = normal_rng(mu, sigma);
    }
"
linreg_gaussian_model <- stan_model(
  model_name = "gaussian_regression",
  model_code = linreg_gaussian_code
)
```

In this first version of the model, we investigate only the linear combination of molecules. The following visualization presents the posterior distributions of the coefficients with medians and 68% intervals. Based on the result, **the concentration of two chemical molecules are both positively affecting the conversion rate**.

```
x <- df_1[, cols[1:2]]
x$intercept <- 1
y <- df_1$y
reg_data <- list(n = nrow(x), k = ncol(x), pr_std = 10.0, x = x, y = y)
nfit <- sampling(
  linreg_gaussian_model,
  data = reg_data,
  iter = 10000,
  warmup = 5000,
  chains = 1,
  seed = 42,
  show_messages = FALSE,
)
```

```
##
## SAMPLING FOR MODEL 'gaussian_regression' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 2.9e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.29 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 10000 [  0%]  (Warmup)
## Chain 1: Iteration: 1000 / 10000 [ 10%]  (Warmup)
## Chain 1: Iteration: 2000 / 10000 [ 20%]  (Warmup)
## Chain 1: Iteration: 3000 / 10000 [ 30%]  (Warmup)
## Chain 1: Iteration: 4000 / 10000 [ 40%]  (Warmup)
## Chain 1: Iteration: 5000 / 10000 [ 50%]  (Warmup)
## Chain 1: Iteration: 5001 / 10000 [ 50%]  (Sampling)
## Chain 1: Iteration: 6000 / 10000 [ 60%]  (Sampling)
## Chain 1: Iteration: 7000 / 10000 [ 70%]  (Sampling)
## Chain 1: Iteration: 8000 / 10000 [ 80%]  (Sampling)
## Chain 1: Iteration: 9000 / 10000 [ 90%]  (Sampling)
## Chain 1: Iteration: 10000 / 10000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.11 seconds (Warm-up)
## Chain 1:                0.122 seconds (Sampling)
## Chain 1:                0.232 seconds (Total)
## Chain 1:
```

```r
samples <- as.data.frame(nfit)
colnames(samples)[seq_len(ncol(x))] <- colnames(x)
mcmc_intervals(
  samples[, seq_len(ncol(x))],
  pars = colnames(x),
  prob = 0.68,
) + labs(
  title = "Posterior Distributions of Coefficients in Linear Model",
  subtitle = "with medians and 68% intervals"
)
```



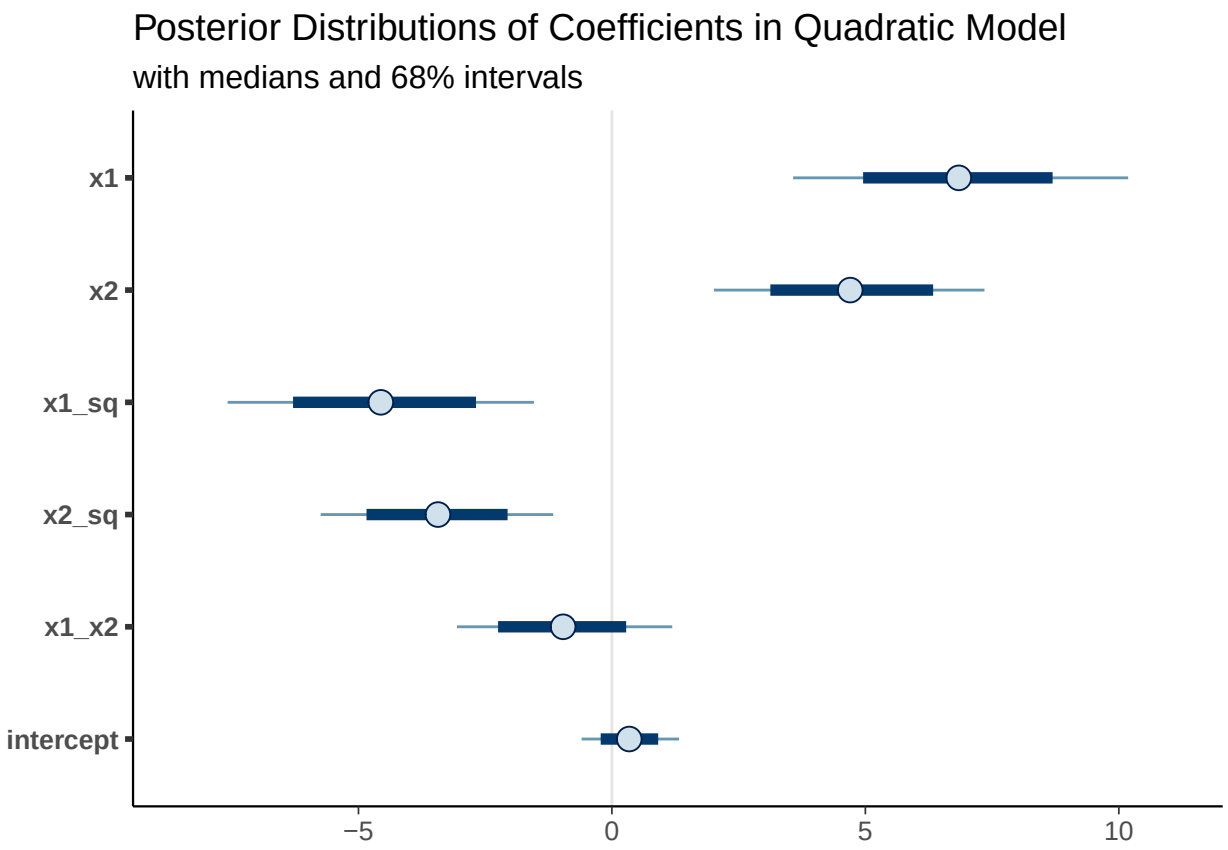
In this first version of the model, we investigate only the terms in a quadratic polynomial. The following visualization presents the posterior distributions of the coefficients with medians and 68% intervals. Based on the result, **the square of concentration and interaction of two molecules have negative impacts on the conversion rate**.

