STA 601 Homework 9

Lingyun Shao

Nov. 12, 2018

9.3

Crime: The file `crime.dat` contains crime rates and data on 15 explanatory variables for 47 U.S. states, in which both the crime rates and the explanatory variables have been centered and scaled to have variance 1. A description of the variables can be obtained by typing `library (MASS);?UScrime` in `R`.

a) Fit a regression model $y = \mathbf{X}\beta + \epsilon$ using the g-prior with g = n, $\nu_0 = 2$ and $\sigma_0^2 = 1$. Obtain marginal posterior means and 95% confidence intervals for β, and compare to the least squares estimates. Describe the relationships between crime and the explanatory variables. Which variables seem strongly predictive of crime rates?

```
crime = read.table(file = 'http://www2.stat.duke.edu/~pdh10/FCBS/Exercises/crime.dat',
                   header = TRUE, stringsAsFactors = FALSE)
y = crime[,1]
X = crime[,-1] %>% as.matrix()
S = 10000
g = n =length(y)
p = ncol(X)
nu.0 = 2
sig2.0 = 1
Hg = (g/(g+1)) * X %*% solve(t(X)%*%X) %*% t(X)
SSRg = t(y) %*% (diag(1, nrow = n) - Hg) %*% y
sig2 = 1/rgamma(S, (nu.0+n)/2, (nu.0*sig2.0+SSRg)/2)

Vb = g * solve(t(X) %*% X)/(g+1)
Eb = Vb %*% t(X) %*% y

E = matrix(rnorm(S*p, 0, sqrt(sig2)), S, p)
# independent Monte Carlo samples from posterior distibution
beta = t( t(E%*%chol(Vb)) + c(Eb))

beta_ols = solve(t(X) %*% X) %*% t(X) %*% y
CI.g = cbind(beta_ols, colMeans(beta),
             apply(beta, 2, function(x) quantile(x, 0.025)),
             apply(beta, 2, function(x) quantile(x, 0.975)))
colnames(CI.g) = c('OLS Est.', 'Posterior Mean', '2.5%', '97.5%')
kable(CI.g, caption = 'Posterior Confidence Interval', digits = 4)
```

Table 1: Posterior Confidence Interval

	OLS Est.	Posterior Mean	2.5%	97.5%
M	0.2865	0.2826	0.0375	0.5275
So	-0.0001	0.0010	-0.3317	0.3345
Ed	0.5445	0.5334	0.2098	0.8574
Po1	1.4716	1.4410	-0.0440	2.9054
Po2	-0.7818	-0.7628	-2.3007	0.7695
LF	-0.0660	-0.0648	-0.3360	0.2094

	OLS Est.	Posterior Mean	2.5%	97.5%
M.F	0.1313	0.1309	-0.1463	0.4118
Pop	-0.0703	-0.0685	-0.2919	0.1633
NW	0.1091	0.1041	-0.2131	0.4139
U1	-0.2705	-0.2671	-0.6283	0.0816
U2	0.3687	0.3631	0.0338	0.6987
GDP	0.2381	0.2292	-0.2392	0.6993
Ineq	0.7263	0.7093	0.2877	1.1355
Prob	-0.2852	-0.2790	-0.5190	-0.0396
Time	-0.0616	-0.0594	-0.2971	0.1794

Comparing to OLS estimates $\hat{\beta}_{ols} = (X^T X)^{-1} X^T y$, the posterior means $\hat{\beta}_{Bayes} = \frac{g}{g+1}\hat{\beta}_{ols}$ are shrunk towards 0. From the result of Table 1, we can see that the posterior averages based on simulation are closer to zero except for `So` which should be due to randomness. OLS estimates are included in the posterior confidence intervals.

