Hierarchical models in Bayesian

Wei Deng

Sep.16, 2017

Theory

Consider a collection of counts $y = (y_1, \ldots, y_n)$. We will model the entries in y as independent Poisson random variables with distinct expectations. Specifically, we assume that $y_i|\lambda \sim Poisson(\lambda_i)$ independently, where $\lambda = (\lambda_1, \ldots, \lambda_n)$. In addition, we assume that $\lambda_i \overset{\text{i.i.d.}}{\sim} \frac{1}{\beta} Gamma(\alpha)$ a priori, where α is the shape and β is the rate.

1. Write the expression for the probability mass function of y given (α, β), i.e., $p(y|\alpha, \beta)$.

$$
\begin{aligned}
p(y|\alpha,\beta) &= \prod_{i=1}^{n} p(y_i|\alpha,\beta) \\
&= \prod_{i=1}^{n} \int_0^\infty p(y_i|\lambda_i) p(\lambda_i|\alpha,\beta)\, \mathrm{d}\lambda_i \\
&= \prod_{i=1}^{n} \int_0^\infty \frac{\lambda_i^{y_i} e^{-\lambda_i}}{y_i!} \frac{\beta^\alpha \lambda_i^{\alpha-1} e^{-\beta\lambda_i}}{\Gamma(\alpha)} \mathrm{d}\lambda_i \\
&= \prod_{i=1}^{n} \frac{\beta^\alpha}{y_i! \cdot \Gamma(\alpha)} \int_0^\infty \lambda_i^{y_i+\alpha-1} e^{-\lambda_i(\beta+1)} \mathrm{d}\lambda_i \\
&= \prod_{i=1}^{n} \frac{\beta^\alpha \cdot \Gamma(y_i+\alpha)}{y_i!(\beta+1)^{y_i+\alpha}\Gamma(\alpha)} \\
&= \prod_{i=1}^{n} \frac{\beta^\alpha \cdot \Gamma(y_i+\alpha-1)(y_i+\alpha-1)}{y_i!(\beta+1)^{y_i+\alpha}\Gamma(\alpha)} \\
&= \prod_{i=1}^{n} \frac{\beta^\alpha \cdot \Gamma(y_i+\alpha-2)(y_i+\alpha-1)(y_i+\alpha-2)}{y_i!(\beta+1)^{y_i+\alpha}\Gamma(\alpha)} \\
&= \ldots \\
&= \prod_{i=1}^{n} \frac{\beta^\alpha \cdot \Gamma(\alpha) \prod_{j=1}^{y_i}(y_i+\alpha-j)}{y_i!(\beta+1)^{y_i+\alpha}\Gamma(\alpha)} \\
&= \prod_{i=1}^{n} \frac{\beta^\alpha \cdot \prod_{j=1}^{y_i}(y_i+\alpha-j)}{y_i!(\beta+1)^{y_i+\alpha}} \\
&= \prod_{i=1}^{n} \left(\prod_{j=1}^{y_i} \frac{y_i+\alpha-j}{y_i+1-j} \frac{\beta^\alpha}{(\beta+1)^{y_i+\alpha}} \right)
\end{aligned}
$$

2. Derive the conditional posterior distribution $p(\lambda|y, \alpha, \beta)$ in closed-form.

$$
\begin{aligned}
p(\lambda|y,\alpha,\beta) &= \frac{p(\lambda,y|\alpha,\beta)}{p(y|\alpha,\beta)} \\
&= \frac{\prod_{i=1}^{n} p(\lambda_i,y_i|\alpha,\beta)}{p(y|\alpha,\beta)} \\
&= \prod_{i=1}^{n}\left(\frac{\lambda_i^{y_i}e^{-\lambda_i}}{y_i!}\,\frac{\beta^{\alpha}\lambda_i^{\alpha-1}e^{-\beta\lambda_i}}{\Gamma(\alpha)}\right)\Big/\left(\frac{\beta^{\alpha}\cdot\Gamma(y_i+\alpha)}{y_i!\cdot\Gamma(\alpha)(\beta+1)^{y_i+\alpha}}\right) \\
&= e^{-(\beta+1)\sum_{i=1}^{n}\lambda_i}\prod_{i=1}^{n}\frac{\lambda_i^{y_i+\alpha-1}(\beta+1)^{y_i+\alpha}}{\Gamma(y_i+\alpha)}
\end{aligned}
$$

Computation

The file $AirPassengers.csv$ contains monthly totals of airline passengers from January 1949 to December 1960 (in the hundreds of thousands). Consider the last column in this file. We will model this collection of counts as in the above Theory problem.

```
set.seed(26)
setwd("C:/Users/Wei/Desktop/STAT 695/HW2")
```

1. Construct a contour plot of the log-likelihood $log\{L(\alpha,\beta|y)\}$.

```
computation_data = read.csv(file="AirPassengers.csv", header=TRUE)

X = computation_data$X
Y = computation_data$AirPassengers
n = length(Y)
```

When y is a large floating number, e.g. $y > 100$, we can safely use $\lfloor y \rfloor !$ to approximate $\Gamma(y)$. Therefore, in computing $\frac{\Gamma(y_i+\alpha)}{y_i!\Gamma(y)}$ in the likelihood function, instead of computing $\Gamma(y+\alpha)$ using $(y+\alpha)!$ and $\Gamma(y)$ using $y!$ separately, we compute $\prod_{i=1}^{y}\frac{i+\alpha}{i}$.

To compute the likelihood function, we first compute $\prod_{i=1}^{n}(\prod_{j=1}^{y_i}\frac{y_i+\alpha-j}{y_i+1-j}$ for each y_i to avoid numerical overflow.

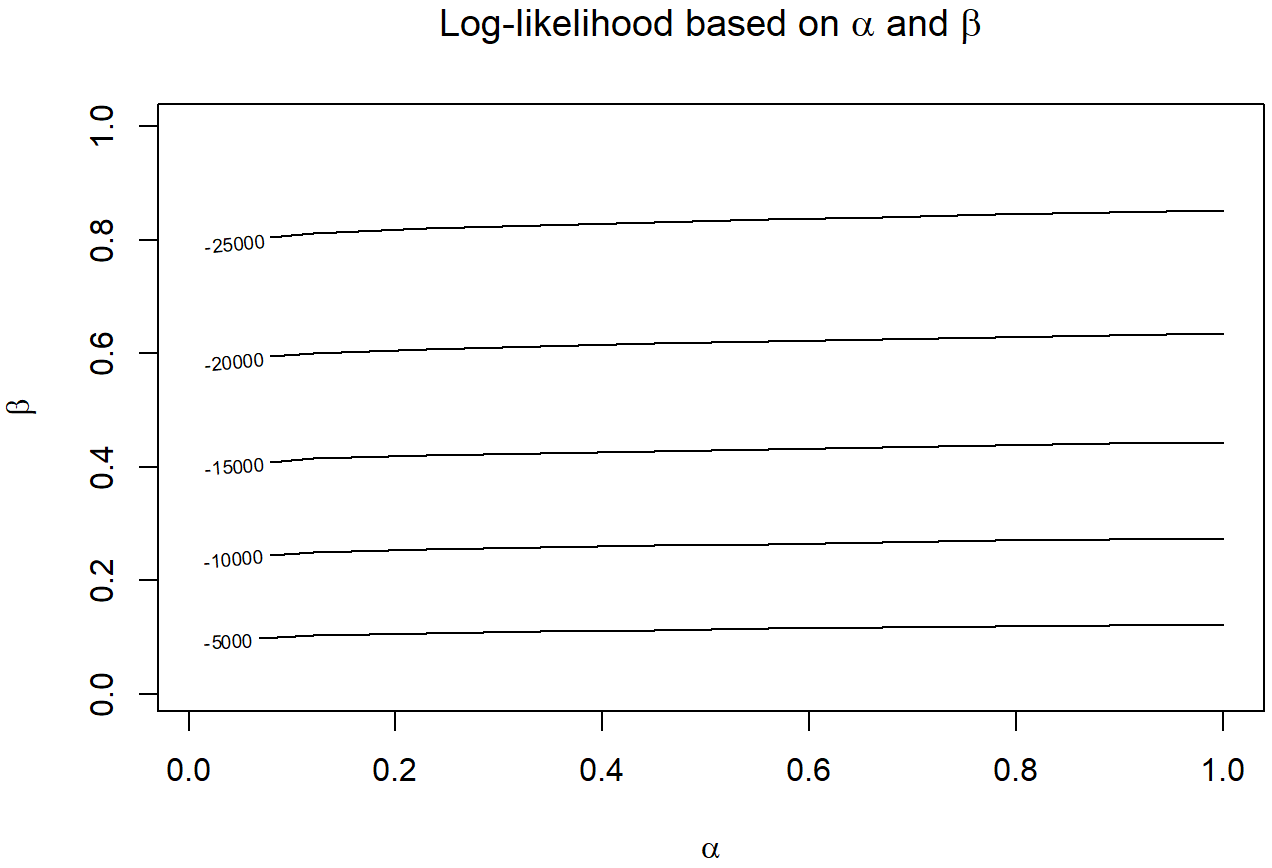
```
recursion = function(y, alpha) {
    if (y > 0) return((y + alpha - 1) / y * recursion(y - 1, alpha))
    else return(1)
}
```