```r
x <- df_1[, cols[1:5]]
x$intercept <- 1
y <- df_1$y
reg_data <- list(n = nrow(x), k = ncol(x), pr_std = 10.0, x = x, y = y)
nfit <- sampling(
  linreg_gaussian_model,
  data = reg_data,
  iter = 10000,
  warmup = 5000,
  chains = 1,
  seed = 42,
```

```
  show_messages = FALSE,
)
```

```
##
## SAMPLING FOR MODEL 'gaussian_regression' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 6e-06 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.06 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 10000 [  0%]  (Warmup)
## Chain 1: Iteration: 1000 / 10000 [ 10%]  (Warmup)
## Chain 1: Iteration: 2000 / 10000 [ 20%]  (Warmup)
## Chain 1: Iteration: 3000 / 10000 [ 30%]  (Warmup)
## Chain 1: Iteration: 4000 / 10000 [ 40%]  (Warmup)
## Chain 1: Iteration: 5000 / 10000 [ 50%]  (Warmup)
## Chain 1: Iteration: 5001 / 10000 [ 50%]  (Sampling)
## Chain 1: Iteration: 6000 / 10000 [ 60%]  (Sampling)
## Chain 1: Iteration: 7000 / 10000 [ 70%]  (Sampling)
## Chain 1: Iteration: 8000 / 10000 [ 80%]  (Sampling)
## Chain 1: Iteration: 9000 / 10000 [ 90%]  (Sampling)
## Chain 1: Iteration: 10000 / 10000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.374 seconds (Warm-up)
## Chain 1:                0.404 seconds (Sampling)
## Chain 1:                0.778 seconds (Total)
## Chain 1:
```

```
samples <- as.data.frame(nfit)
colnames(samples)[seq_len(ncol(x))] <- colnames(x)
mcmc_intervals(
  samples[, seq_len(ncol(x))],
  pars = colnames(x),
  prob = 0.68,
) + labs(
  title = "Posterior Distributions of Coefficients in Quadratic Model",
  subtitle = "with medians and 68% intervals"
)
```



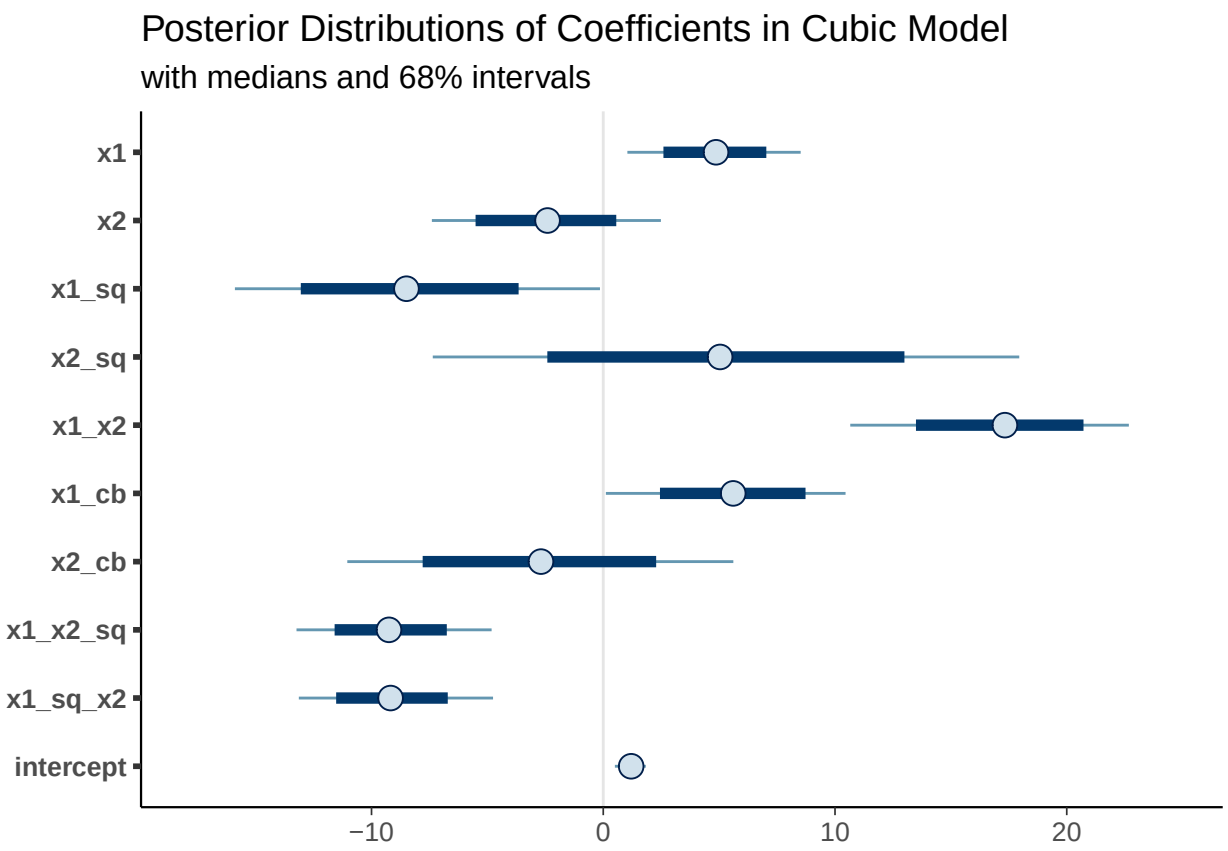
Next, we investigate the terms in a cubic polynomial. The following visualization presents the posterior distributions of the coefficients with medians and 68% intervals. Based on the result: - introducing cubic terms affect the coefficient posteriors of x_2, x_2^2, and x_1x_2; - the uncertainty of coefficients x_1, x_2, x_1x_2, x_1^3, $x_1^2x_2$, and $x_1x_2^2$ are lower than others.

```
x <- df_1[, cols[1:9]]
x$intercept <- 1
y <- df_1$y
reg_data <- list(n = nrow(x), k = ncol(x), pr_std = 10.0, x = x, y = y)
nfit <- sampling(
  linreg_gaussian_model,
  data = reg_data,
  iter = 10000,
  warmup = 5000,
  chains = 1,
  seed = 42,
  show_messages = FALSE,
)
```

```
##
## SAMPLING FOR MODEL 'gaussian_regression' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 6e-06 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.06 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 10000 [  0%]  (Warmup)
```

```
## Chain 1: Iteration: 1000 / 10000 [ 10%]  (Warmup)
## Chain 1: Iteration: 2000 / 10000 [ 20%]  (Warmup)
## Chain 1: Iteration: 3000 / 10000 [ 30%]  (Warmup)
## Chain 1: Iteration: 4000 / 10000 [ 40%]  (Warmup)
## Chain 1: Iteration: 5000 / 10000 [ 50%]  (Warmup)
## Chain 1: Iteration: 5001 / 10000 [ 50%]  (Sampling)
## Chain 1: Iteration: 6000 / 10000 [ 60%]  (Sampling)
## Chain 1: Iteration: 7000 / 10000 [ 70%]  (Sampling)
## Chain 1: Iteration: 8000 / 10000 [ 80%]  (Sampling)
## Chain 1: Iteration: 9000 / 10000 [ 90%]  (Sampling)
## Chain 1: Iteration: 10000 / 10000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 1.95 seconds (Warm-up)
## Chain 1:                2.606 seconds (Sampling)
## Chain 1:                4.556 seconds (Total)
## Chain 1:
```