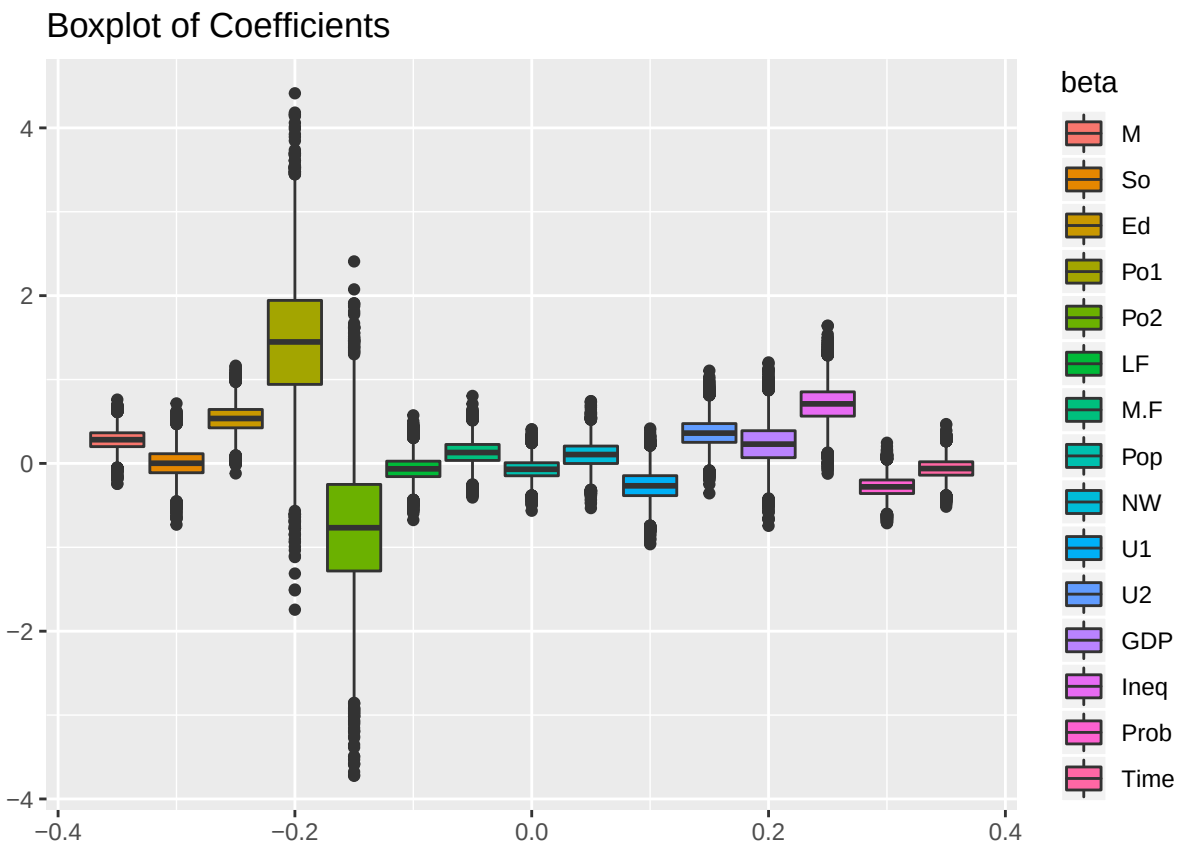
Based on Table 1, we can see that the posterior means of `M`, `So`, `Ed`, `Po1`, `M.F`, `NW`, `U2`, `GDP`, `Ineq` are positive, which means that these variables are positively related to the response variable `crime`, i.e. increase on these variables may lead to increase of `crime`.

The posterior means of `Po2`, `LF`, `Pop`, `U1`, `Prob`, `Time` are negetive, meaning that these variables are negatively related to the response variable `crime`, i.e. increase on these variables may lead to decrease of `crime`.

Also, we can see that the posterior confidence intervals of `So`, `Po1`, `Po2`, `LF`, `M.F`, `Pop`, `NW`, `U1`, `GDP`, `Prob`, `Time` include 0, which means that they are not statistically significantly non-zero, i.e. not strong predictive.

On the contrary, we have `M`, `Ed`, `U2`, `Ineq` being strongly predictive. Also notice that the bounds of `Po1`, `Prob` touch on 0, which suggests that `Po1` and `Prob` may also be predictive.

```
beta.dt = beta %>%
  as_tibble() %>%
  gather(beta, value) %>%
  mutate(beta = factor(beta, levels = rownames(CI.g)))

ggplot(data = beta.dt) +
  geom_boxplot(aes(y = value, fill = beta)) +
  labs(title = 'Boxplot of Coefficients', y = '')
```



Since the variables are all centered and scaled, the coefficients should all be on the same scale, then we can use a boxplot to exam the result. The boxplot seems to have a lot of 'outliers' beyond inner quantile ranges because the posterior distribution from g prior should have a heavier tail than normal distribution. From the plot, we can see that `M`, `Ed`, `Po1`, `U1`, `Ineq` and `Prob` seem strongly predictive.

b) Lets see how well regression models can predict crime rates based on the $\mathbf{X}$-variables. Randomly divide the crime roughly in half, into a training set $\{y_{tr}, \mathbf{X}_{tr}\}$ and a test set $\{y_{te}, \mathbf{X}_{te}\}$

i. Using only the training set, obtain least squares regression coefficients $\hat{\beta}_{ols}$. Obtain predicted values for the test data by computing $\hat{y}_{ols} = \mathbf{X}_{te}\hat{\beta}_{ols}$. Plot $\hat{y}_{ols}$ versus y_{te} and compute the prediction error $\frac{1}{n_{te}}\sum(y_{i,te} - \hat{y}_{i,ols})^2$.