The log-likelihood looks like the following:

```
log_likelihood = function(alpha, beta) {
  n * (alpha * log(beta)) - (sum(Y) + n * alpha) * log(beta + 1) + sum(log(mapply(recursion, Y, alpha)))
}
```

Basis in terms of (α,β) and $(\alpha, log(\beta))$

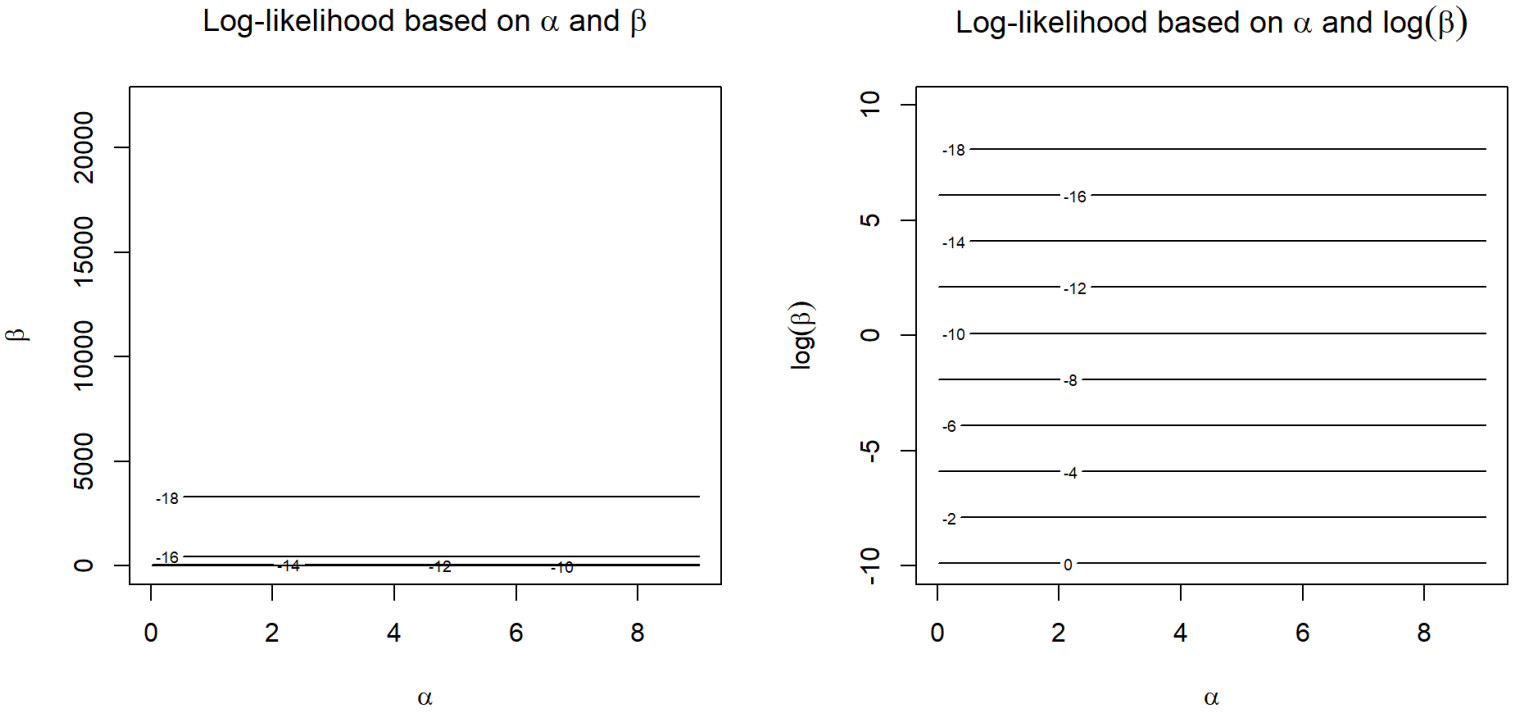
```
nums = 10
alpha_grid = seq(0.01, 1, length.out=nums)
beta_grid = seq(0.01, 1, length.out=nums)

grids = expand.grid(alpha_grid, beta_grid)
val_log_likelihood = matrix(mapply(log_likelihood, grids$Var1, grids$Var2), nrow=nums)

contour(alpha_grid, beta_grid, val_log_likelihood,
        xlab=bquote(paste(alpha)),
        ylab=bquote(paste(beta)),
        main=bquote(paste("Log-likelihood based on "*alpha*" and "*beta)))
```



2. Suggest a hyperprior on (α, β). Construct a contour plot of its log-density.

Since not too much prior information is given, we use flat prior for α and $log(\beta)$ with finite support. Here $\beta \sim LogUniformDistribution, p(\beta) = \frac{1}{x(log(b)-log(a))}$ for $a <= \beta <= b$

```
library(KScorrect)

LIMIT = 10
alpha_grid = seq(0.01, 10, 1)
log_beta_grid = seq(-LIMIT, LIMIT, 0.1)
grids = expand.grid(alpha_grid, beta_grid)
val_log_likelihood = matrix(mapply(log_likelihood, grids$Var1, grids$Var2), nrow=nums)

log_prior = matrix(NA, nrow=length(alpha_grid), ncol=length(log_beta_grid))

for (i in 1: length(alpha_grid)) {
  for (j in 1: length(log_beta_grid)) {
    log_prior[i,j] = log(1/1000) + log(dlunif(exp(log_beta_grid[j]), exp(-LIMIT), exp(LIMIT), base =
exp(1)))
  }
}

par(mfrow=c(1,2))
contour(alpha_grid, exp(log_beta_grid), log_prior,
        xlab=bquote(paste(alpha)),
        ylab=bquote(paste(beta)),
        main=bquote(paste("Log-likelihood based on "*alpha*" and "*beta)))

contour(alpha_grid, log_beta_grid, log_prior,
        xlab=bquote(paste(alpha)),
        ylab=bquote(paste(log(beta))),
        main=bquote(paste("Log-likelihood based on "*alpha*" and "*log(beta))))
```



3. Create a visualization of the marginal posterior density $p(\alpha, \beta|y)$. Argue, using either computation or theory, that your choice of hyperprior results in a proper marginal posterior.

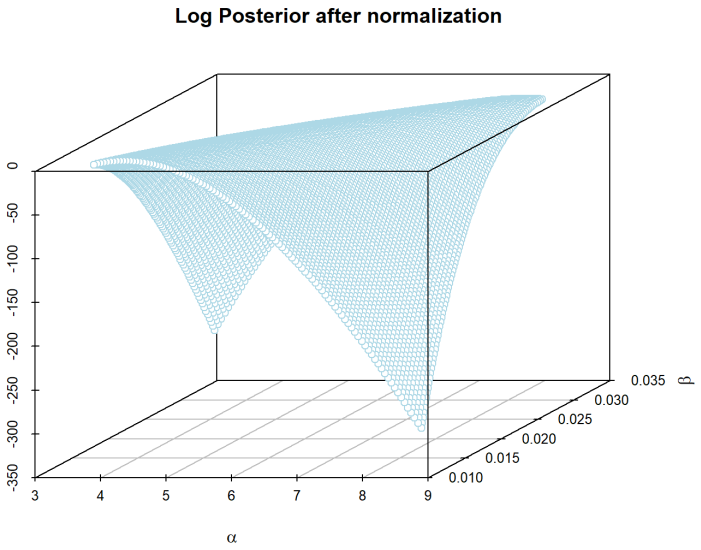
$$
\begin{aligned}
p(\alpha,\beta|y) &\propto p(y|\alpha,\beta)p(\alpha)p(\beta) \\
&\propto p(y|\alpha,\beta)p(\beta) \\
&\propto \prod_{i=1}^{n} \frac{\beta^{\alpha}\cdot\Gamma(y_i+\alpha)}{y_i!\cdot\Gamma(\alpha)(\beta+1)^{y_i+\alpha}}p(\beta) \\
&\propto (\frac{\beta}{\beta+1})^{n\alpha}\prod_{i=1}^{n}\frac{\Gamma(y_i+\alpha)}{\Gamma(\alpha)}p(\beta)/(\beta+1)^{\sum_{i=1}^{n}y_i}
\end{aligned}
$$