```r
samples <- as.data.frame(nfit)
colnames(samples)[seq_len(ncol(x))] <- colnames(x)
mcmc_intervals(
  samples[, seq_len(ncol(x))],
  pars = colnames(x),
  prob = 0.68,
) + labs(
  title = "Posterior Distributions of Coefficients in Cubic Model",
  subtitle = "with medians and 68% intervals"
)
```

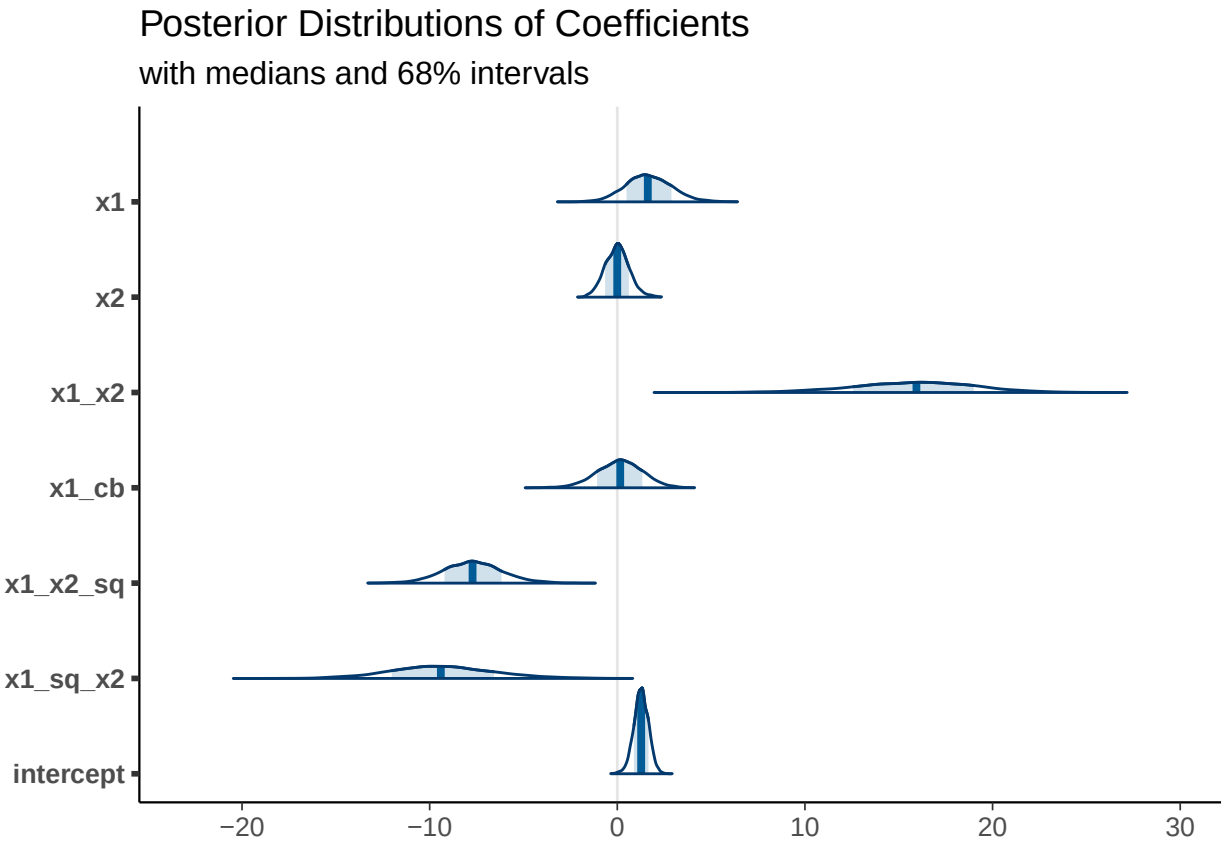


The following investigate if we discard the high uncertainty coefficients and only keep those with lower uncertainty in posterior. The results indicates that: - effects of x_2 and x_1^3 are close to zero; - in turns, the uncertainty in coefficient posteriors of x_1x_2 and $x_1^2x_2$ are significantly higher.

```
x <- df_1[, cols[c(1, 2, 5, 6, 8, 9)]]
x$intercept <- 1
y <- df_1$y
reg_data <- list(n = nrow(x), k = ncol(x), pr_std = 10.0, x = x, y = y)
nfit <- sampling(
  linreg_gaussian_model,
  data = reg_data,
  iter = 10000,
  warmup = 5000,
  chains = 1,
  seed = 42,
  show_messages = FALSE,
)
```

```
##
## SAMPLING FOR MODEL 'gaussian_regression' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 5e-06 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.05 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 10000 [  0%]  (Warmup)
## Chain 1: Iteration: 1000 / 10000 [ 10%]  (Warmup)
## Chain 1: Iteration: 2000 / 10000 [ 20%]  (Warmup)
## Chain 1: Iteration: 3000 / 10000 [ 30%]  (Warmup)
## Chain 1: Iteration: 4000 / 10000 [ 40%]  (Warmup)
## Chain 1: Iteration: 5000 / 10000 [ 50%]  (Warmup)
## Chain 1: Iteration: 5001 / 10000 [ 50%]  (Sampling)
## Chain 1: Iteration: 6000 / 10000 [ 60%]  (Sampling)
## Chain 1: Iteration: 7000 / 10000 [ 70%]  (Sampling)
## Chain 1: Iteration: 8000 / 10000 [ 80%]  (Sampling)
## Chain 1: Iteration: 9000 / 10000 [ 90%]  (Sampling)
## Chain 1: Iteration: 10000 / 10000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.461 seconds (Warm-up)
## Chain 1:                0.561 seconds (Sampling)
## Chain 1:                1.022 seconds (Total)
## Chain 1:
```