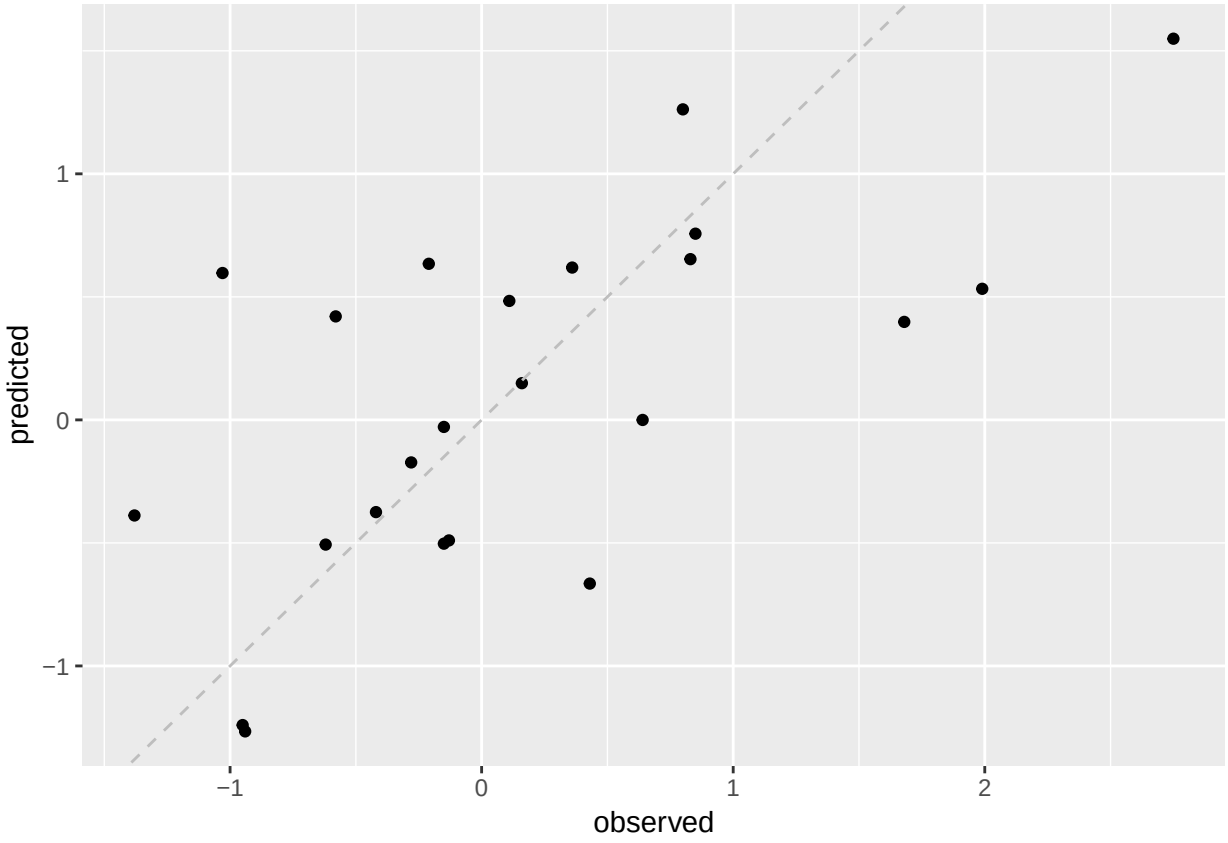
```
MSE = function(obs, pred) {mean((obs - pred)^2)}

ols_error = function(i=1) { # set argument i for sapply function
  train = sample(n, size = ceiling(n/2), replace = FALSE)
  X.train = X[train,]; X.test = X[-train,]
  y.train = y[train]; y.test = y[-train]
  beta = solve(t(X.train) %*% X.train) %*% t(X.train) %*% y.train
  y.pred = X.test %*% beta
  return(list(y.test = y.test, y.pred = y.pred, beta = beta, MSE = MSE(X.test %*% beta, y.test)))
}
set.seed(1)
train = sample(n, size = ceiling(n/2), replace = FALSE)
res.ols = ols_error()
res.ols$beta %>%
```

```
  t %>%
  kable(digits = 2, caption = 'OLS estimate')
```