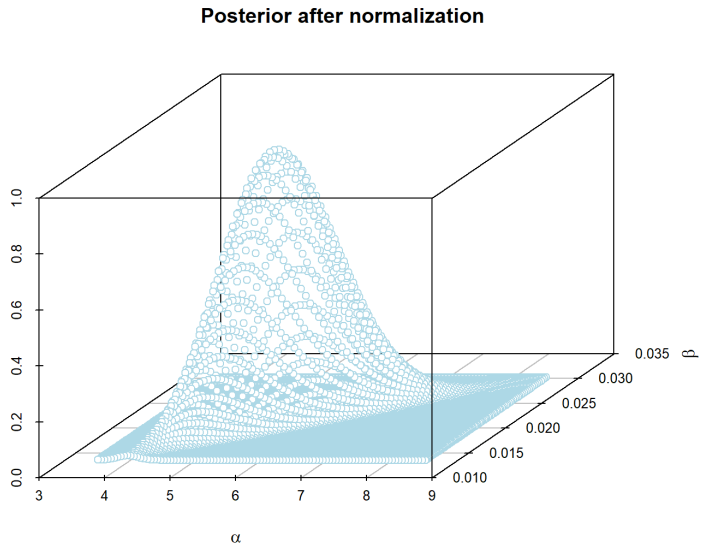
$$
\begin{aligned}
\int_a^b\int_A^B p(\alpha,\beta|y)d\alpha d\beta &\propto \int_a^b\int_A^B (\frac{\beta}{\beta+1})^{n\alpha}\prod_{i=1}^{n}(\frac{\Gamma(y_i+\alpha)}{\Gamma(\alpha)})p(\beta)/(\beta+1)^{\sum_{i=1}^{n}y_i}d\alpha d\beta \\
&\leq \int_a^b\int_A^B (\frac{\beta}{\beta+1})^{n\alpha}\prod_{i=1}^{n}(\frac{\Gamma(y_i+\alpha)}{\Gamma(\alpha)})p(\beta)d\alpha d\beta \\
&\leq \int_a^b\int_A^B (\frac{\beta}{\beta+1})^{n\alpha}(N+\alpha)^{M}p(\beta)d\alpha d\beta \\
&\leq \int_a^b\int_A^B 1^{n\alpha}(N+\alpha)^{M}p(\beta)d\alpha d\beta \\
&= \int_a^b\int_A^B (N+\alpha)^{M}\frac{1}{\beta(log(b)-log(a))}d\alpha d\beta \\
&\propto \int_A^B (N+\alpha)^{M}d\alpha \\
&< \infty
\end{aligned}
$$

where N and M are some constants in computing Gamma functions. Since the integral of marginal posterior distribution is finite, thus my choice of hyperprior results in a proper marginal posterior.

```
library(scatterplot3d)
log_posterior_alpha_log_beta = function(alpha, log_beta) {
  log_likelihood(alpha, exp(log_beta)) + log(dlunif(exp(log_beta), exp(-LIMIT), exp(LIMIT), base =
exp(1)))
}

nums = 100
alpha_grid = seq(3.5, 8.5, length.out=nums)
log_beta_grid = seq(-4.3, -3.5, length.out=nums)
grids = expand.grid(alpha_grid, log_beta_grid) # create a grid for the 2-d tuples
val_log_posterior = mapply(log_posterior_alpha_log_beta, grids$Var1, grids$Var2)
normalization_item = max(val_log_posterior)
val_log_posterior = val_log_posterior - normalization_item # normalize
```

```
par(mfrow=c(1,2))
scatterplot3d(grids$Var1, exp(grids$Var2), val_log_posterior, color="lightblue",pch=21, xlab=expression(alp
ha), ylab=expression(beta), zlab="", main="Log Posterior after normalization")
scatterplot3d(grids$Var1, exp(grids$Var2), exp(val_log_posterior), color="lightblue",pch=21, xlab=expressio
n(alpha), ylab=expression(beta), zlab="", main="Posterior after normalization")
```





4. Use the optim function in R to find both the mode and the Hessian at the mode for the logarithm of $p(\alpha, \beta|y)$. Construct the corresponding Normal approximation, and provide a visualization comparing this approximation to the actual marginal posterior.

```
log_posterior_alpha_beta = function(par) log_posterior_alpha_log_beta(par[1], log(par[2]))
optimal = optim(c(1, 1), log_posterior_alpha_beta, control=list(fnscale=-1), hessian=TRUE)
u = optimal$par
hessian = optimal$hessian
cov = -solve(hessian)
```

```
u
```

```
## [1] 5.53166694 0.01972209
```

```
hessian
```

```
##             [,1]         [,2]
## [1,]  -27.91994     7166.509
## [2,] 7166.50877 -2016353.835
```

Use sample function to draw (α, β) from density function

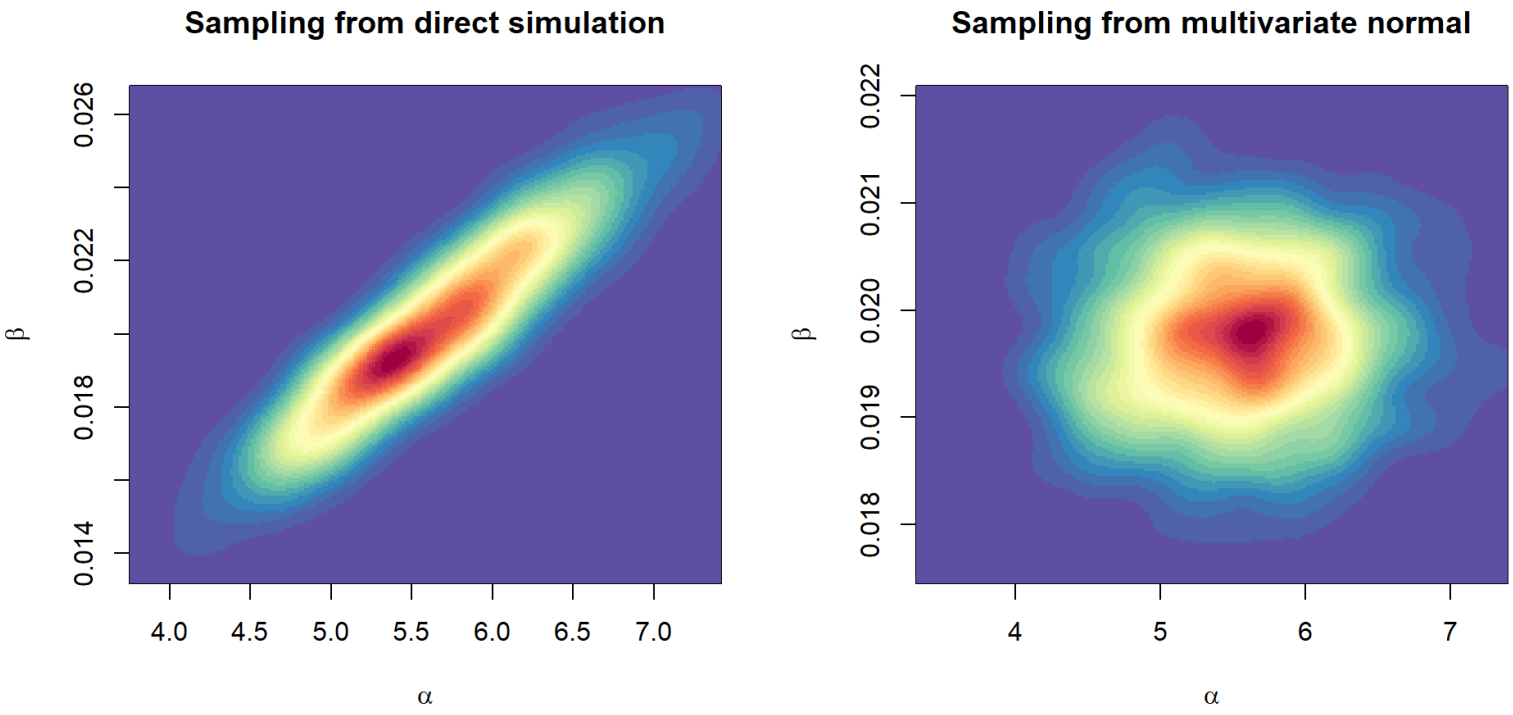
```
trials = 10000
sample_posterior = function(par) exp(log_posterior_alpha_beta(par) - normalization_item) # for fear that th
e highest likelihood is still too small
alpha_grid = sample(seq(u[1] - sqrt(cov[1,1]) * 3, u[1] + sqrt(cov[1,1]) * 3, length.out=trials),
replace=TRUE, size = trials)
beta_grid = sample(seq(u[2] - sqrt(cov[2,2]) * 3, u[2] + sqrt(cov[2,2]) * 3, length.out=trials), replace=T
RUE, size = trials)
grid = rbind(alpha_grid, beta_grid)
den_prop = apply(grid, 2, sample_posterior)
idx = sample(seq(1, trials), size =1000,replace = TRUE, prob = den_prop)
samples_posterior = t(grid[,idx])
```

Compute the normal approximation and draw (α, β)

```
C = chol(cov)
num_draws = 1000
mu = matrix(rep(u, num_draws), nrow=2)
yi = rbind(rnorm(num_draws), rnorm(num_draws))
samples_2d_normal = t(mu + C %*%  yi)
```

Visualize the results in terms of (α, β)

```
library(RColorBrewer)
library(MASS)
rf = colorRampPalette(rev(brewer.pal(11,'Spectral')))
r = rf(32)
par(mfrow=c(1,2))
k = kde2d(samples_posterior[,1], samples_posterior[,2], n=200)
image(k, col=r, xlab=expression(alpha), ylab=expression(beta), main="Sampling from direct simulation")
k = kde2d(samples_2d_normal[,1], samples_2d_normal[,2], n=200)
image(k, col=r, xlab=expression(alpha), ylab=expression(beta), main="Sampling from multivariate normal")
```



5. Draw 1000 values of (α, β) from their marginal posterior using a discrete grid approximation (making sure to describe your choice of discrete grid), and draw 1000 values of (α, β) from the Normal approximation to their marginal posterior.