```
samples <- as.data.frame(nfit)
colnames(samples)[seq_len(ncol(x))] <- colnames(x)
mcmc_areas(
  samples[, seq_len(ncol(x))],
  pars = colnames(x),
  prob = 0.68,
) + labs(
  title = "Posterior Distributions of Coefficients",
  subtitle = "with medians and 68% intervals"
)
```



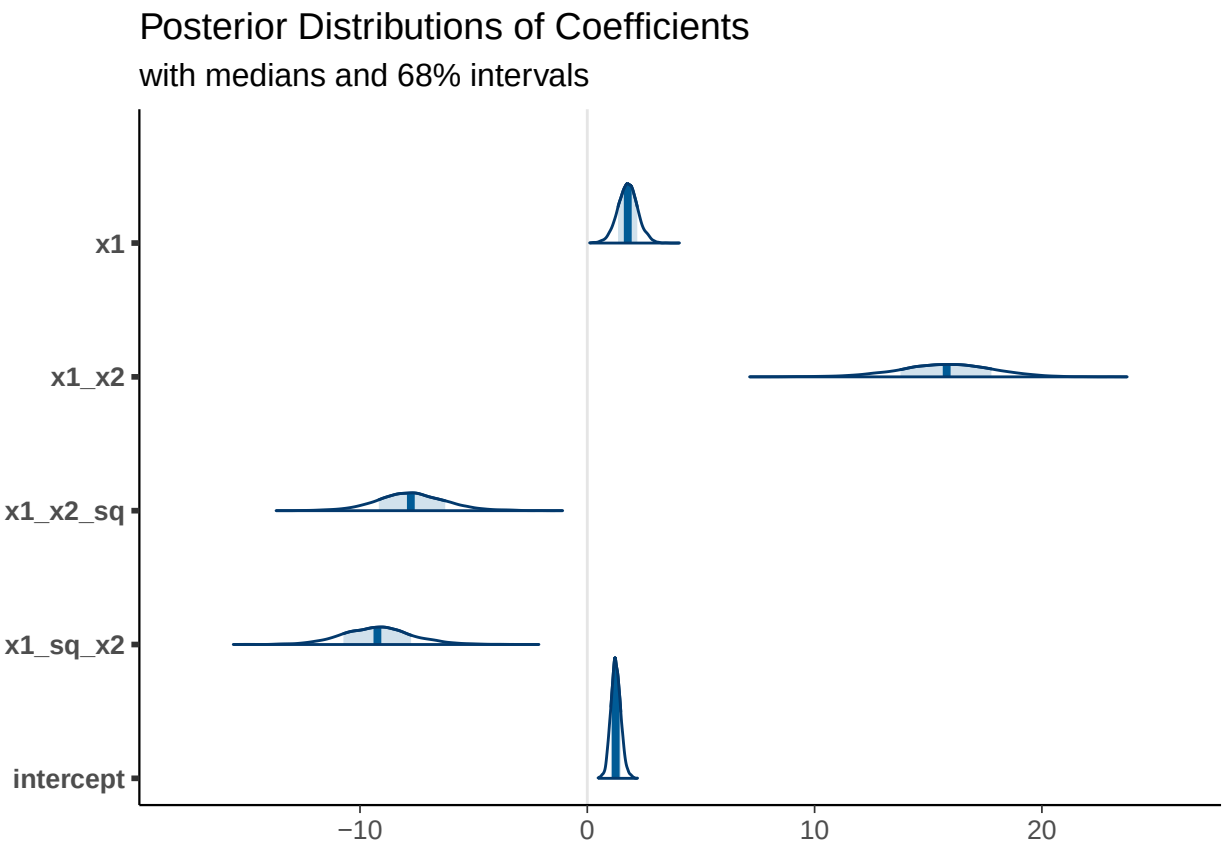
Finally, we keep only the terms with low uncertainties and plot the following visualization. From the result, we can see that despite there are relatively high uncertainties in coefficient posteriors, all distributions are significantly deviated from zeros. In this fitted model: - concentration of the first molecule and the interaction of two molecules positively affect the conversion rate; - cubic interaction of the concentrations $x_1^2 x_2$ and $x_1 x_2^2$ negatively impact the conversion rate.

```
x <- df_1[, cols[c(1, 5, 8, 9)]]
x$intercept <- 1
y <- df_1$y
reg_data <- list(n = nrow(x), k = ncol(x), pr_std = 10.0, x = x, y = y)
nfit <- sampling(
  linreg_gaussian_model,
  data = reg_data,
  iter = 10000,
  warmup = 5000,
  chains = 1,
  seed = 42,
  show_messages = FALSE,
)
```

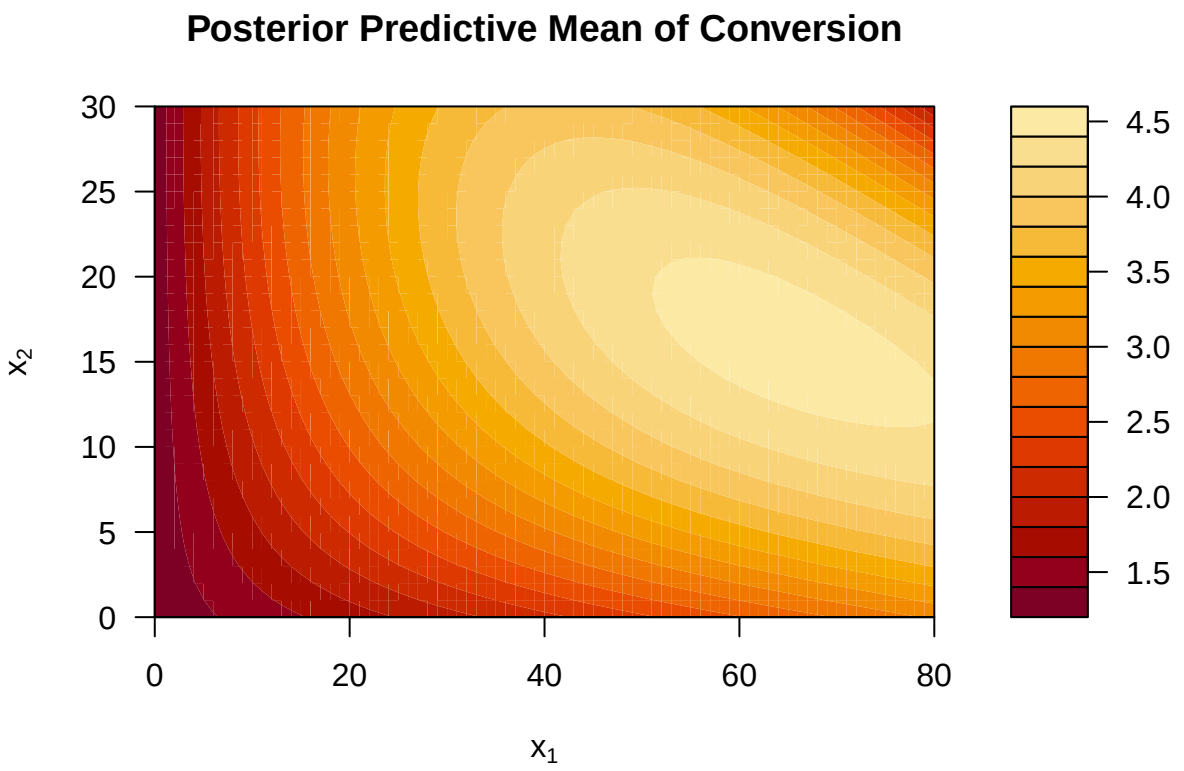
```
##
## SAMPLING FOR MODEL 'gaussian_regression' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 5e-06 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.05 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
```