In fact, we already got the result from Q4, samples_posterior is drawn from the posterior, samples_2d_normal comes from the approximation.

```
head(samples_posterior)
```

```
##      alpha_grid  beta_grid
## [1,]   5.590528 0.02009665
## [2,]   5.004987 0.01778080
## [3,]   5.275325 0.01868402
## [4,]   5.105836 0.01782646
## [5,]   5.373107 0.01969712
## [6,]   5.891543 0.02064886
```

```
head(samples_2d_normal)
```

```
##          [,1]       [,2]
## [1,] 5.830553 0.02003795
## [2,] 5.901819 0.01904897
## [3,] 5.305866 0.01945160
## [4,] 6.765882 0.02020136
## [5,] 6.029418 0.01918648
## [6,] 5.569827 0.02041620
```

6. Compare the previous two sets of draws by calculating the following summaries for each: 0.25, 0.5, 0.75 quantiles, mean, standard deviation, and kurtosis for the individual hyperparameters, and the correlation between the hyperparameters. Do you believe that your final inferences will be sensitive to the choice of approximation for the marginal posterior? Why or why not?

Based on the prior I used, it makes more sense to sample from normal approximation because its correlation coefficient is closer to zero.

```
library(moments)
mySummary = function(dat) {
  r = list()
  r$quantitle = rbind(quantile(dat[,1], probs=c(0.025,0.25,0.5,0.75,0.975)), quantile(dat[,2], probs=c(0.0
25,0.25,0.5,0.75,0.975)))
  r$mean = c(mean(dat[,1]), mean(dat[,2]))
  r$sd = c(sd(dat[,1]), sd(dat[,2]))
  r$skewness = skewness(dat)
  r$kurtosis = kurtosis(dat)
  r$correlation_coefficient = cor(dat[,1], dat[,2])
  rownames(r$quantitle) = names(r$mean) = names(r$sd) = names(r$skewness) = names(r$kurtosis) = c("alpha",
 "beta")
  return(mapply(round, r, 3))
}
```

```
mySummary(samples_posterior)
```

```
## $quantitle
##        2.5%   25%   50%   75% 97.5%
## alpha 4.494 5.183 5.618 6.102 6.970
## beta  0.016 0.019 0.020 0.022 0.025
##
## $mean
## alpha  beta
## 5.646 0.020
##
## $sd
## alpha  beta
## 0.637 0.002
##
## $skewness
## alpha  beta
## 0.126 0.109
##
## $kurtosis
## alpha  beta
## 2.761 2.739
##
## $correlation_coefficient
## [1] 0.95
```

```
mySummary(samples_2d_normal)
```

```
## $quantitle
##        2.5%   25%  50%   75% 97.5%
## alpha 4.307 5.095 5.54 5.937 6.739
## beta  0.018 0.019 0.02 0.020 0.021
##
## $mean
## alpha  beta
## 5.518 0.020
##
## $sd
## alpha  beta
## 0.630 0.001
##
## $skewness
##  alpha   beta
## -0.012  0.036
##
## $kurtosis
## alpha  beta
## 2.999 2.968
##
## $correlation_coefficient
## [1] 0.059
```

7. For each hyperparameter-pair drawn from the approximation of your choice, draw a sample of the λ_i's for all the observations. Plot the posterior mean of each λ_i along with the corresponding observation y_i as in Lecture 8. Comment on what you observe.

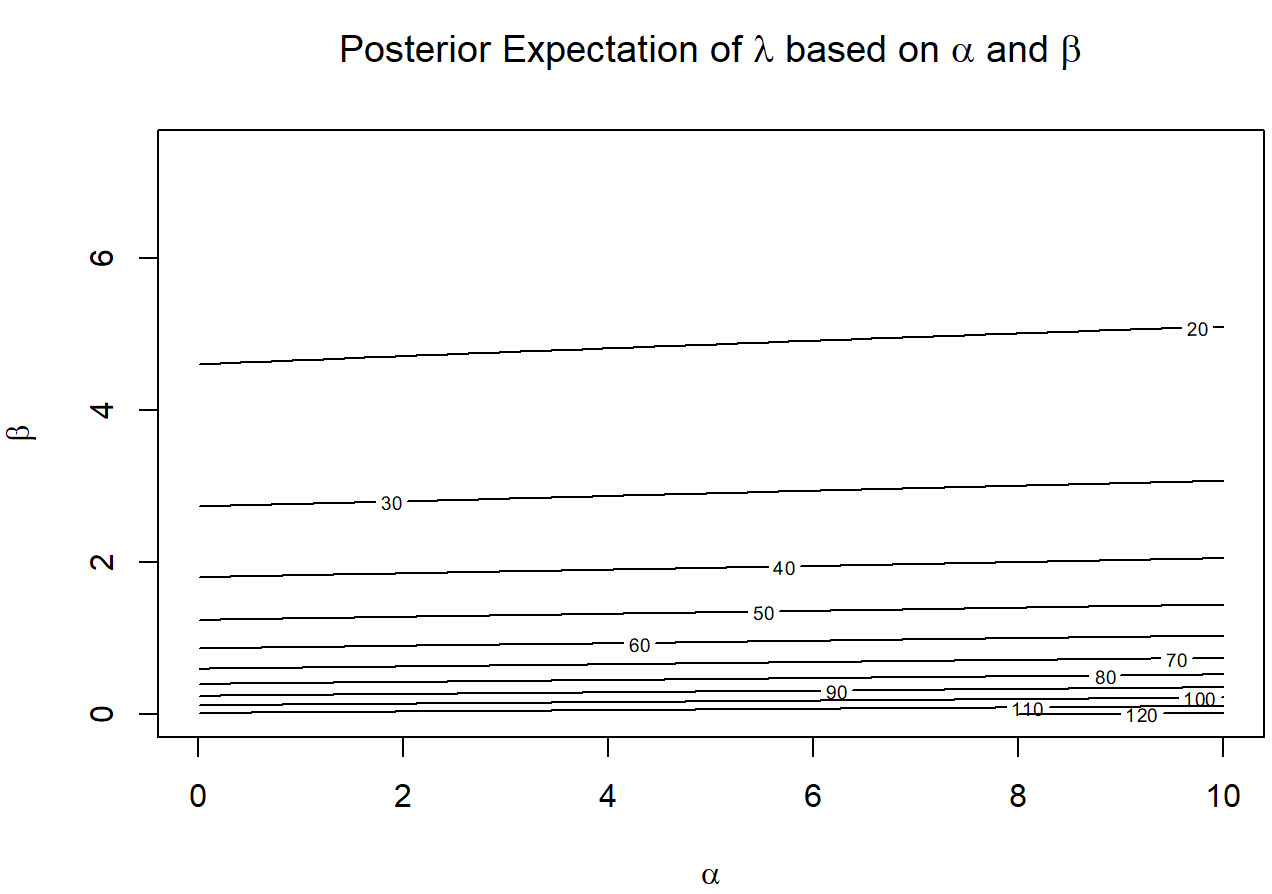
$$
\begin{aligned}
p(\lambda_i|y_i,\alpha,\beta) &= \frac{p(y_i|\lambda_i,\alpha,\beta)p(\lambda_i|\alpha,\beta)}{p(y_i|\alpha,\beta)} \\
&= \frac{\lambda_i^{y_i+\alpha-1}e^{-\lambda_i(\beta+1)}(\beta+1)^{y_i+\alpha}}{\Gamma(y_i+\alpha)}
\end{aligned}
$$

Thus, the expectation of λ is

$$
\begin{aligned}
E(\lambda_i|y_i,\alpha,\beta) &= \frac{(\beta+1)^{y_i+\alpha}}{\Gamma(y_i+\alpha)}\int_0^\infty \lambda_i\lambda_i^{y_i+\alpha-1}e^{-\lambda_i(\beta+1)}d\lambda_i \\
&= \frac{(\beta+1)^{y_i+\alpha}}{\Gamma(y_i+\alpha)}\frac{\Gamma(y_i+\alpha+1)}{(\beta+1)^{y_i+\alpha+1}} \\
&= \frac{y_i+\alpha}{\beta+1}
\end{aligned}
$$

We only display the posterior expectations of λ_1 conditional on α and β

```
expectation = function(alpha, beta) (Y[1] + alpha) / (beta + 1)

gnum = 100
alpha_grid = seq(0.01, 10, length.out=gnum)
beta_grid = exp(seq(-10, 2, length.out=gnum))

conditional_mean_lambda_evaluations = matrix(NA, nrow=length(alpha_grid), ncol=length(beta_grid))
for (i in 1: gnum) {
  for (j in 1: gnum) {
    conditional_mean_lambda_evaluations[i,j] = expectation(alpha_grid[i], beta_grid[j])
  }
}

contour(alpha_grid, beta_grid, conditional_mean_lambda_evaluations,
        xlab=bquote(paste(alpha)),
        ylab=bquote(paste(beta)),
        main=bquote(paste("Posterior Expectation of "*lambda*" based on "*alpha*" and "*beta)))
```



From the figure above, we can see that α doesn't have too much influence on the expectation of λ, β does.

Applied

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. You are employed as a consultant for the lab. In an initial meeting with the Notlem scientists, you were informed that they intend to conduct their study in the following manner.

1. Place stem cells in each well of a 24-well rectangular plate.

2. Assign pairs of concentrations for the two chemical modulators to each well in the manner you specify.

3. Incubate the entire plate.

4. Measure the expression level of a specific protein for each well.

5. Transform the observed expression levels into continuous conversion measures" that reside in R, with larger values signifying better conversion.

The scientists also state that:

1, their resource and time constraints limit the number of experimental runs in the study to 72, i.e., three plates of 24-wells,

2, 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

3, polynomial models of conversion as a function of chemical modulators' concentrations have been shown to enjoy some measure of success for this type of problem in previous literature.

Your task is to:

1, tell the scientists the specific pairs of concentrations to run in the study according to the procedure above,

2, analyze the resulting data to build a Bayesian regression model of stem cell conversion as a function of the two chemical modulators' concentrations,

3, use the model to calculate the posterior predictive distribution of the concentration settings that yield maximum conversion, and finally

4, construct the posterior predictive distribution of the conversion corresponding to a specific point prediction of the concentration that yields maximum conversion.

Any design strategy you decide to adopt must satisfy the scientists' resource constraints. Please be sure to note that an entire plate will be incubated at one point in time. Thus, your only possible design strategies are to either send:

1, one e-mail containing 72 runs to be conducted at one point in time,

2, two e-mails at separate times, with the first containing the initial 24 runs, and the second containing the final 48 runs,

3, two e-mails at separate times, with the first containing the initial 48 runs, and the second containing the final 24 runs, and finally

4, three e-mails at separate times, each containing 24 runs.

See sample_design.csv for a sample design CSV file with 24 design points. Please follow this format exactly (use the write.table command with the col.names and row.names options set to FALSE). The scientists will e-mail you the results in an augmented CSV file. No credit will be given unless you provide all your reasoning, computations, and discussion of results in a concise and coherent manner.

First, let's normalize the input

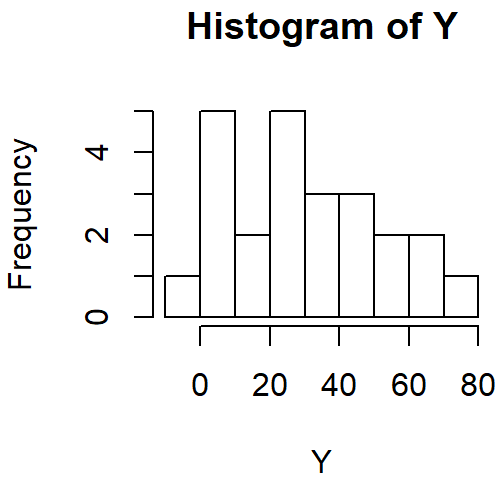
```
setwd("C:/Users/Wei/Desktop/STAT 695/HW2")
data = read.csv(file="rslt.csv", header=FALSE, sep = " ")
x = data[,1:2]
Y = data[,3]

X = matrix(rep(0, 2 * nrow(x)), ncol=2)
for (i in 1: ncol(X)) X[, i] = (x[, i] - mean(x[, i])) / sd(x[, i])
```

What does the the target data looks like

```
sd(Y)
```

```
## [1] 22.39741
```

```
hist(Y)
```



From the histogram, we can assume Y follows normal distribution. If we believe the data points follow joint Gaussian distribution, then Gaussian process regression seems a good idea in this case, it helps us fit the unknown function with error bar. Since the standard deviation of Y is 22.

I first set the hyperparameters of signal variance σ_f and noise variance σ_n^2 as $\sigma_f^2 = \sigma_n^2 = 20^2$, moreover, the lengthscale $l = 1$.

```
sigma_n = 20
sigma_f = 20
l = 1
```

This setting can be a very rough and conservative approximation and I will tune the hyperparameter later.

From GPML by Rasmussen and Williams, we know that if $f(x) \sim Gaussian\ Process(m(x), k(x, x'))$, then

$\bar{f}_* = m(X_*) + K(X_*, X)K_y^{-1}(y - m(X))$

$V(f_*) = K(X_*, X_*) - K(X_*, X)K_y^{-1}K(X, X_*)$

where $K_y = K + \sigma_n^2 I$, $K(\cdot)$ is squared exponential kernel, link here (http://www.cs.toronto.edu/~duvenaud/cookbook/).

```
# Calculate squared exponential kernel
# l: lengthscale, determines the length of the 'wiggles' in your function
# sigma_f: average distance of your function away from its mean
rbfKernel = function(X1, X2, l = 1, sigma_f = 1) {
  Kse = matrix(rep(0, nrow(X1) * nrow(X2)), nrow = nrow(X1))
  for (i in 1: nrow(Kse))
    for (j in 1: ncol(Kse))
      Kse[i,j] = exp(-0.5 * sum((X1[i, ] - X2[j, ])^2) / l^2) * sigma_f^2
  return(Kse)
}
```

To show us the performance of fitting, I just set the testing data the same as training data.

```
X_pred = X
```

Calculate relevant kernels in order to compute posterior mean and variance.

```
var_obs = rbfKernel(X, X, l, sigma_f)
Kconv = rbfKernel(X, X_pred, l, sigma_f)
var_pred = rbfKernel(X_pred, X_pred, l, sigma_f)
```

Now, get our expected distribution $f(x) \sim GP(f\_pred, fvar\_pred)$, I will update the way of solving inverse in the next part.

```
I_n = diag(1, ncol(var_obs))
f_pred = as.vector(mean(Y) + t(Kconv) %*% solve(var_obs + sigma_n^2 * I_n) %*% (Y - mean(Y)))
fvar_pred = var_pred - t(Kconv) %*% solve(var_obs + sigma_n^2 * I_n) %*% Kconv
```

Distinguish points within and outside error bar.

```
nums = length(Y)
expected_Y = rep(NA, nums)
outlier_Y = rep(NA, nums)
error_bar = 2*sqrt(diag(fvar_pred))
iff = (Y >= f_pred - error_bar) & (Y <= f_pred + error_bar)
expected_Y[iff] = Y[iff]
outlier_Y[!iff] = Y[!iff]
```

The fitting is not bad, nearly 80% data is within the confidence (credit) intervel

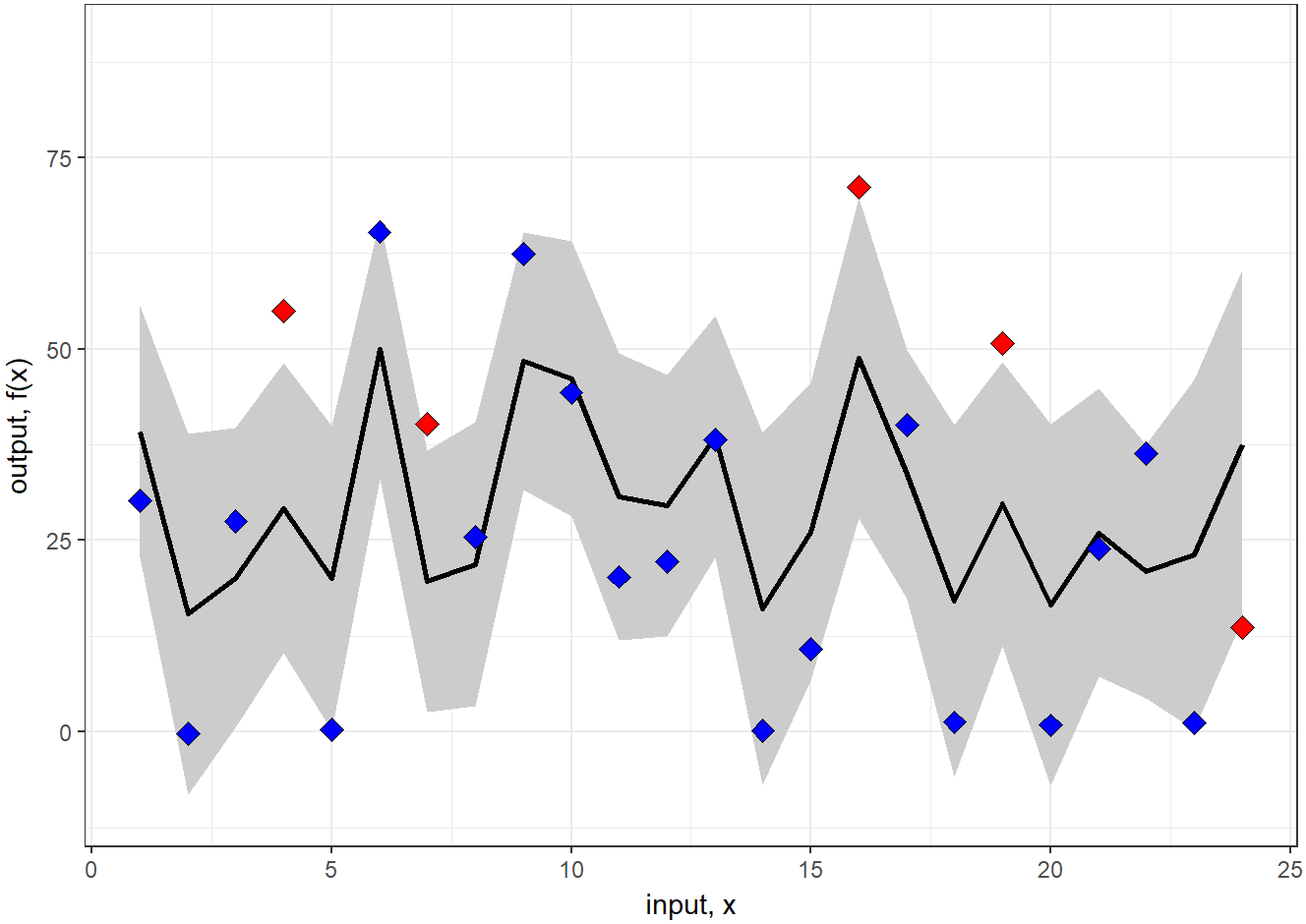
```
mean(sqrt(diag(fvar_pred)))
```

```
## [1] 9.646512
```

```
sum(iff) / length(iff)
```