```
## Chain 1: Iteration:    1 / 10000 [  0%]  (Warmup)
## Chain 1: Iteration: 1000 / 10000 [ 10%]  (Warmup)
## Chain 1: Iteration: 2000 / 10000 [ 20%]  (Warmup)
## Chain 1: Iteration: 3000 / 10000 [ 30%]  (Warmup)
## Chain 1: Iteration: 4000 / 10000 [ 40%]  (Warmup)
## Chain 1: Iteration: 5000 / 10000 [ 50%]  (Warmup)
## Chain 1: Iteration: 5001 / 10000 [ 50%]  (Sampling)
## Chain 1: Iteration: 6000 / 10000 [ 60%]  (Sampling)
## Chain 1: Iteration: 7000 / 10000 [ 70%]  (Sampling)
## Chain 1: Iteration: 8000 / 10000 [ 80%]  (Sampling)
## Chain 1: Iteration: 9000 / 10000 [ 90%]  (Sampling)
## Chain 1: Iteration: 10000 / 10000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.306 seconds (Warm-up)
## Chain 1:                0.387 seconds (Sampling)
## Chain 1:                0.693 seconds (Total)
## Chain 1:
```

```r
samples <- as.data.frame(nfit)
colnames(samples)[seq_len(ncol(x))] <- colnames(x)
mcmc_areas(
  samples[, seq_len(ncol(x))],
  pars = colnames(x),
  prob = 0.68,
) + labs(
  title = "Posterior Distributions of Coefficients",
  subtitle = "with medians and 68% intervals"
)
```



```r
log_avg_intensity <- function(x1, x2, weights) {
  intensity <- weights[, 1] * x1 +
    weights[, 2] * x1 * x2 +
    weights[, 3] * x1 * x2^2 +
    weights[, 4] * x1^2 * x2 +
    weights[, 5]
  out <- mean(intensity)
  return(out)
}

x1 <- seq(0, 80, 1.0)
x2 <- seq(0, 30, 1.0)
grid <- expand.grid(x1 = x1, x2 = x2)
weight_samples <- samples[, 1:10]
z <- mapply(
  function(x1, x2) log_avg_intensity(x1 / 80, x2 / 30, weight_samples),
  grid$x1,
  grid$x2
)
z <- matrix(z, nrow = length(x1), ncol = length(x2))
col <- hcl.colors(20, "YlOrRd")
filled.contour(
  x1,
  x2,
  z,
  col = col,
  xlab = expression("x"[1]),
  ylab = expression("x"[2]),
  main = "Posterior Predictive Mean of Conversion"
)
```



The contour indicates that for $72 \leq x_1 \leq 80$ and $12 \leq x_2 \leq 24$, the conversion rate can be maximized. Therefore, we uniformly sample 48 pairs of concentrations from this region for the second experiment.

```
set.seed(42)
x1 <- runif(48, 0.9, 1) * 80
x2 <- runif(48, 0.4, 0.8) * 30
samples <- data.frame(x1 = x1, x2 = x2)
write.table(
  samples,
  "data/hw2_second_guess.csv",
  row.names = FALSE,
  col.names = FALSE,
  sep = " "
)
```

We ran the experiments on the second experiment and get the following observations. As we expected, the range and mean of observed conversion rate indicate that we have observed a higher conversion rate than the first experiment.

```
# Read data
if (file.exists("data/hw2_second_result.csv")) {
  df_2 <- read.csv("data/hw2_second_result.csv", header = FALSE, sep = " ")
} else {
  stop("FileNotFound: data not found at 'data/hw2_second_result.csv'.")
}
colnames(df_2) <- c("x1", "x2", "y")
summary(df_2)
```

```
##        x1              x2              y
##  Min.   :72.03   Min.   :12.00   Min.   :3.569
##  1st Qu.:75.12   1st Qu.:14.47   1st Qu.:4.059
```

```
##  Median :77.13   Median :17.27   Median :4.224
##  Mean   :76.73   Mean   :17.47   Mean   :4.274
##  3rd Qu.:79.13   3rd Qu.:20.40   3rd Qu.:4.515
##  Max.   :79.91   Max.   :23.79   Max.   :5.139
```

Next, we combine the results from the two experiments, apply the same normalization on the concentration levels, and transform to create attributes accordingly:

```
# Preprocess the features through normalization
df_2$x1 <- df_2$x1 / 80
df_2$x2 <- df_2$x2 / 30

df <- rbind(df_1[, colnames(df_2)], df_2)
summary(df)
```

```
##        x1                x2               y
##  Min.   :0.001081   Min.   :0.0000   Min.   :0.645
##  1st Qu.:0.810518   1st Qu.:0.4740   1st Qu.:3.552
##  Median :0.944137   Median :0.5360   Median :4.098
##  Mean   :0.808574   Mean   :0.5573   Mean   :3.811
##  3rd Qu.:0.982846   3rd Qu.:0.6951   3rd Qu.:4.421
##  Max.   :0.998889   Max.   :1.0219   Max.   :5.139
```