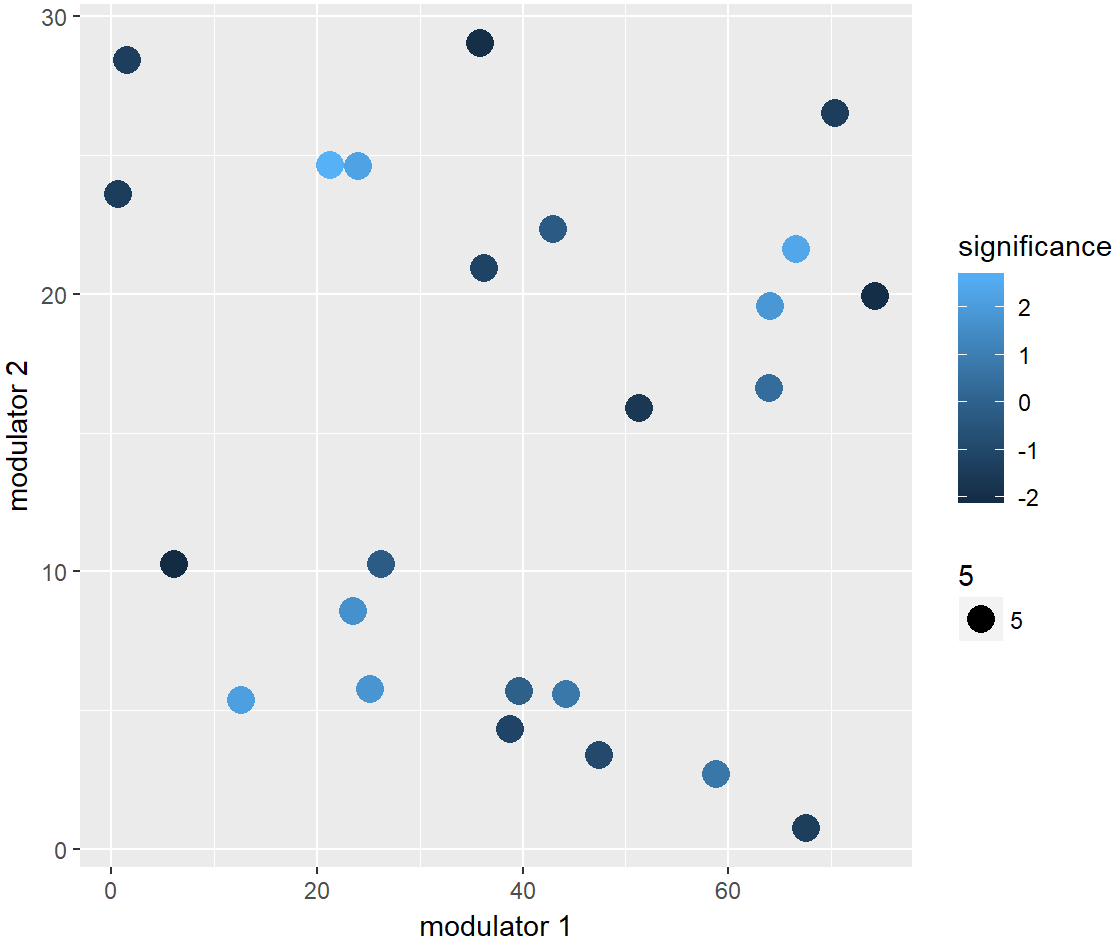
```
## [1] 0.7916667
```

```
library(ggplot2)
ggplot() +
geom_ribbon(data=NULL, aes(x=seq(1, nums), y=f_pred, ymin=f_pred - error_bar, ymax=f_pred + error_bar), fi
ll="grey80") + # error bar
geom_line(data=NULL, aes(x=seq(1, nums), y=f_pred), size=1) + # mean
geom_point(data=NULL, aes(x=seq(1, nums), y=expected_Y), size=3, shape=23, fill="blue") +  # observed data
 as expected
geom_point(data=NULL, aes(x=seq(1, nums), y=outlier_Y), size=3, shape=23, fill="red") +  # observed data n
ot fitting well
theme_bw() +
scale_y_continuous(lim=c(-10, 90), name="output, f(x)") +
xlab("input, x")
```



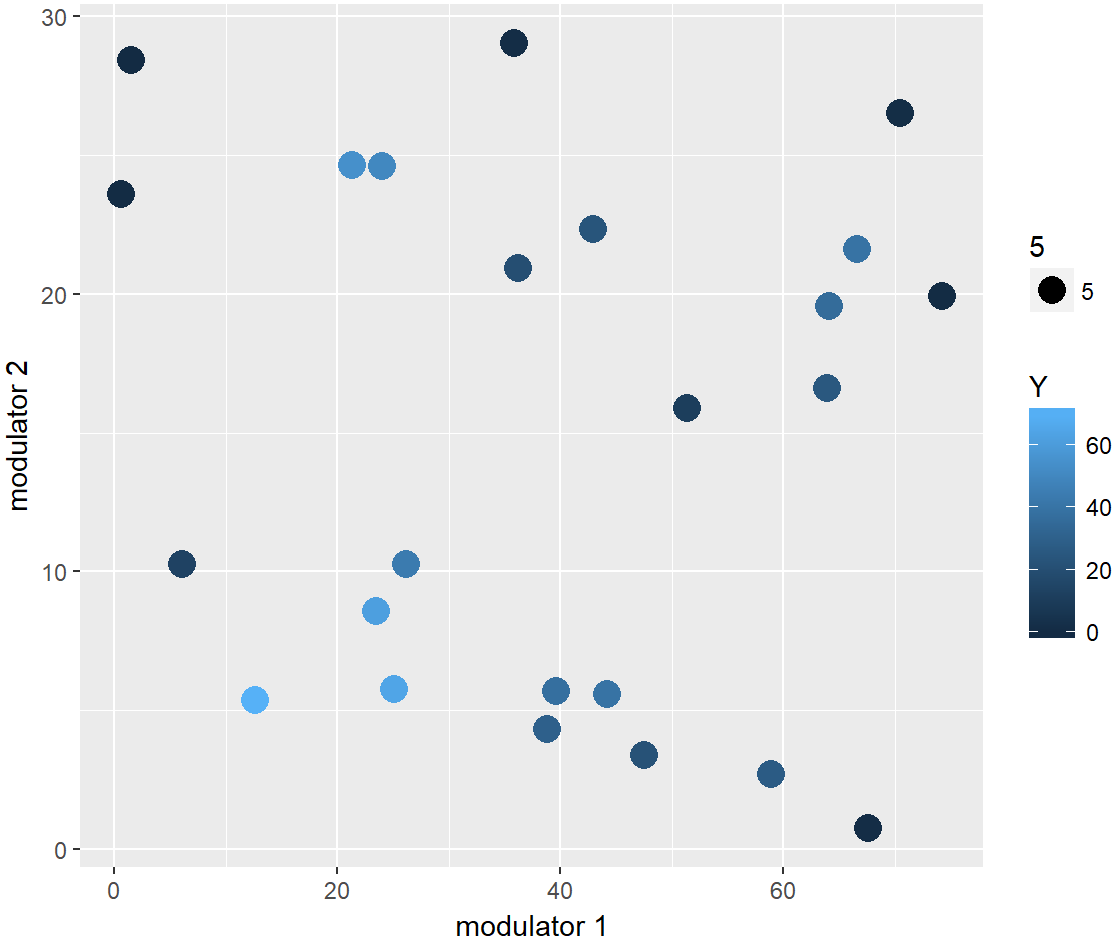
In the following figure, we can see three cluster of points are higher than predicted values, i.e. points near (19, 7), (24, 25) and (65, 22).

```
significance = (Y - f_pred) / sqrt(diag(fvar_pred))
ggplot(NULL, aes(x = x[,1], y = x[,2], color = significance, size=5)) +
geom_point() +
labs(x = "modulator 1") +
labs(y = "modulator 2")
```



In the figure below, we see that best possible pairs would be roughly (19, 7) > (24, 25) > (65, 22).

```
ggplot(NULL, aes(x = x[,1], y = x[,2], color = Y, size=5)) +
geom_point() +
labs(x = "modulator 1") +
labs(y = "modulator 2")
```



Following this rule, we request **the most data points around (19, 7)**, lots data points around (24, 25) and (65, 22), for fear that the maximum conversion pair occurs near the boundary, we should also **take care of the boundary points**. File rslt_V2.csv contains 72 pairs of data.

```
data = read.csv(file="rslt_V2.csv", header=FALSE, sep = " ")
x = data[,1:2]
Y = data[,3]
```

Ignore those points with low conversion ratio.

```
iff = Y > 10
Y = Y[iff]
x = x[iff,]
```

Standardize the input

```
X = matrix(rep(0, 2 * nrow(x)), ncol=2)
for (i in 1: ncol(X)) X[, i] = (x[, i] - mean(x[, i])) / sd(x[, i])
```

In addition, shrink the hyperparameter σ_n because noise in the observed data should not be as large as 20, reduce l since we have more data points and we want to be more radical to make the fitted line more wiggle.

```
sigma_n = 5
sigma_f = 20
l = 0.5
```

The prediction goes as follows.

```
X_pred = X
var_obs = rbfKernel(X, X, l, sigma_f)
Kconv = rbfKernel(X, X_pred, l, sigma_f)
var_pred = rbfKernel(X_pred, X_pred, l, sigma_f)
```

Write cholesky decomposition to stabilize and speed up the inverse (altough didn't make a difference in this case)

```
L = chol(var_obs + sigma_n^2 * diag(1, ncol(var_obs)))
invL = solve(L)
alpha = invL %*% t(invL) %*% (Y - mean(Y))
f_pred = as.vector(mean(Y) + t(Kconv) %*% alpha)
v_ = t(invL) %*% Kconv
fvar_pred = var_pred - t(v_) %*% v_
```

Since $p(y|X) = \int p(y|f,X)p(f|X)df$, the log marginal likelihood follows:

$$logp(f|X) = -\frac{1}{2}f^T K^{-1} f - \frac{1}{2}log|K| - \frac{n}{2}log(2\pi)$$

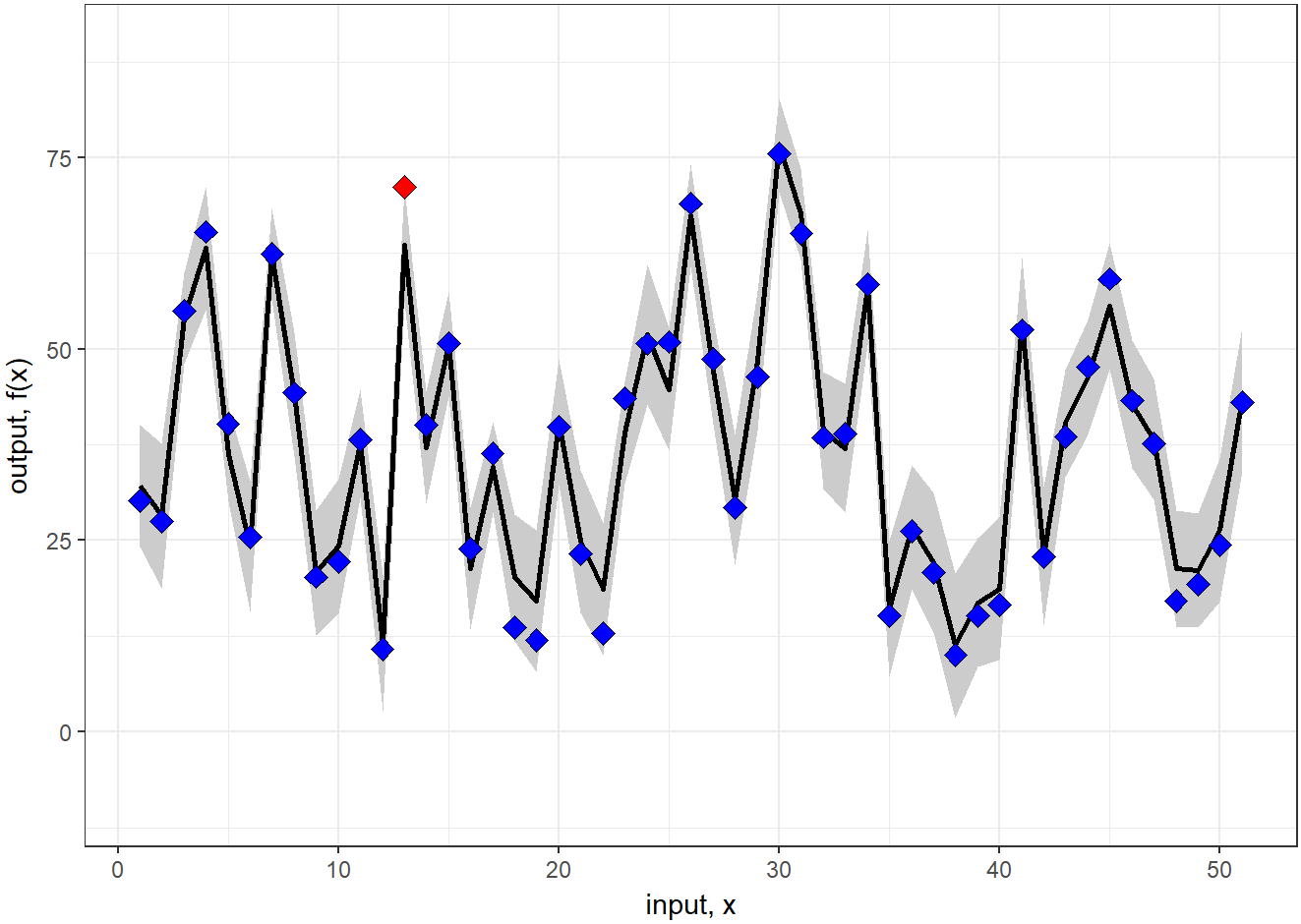
```
log_marginal_likelihood = function(X, Y) {
  -0.5 * t(Y - mean(Y)) %*% alpha - sum(log(diag(L))) - length(Y) / 2 * log(2 * pi)
}
```

Due to time limit, I didn't show you too much information about the log marginal likelihood. Let's see how does the 2nd fitting look like:

```
nums = length(Y)
expected_Y = rep(NA, nums)
outlier_Y = rep(NA, nums)
error_bar = 2*sqrt(diag(fvar_pred))
iff = (Y >= f_pred - error_bar) & (Y <= f_pred + error_bar)
expected_Y[iff] = Y[iff]
outlier_Y[!iff] = Y[!iff]
```