```
# Preprocess the features through polynomial expansion
df$x1_sq <- df$x1^2
df$x2_sq <- df$x2^2
df$x1_x2 <- df$x1 * df$x2
df$x1_cb <- df$x1^3
df$x2_cb <- df$x2^3
df$x1_x2_sq <- df$x1 * df$x2^2
df$x1_sq_x2 <- df$x1^2 * df$x2
```

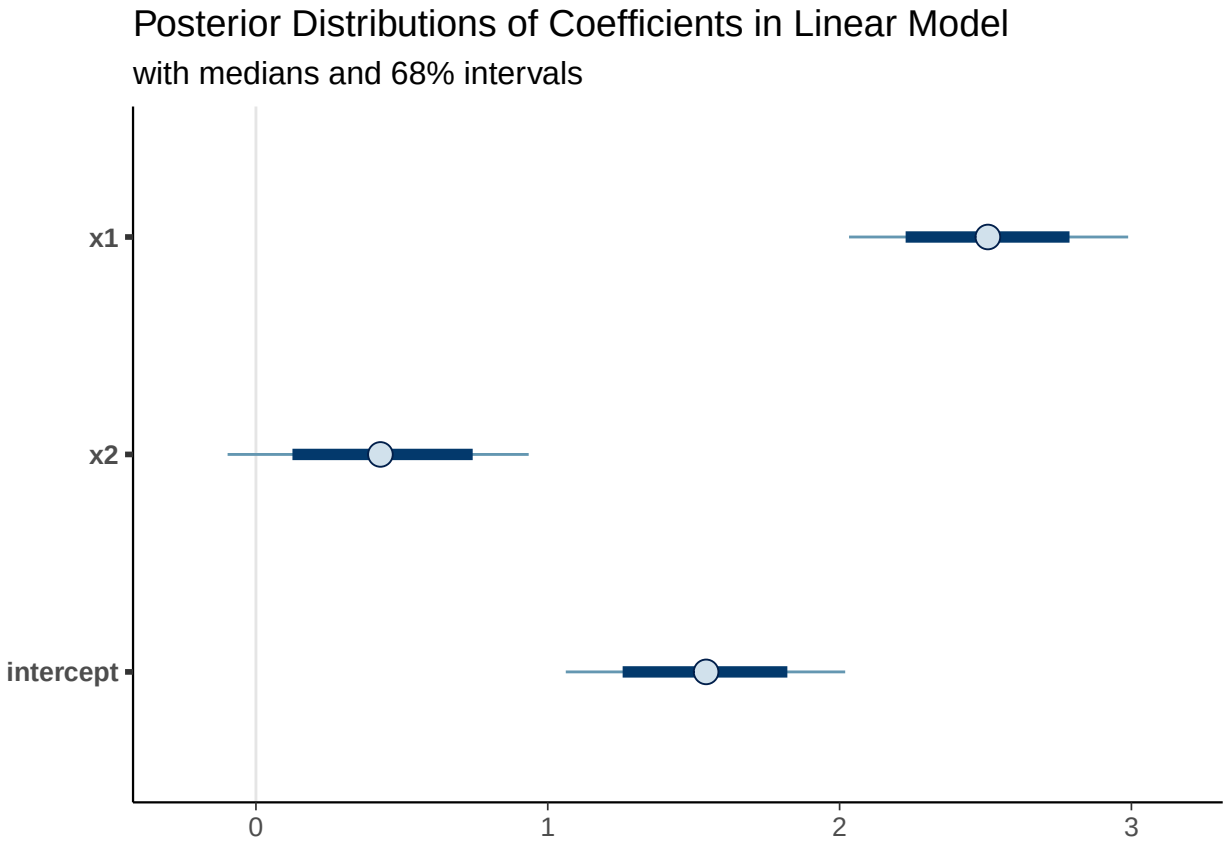
For the rigor of science, I run all the previous procedures with on the merged dataset to determine which model best describe the correlation. First, the following shows the results from a linear model. We can see that this time the model raises a significantly different result compared to the one from the first experiment. This observation indicates: - the linear model are sensitive to the data observation, since the merged dataset is skewed majorly by the second experiment; - the linear model may not be sufficient to capture the interaction between two molecules.

```
x <- df[, cols[1:2]]
x$intercept <- 1
y <- df$y
reg_data <- list(n = nrow(x), k = ncol(x), pr_std = 10.0, x = x, y = y)
nfit <- sampling(
  linreg_gaussian_model,
  data = reg_data,
  iter = 10000,
  warmup = 5000,
  chains = 1,
  seed = 42,
  show_messages = FALSE,
)
```

```
##
## SAMPLING FOR MODEL 'gaussian_regression' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 7e-06 seconds
```

```
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.07 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 10000 [  0%]  (Warmup)
## Chain 1: Iteration: 1000 / 10000 [ 10%]  (Warmup)
## Chain 1: Iteration: 2000 / 10000 [ 20%]  (Warmup)
## Chain 1: Iteration: 3000 / 10000 [ 30%]  (Warmup)
## Chain 1: Iteration: 4000 / 10000 [ 40%]  (Warmup)
## Chain 1: Iteration: 5000 / 10000 [ 50%]  (Warmup)
## Chain 1: Iteration: 5001 / 10000 [ 50%]  (Sampling)
## Chain 1: Iteration: 6000 / 10000 [ 60%]  (Sampling)
## Chain 1: Iteration: 7000 / 10000 [ 70%]  (Sampling)
## Chain 1: Iteration: 8000 / 10000 [ 80%]  (Sampling)
## Chain 1: Iteration: 9000 / 10000 [ 90%]  (Sampling)
## Chain 1: Iteration: 10000 / 10000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.195 seconds (Warm-up)
## Chain 1:                0.227 seconds (Sampling)
## Chain 1:                0.422 seconds (Total)
## Chain 1:
```

```r
samples <- as.data.frame(nfit)
colnames(samples)[seq_len(ncol(x))] <- colnames(x)
mcmc_intervals(
  samples[, seq_len(ncol(x))],
  pars = colnames(x),
  prob = 0.68,
) + labs(
  title = "Posterior Distributions of Coefficients in Linear Model",
  subtitle = "with medians and 68% intervals"
)
```



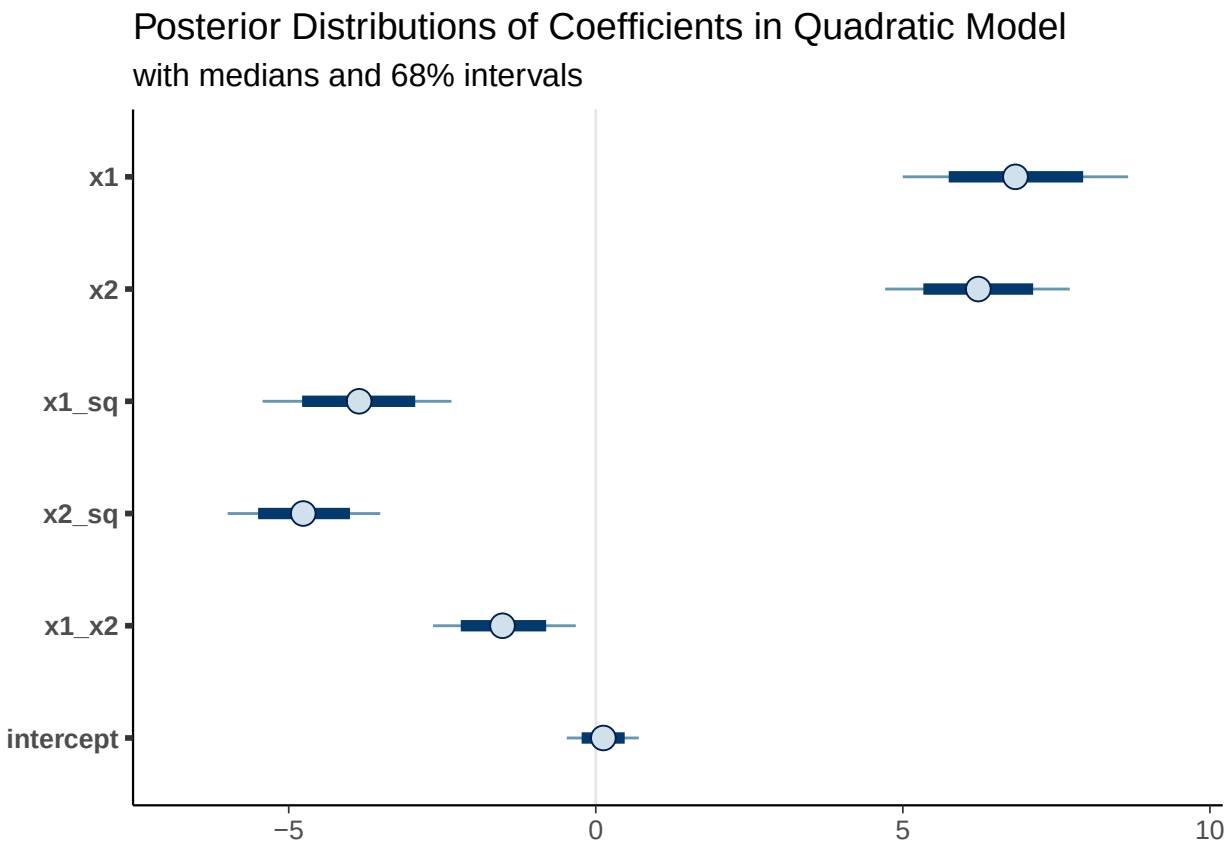
Then, I run the quadratic model and get the following result: - compared to the previous quadratic model, all posterior distributions have a higher uncertainty; - different from the previous one, this model assert that interaction of the two molecules might positively affect the conversion rate.