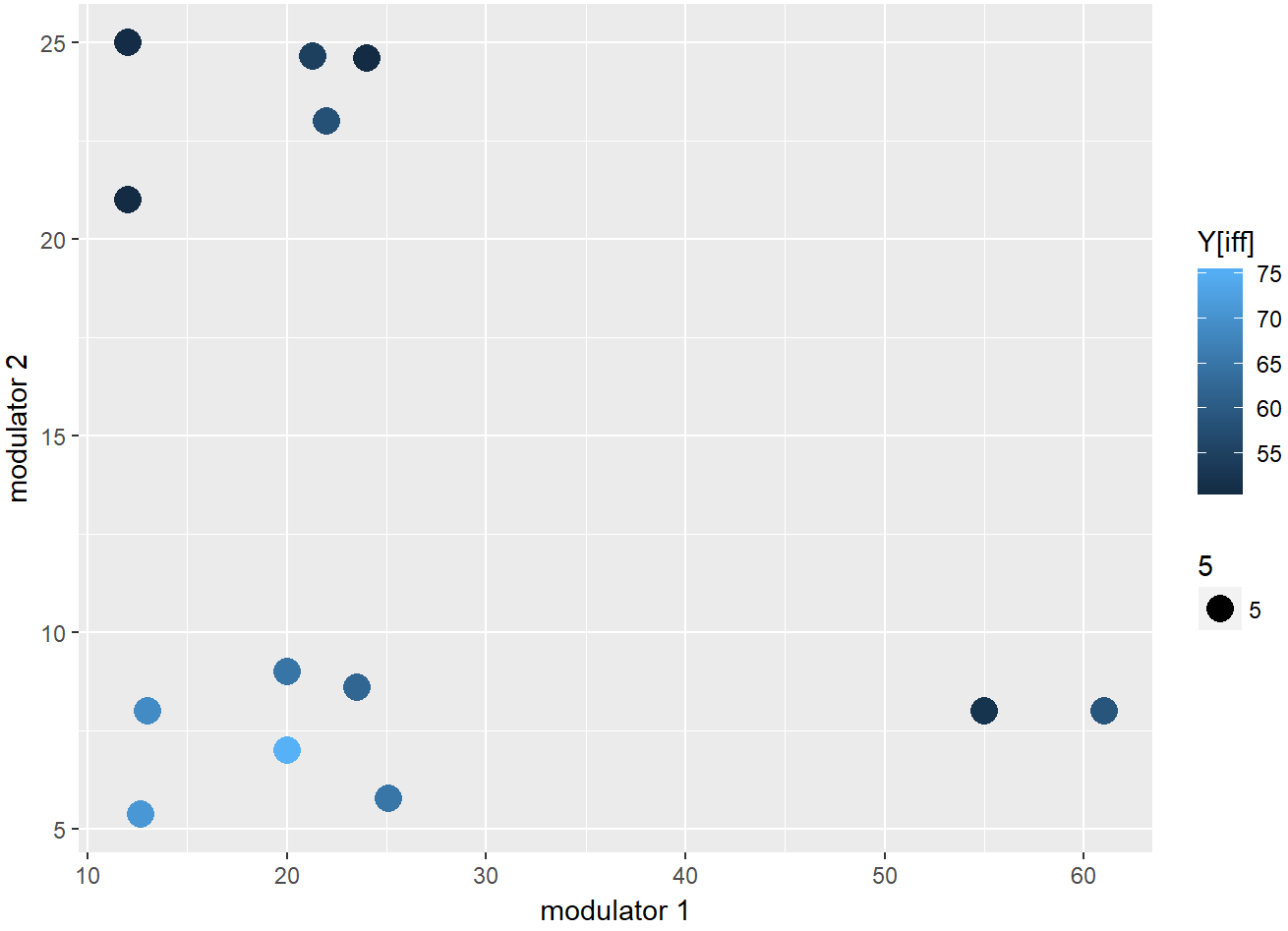
This time, the fitting is much better, as a matter of fact, the luckist thing is that the highest data point is within our error bar.

```
library(ggplot2)
ggplot() +
geom_ribbon(data=NULL, aes(x=seq(1, nums), y=f_pred, ymin=f_pred - error_bar, ymax=f_pred + error_bar), fi
ll="grey80") + # error bar
geom_line(data=NULL, aes(x=seq(1, nums), y=f_pred), size=1) + # mean
geom_point(data=NULL, aes(x=seq(1, nums), y=expected_Y), size=3, shape=23, fill="blue") +  # as expected
geom_point(data=NULL, aes(x=seq(1, nums), y=outlier_Y), size=3, shape=23, fill="red") +  # not fitting
well
theme_bw() +
scale_y_continuous(lim=c(-10, 90), name="output, f(x)") +
xlab("input, x")
```

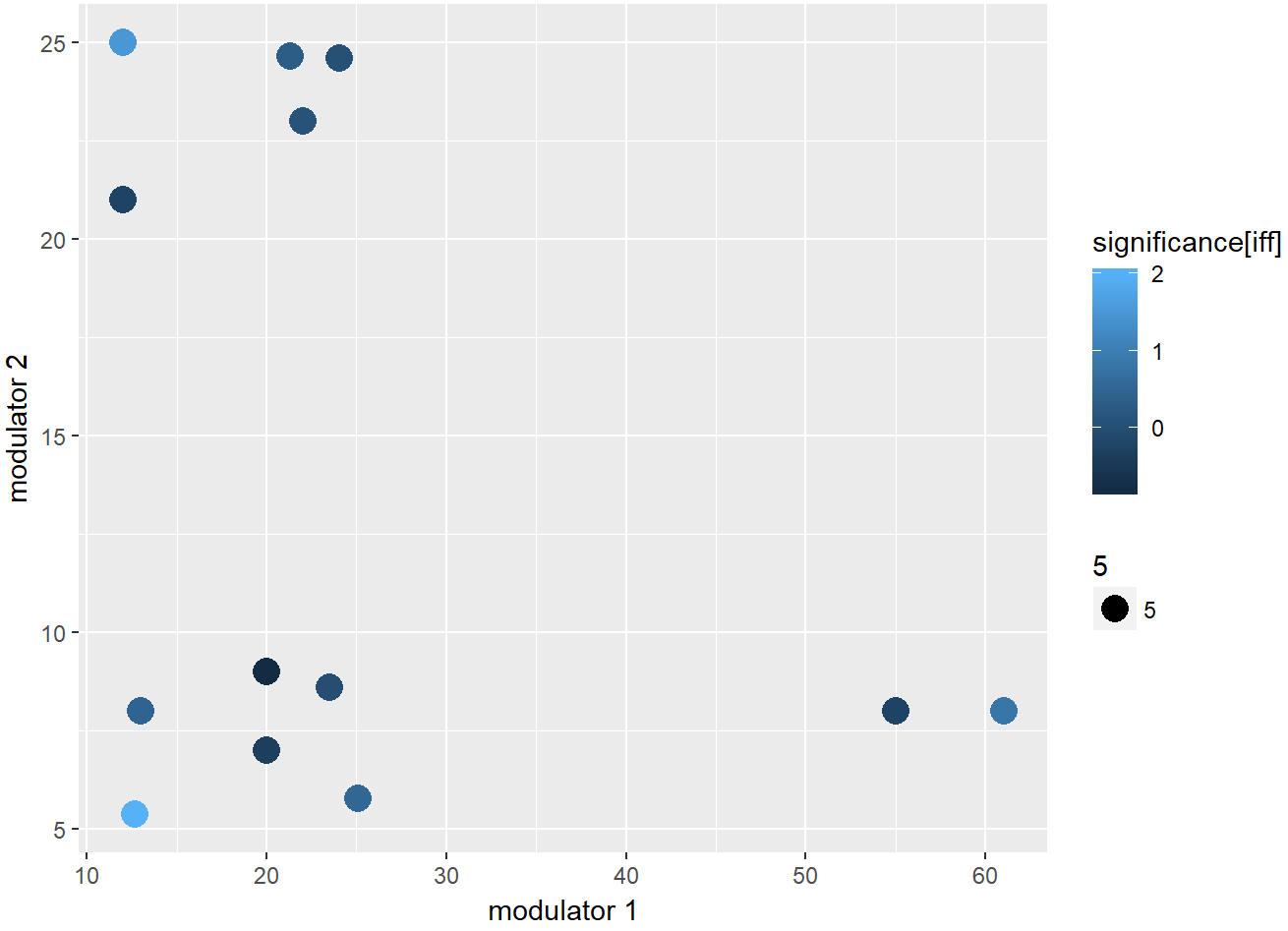


We see from the figure below that the highest conversion ratio appears at (20, 7).

```
iff = Y > 50 # filter data points with conversion ratio lower than 50
ggplot(NULL, aes(x = x[,1][iff], y = x[,2][iff], color = Y[iff], size=5)) +
geom_point() +
labs(x = "modulator 1") +
labs(y = "modulator 2")
```




```
significance = (Y - f_pred) / sqrt(diag(fvar_pred))
ggplot(NULL, aes(x = x[,1][iff], y = x[,2][iff], color = significance[iff], size=5)) +
geom_point() +
labs(x = "modulator 1") +
labs(y = "modulator 2")
```



From the figure above, we can focus on a smaller data set, namely $[10, 30] \times [5, 10]$, generate our testing data as shown in the code and **do remember to normalize the data**.

```
X_pred = expand.grid(seq(10, 30, len=10),
                     seq(5, 10, len=10))
for (i in 1: ncol(X)) X_pred[, i] = (X_pred[, i] - mean(x[, i])) / sd(x[, i])
```

Compute relevant kernels, this time it requires quite a long time to compute.

```
var_obs = rbfKernel(X, X, l, sigma_f)
Kconv = rbfKernel(X, X_pred, l, sigma_f)
var_pred = rbfKernel(X_pred, X_pred, l, sigma_f)
```

Some trial computations again…

```
L = chol(var_obs + sigma_n^2 * diag(1, ncol(var_obs)))
invL = solve(L)
alpha = invL %*% t(invL) %*% (Y - mean(Y))
v_ = t(invL) %*% Kconv
```

Finally, the posterior predictive distribution mean and covariance function comes out.

```
f_pred = as.vector(mean(Y) + t(Kconv) %*% alpha)
fvar_pred = var_pred - t(v_) %*% v_
```

The point that yields the highest conversion ratio locates at (17.8, 6.84) with expected value 78.66.

```
df_pred = data.frame(x1=X_pred[, 1] * sd(x[, 1]) + mean(x[, 1]),
                     x2=X_pred[, 2] * sd(x[, 2]) + mean(x[, 2]),
                     y=f_pred)
df_pred[which.max(df_pred[,3]), ]
```

```
##          x1       x2        y
## 920 17.7551 6.836735 78.65652
```

Visualize the data.

```
library(scatterplot3d)
scatterplot3d(df_pred$x1, df_pred$x2, f_pred, color="blue",pch=21, xlab="modulator 1", ylab="modulator 2",
zlab="", main="Posterior predictive distribution")
```