```
x <- df[, cols[1:5]]
x$intercept <- 1
y <- df$y
reg_data <- list(n = nrow(x), k = ncol(x), pr_std = 10.0, x = x, y = y)
nfit <- sampling(
  linreg_gaussian_model,
  data = reg_data,
  iter = 10000,
  warmup = 5000,
  chains = 1,
  seed = 42,
  show_messages = FALSE,
)
```

```
##
## SAMPLING FOR MODEL 'gaussian_regression' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 6e-06 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.06 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 10000 [  0%]  (Warmup)
## Chain 1: Iteration: 1000 / 10000 [ 10%]  (Warmup)
```

```
## Chain 1: Iteration: 2000 / 10000 [ 20%]  (Warmup)
## Chain 1: Iteration: 3000 / 10000 [ 30%]  (Warmup)
## Chain 1: Iteration: 4000 / 10000 [ 40%]  (Warmup)
## Chain 1: Iteration: 5000 / 10000 [ 50%]  (Warmup)
## Chain 1: Iteration: 5001 / 10000 [ 50%]  (Sampling)
## Chain 1: Iteration: 6000 / 10000 [ 60%]  (Sampling)
## Chain 1: Iteration: 7000 / 10000 [ 70%]  (Sampling)
## Chain 1: Iteration: 8000 / 10000 [ 80%]  (Sampling)
## Chain 1: Iteration: 9000 / 10000 [ 90%]  (Sampling)
## Chain 1: Iteration: 10000 / 10000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.785 seconds (Warm-up)
## Chain 1:                1.104 seconds (Sampling)
## Chain 1:                1.889 seconds (Total)
## Chain 1:
```

```r
samples <- as.data.frame(nfit)
colnames(samples)[seq_len(ncol(x))] <- colnames(x)
mcmc_intervals(
  samples[, seq_len(ncol(x))],
  pars = colnames(x),
  prob = 0.68,
) + labs(
  title = "Posterior Distributions of Coefficients in Quadratic Model",
  subtitle = "with medians and 68% intervals"
)
```



Next, I run the cubic model on the new dataset and get the following result: - compared to the quadratic

model, some coefficients (*e.g.*, x_1, x_2, x_1x_2) have lower posterior uncertainties; - the new cubic model have similar coefficient posterior to the previous one fitted on solely the first experiment; - based on the posterior uncertainty, it is reasonable to only keep x_1, x_2, $x_1^2x_2$, and $x_1x_2^2$ terms.

```
x <- df[, cols[1:9]]
x$intercept <- 1
y <- df$y
reg_data <- list(n = nrow(x), k = ncol(x), pr_std = 10.0, x = x, y = y)
nfit <- sampling(
  linreg_gaussian_model,
  data = reg_data,
  iter = 10000,
  warmup = 5000,
  chains = 1,
  seed = 42,
  show_messages = FALSE,
)
```

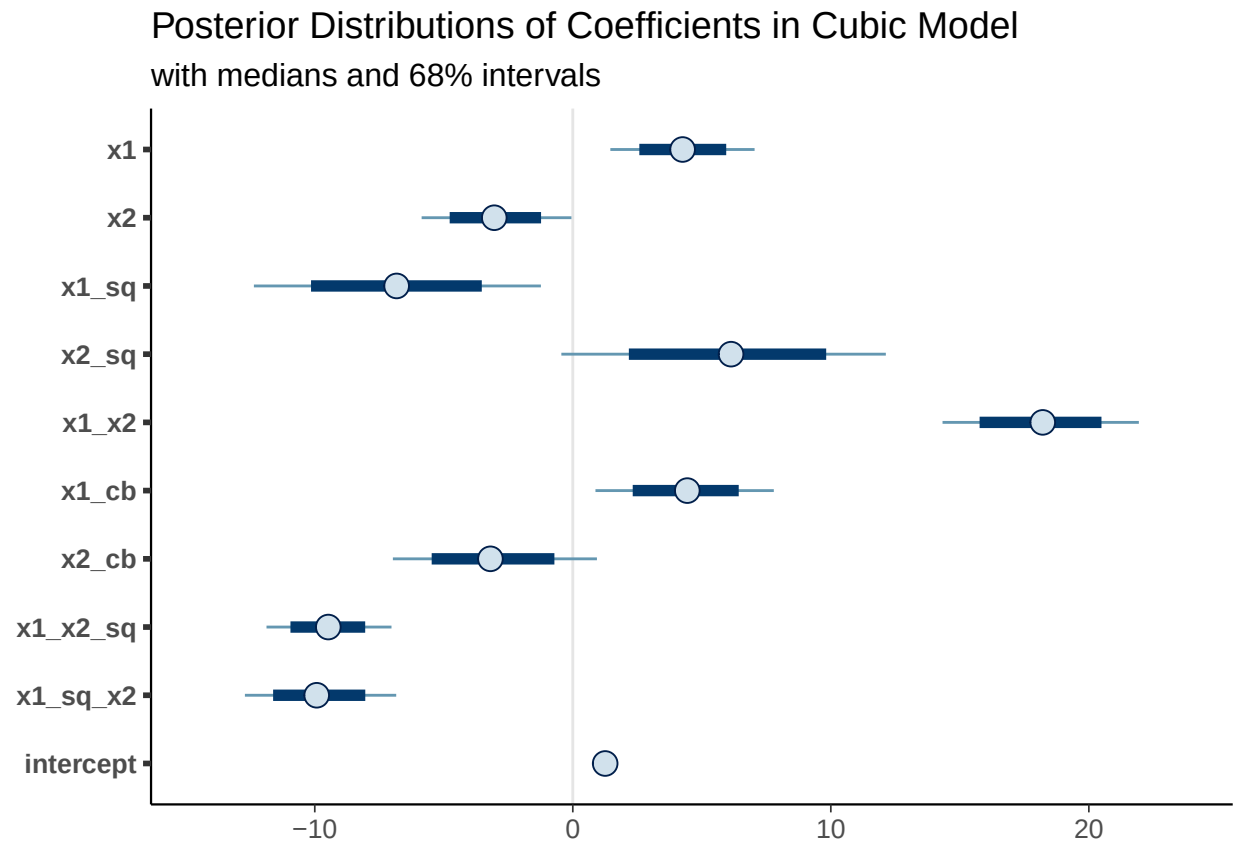
```
##
## SAMPLING FOR MODEL 'gaussian_regression' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 1.9e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.19 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 10000 [  0%]  (Warmup)
## Chain 1: Iteration: 1000 / 10000 [ 10%]  (Warmup)
## Chain 1: Iteration: 2000 / 10000 [ 20%]  (Warmup)
## Chain 1: Iteration: 3000 / 10000 [ 30%]  (Warmup)
## Chain 1: Iteration: 4000 / 10000 [ 40%]  (Warmup)
## Chain 1: Iteration: 5000 / 10000 [ 50%]  (Warmup)
## Chain 1: Iteration: 5001 / 10000 [ 50%]  (Sampling)
## Chain 1: Iteration: 6000 / 10000 [ 60%]  (Sampling)
## Chain 1: Iteration: 7000 / 10000 [ 70%]  (Sampling)
## Chain 1: Iteration: 8000 / 10000 [ 80%]  (Sampling)
## Chain 1: Iteration: 9000 / 10000 [ 90%]  (Sampling)
## Chain 1: Iteration: 10000 / 10000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 3.777 seconds (Warm-up)
## Chain 1:                5.504 seconds (Sampling)
## Chain 1:                9.281 seconds (Total)
## Chain 1:
```

```
## Warning: There were 27 transitions after warmup that exceeded the maximum treedepth. Increase max_tr
## https://mc-stan.org/misc/warnings.html#maximum-treedepth-exceeded
```

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

```
samples <- as.data.frame(nfit)
colnames(samples)[seq_len(ncol(x))] <- colnames(x)
mcmc_intervals(
  samples[, seq_len(ncol(x))],
  pars = colnames(x),
  prob = 0.68,
) + labs(
```

```
  title = "Posterior Distributions of Coefficients in Cubic Model",
  subtitle = "with medians and 68% intervals"
)
```



Finally, we fit the model using data collected from the two experiments,

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2^2 + \beta_4 x_1^2 x_2 + \varepsilon, \quad \boldsymbol{\beta} \sim \mathcal{N}(0, 10^2 \mathbf{I}), \quad \varepsilon \sim \text{Half-Cauchy}(0, 1)$$

and get the following results. The optimal combination of two molecules given by the model is $x_1 = 65$, $x_2 = 16$.

```
x <- df_1[, cols[c(1, 5, 8, 9)]]
x$intercept <- 1
y <- df_1$y
reg_data <- list(n = nrow(x), k = ncol(x), pr_std = 10.0, x = x, y = y)
nfit <- sampling(
  linreg_gaussian_model,
  data = reg_data,
  iter = 10000,
  warmup = 5000,
  chains = 1,
  seed = 42,
  show_messages = FALSE,
)
```

```
##
## SAMPLING FOR MODEL 'gaussian_regression' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 6e-06 seconds
```

```
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.06 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 10000 [  0%]  (Warmup)
## Chain 1: Iteration: 1000 / 10000 [ 10%]  (Warmup)
## Chain 1: Iteration: 2000 / 10000 [ 20%]  (Warmup)
## Chain 1: Iteration: 3000 / 10000 [ 30%]  (Warmup)
## Chain 1: Iteration: 4000 / 10000 [ 40%]  (Warmup)
## Chain 1: Iteration: 5000 / 10000 [ 50%]  (Warmup)
## Chain 1: Iteration: 5001 / 10000 [ 50%]  (Sampling)
## Chain 1: Iteration: 6000 / 10000 [ 60%]  (Sampling)
## Chain 1: Iteration: 7000 / 10000 [ 70%]  (Sampling)
## Chain 1: Iteration: 8000 / 10000 [ 80%]  (Sampling)
## Chain 1: Iteration: 9000 / 10000 [ 90%]  (Sampling)
## Chain 1: Iteration: 10000 / 10000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.326 seconds (Warm-up)
## Chain 1:                0.386 seconds (Sampling)
## Chain 1:                0.712 seconds (Total)
## Chain 1:
```

```r
samples <- as.data.frame(nfit)
colnames(samples)[seq_len(ncol(x))] <- colnames(x)

log_avg_intensity <- function(x1, x2, weights) {
  intensity <- weights[, 1] * x1 +
    weights[, 2] * x1 * x2 +
    weights[, 3] * x1 * x2^2 +
    weights[, 4] * x1^2 * x2 +
    weights[, 5]
  out <- mean(intensity)
  return(out)
}

x1 <- seq(0, 80, 0.5)
x2 <- seq(0, 30, 0.5)
grid <- expand.grid(x1 = x1, x2 = x2)
weight_samples <- samples[, 1:10]
z <- mapply(
  function(x1, x2) log_avg_intensity(x1 / 80, x2 / 30, weight_samples),
  grid$x1,
  grid$x2
)
z <- matrix(z, nrow = length(x1), ncol = length(x2))


# Determine the best combination of concentrations
index <- which(z == max(z), arr.ind = TRUE)
x1_opt <- x1[index[1]]
x2_opt <- x2[index[2]]

col <- hcl.colors(20, "YlOrRd")
filled.contour(
```

```
  x1,
  x2,
  z,
  col = col,
  xlab = expression("x"[1]),
  ylab = expression("x"[2]),
  plot.title = title(
    main = "Posterior Predictive Mean of Conversion",
    sub = "fitted on two experiments"
  ),
  plot.axes = {
    axis(1)
    axis(2)
    abline(v = x1_opt, lty = 2, col = "black")
    abline(h = x2_opt, lty = 2, col = "black")
    points(x1_opt, x2_opt, pch = 16, col = "red", cex = 1.5)
    text(
      x1_opt + 1,
      x2_opt + 1,
      labels = sprintf("(%g, %g)", round(x1_opt, 2), round(x2_opt, 2)),
      pos = 4, # Position to the right of the point
      col = "black",
      cex = 0.8 # Adjust text size as needed
    )
  }
)
```