Table 2: OLS estimate

M	So	Ed	Po1	Po2	LF	M.F	Pop	NW	U1	U2	GDP	Ineq	Prob	Time
0.22	-0.31	0.05	2.31	-1.72	-0.22	0.25	-0.21	0.3	-0.19	0.12	0.6	0.7	-0.33	-0.23

```
data.frame(observed = res.ols$y.test, predicted = res.ols$y.pred) %>%
  ggplot(aes(x = observed, y = predicted)) +
  geom_point() +
  geom_abline(slope = 1, intercept = 0, color = 'gray',
              lty = 2)
```




```
res.ols$MSE # OLS prediction error
```

```
## [1] 0.5699224
```

The OLS coefficients based on a randomly split training set are given in the above table. $\hat{y}_{ols}$ and y_{test} are ploted, where the gray line is $y = x$.

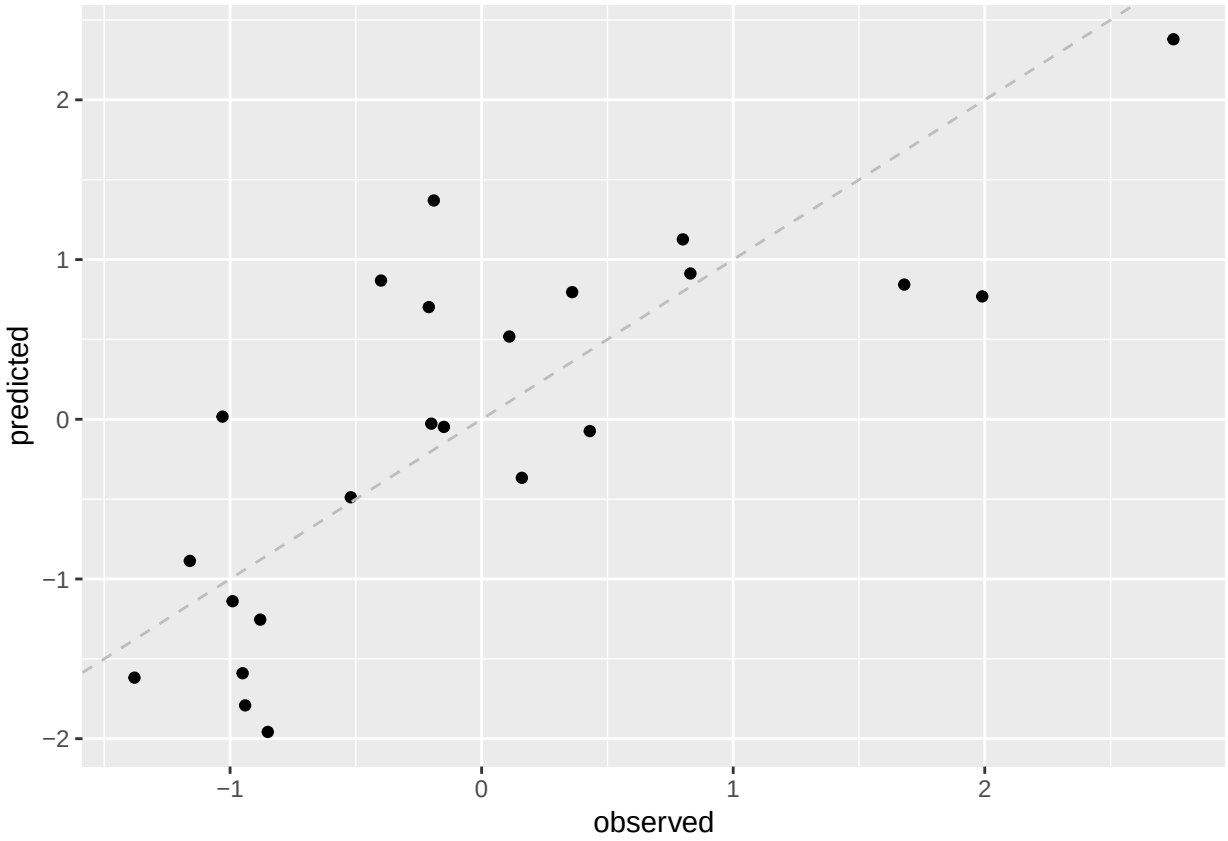
ii. Now obtain the posterior mean $\hat{\beta}_{Bayes} = E[\beta|y_{tr}]$ using the g-prior described above and the training data only. Obtain predictions for the test set $\hat{y}_{Bayes} = \mathbf{X}_{test}\hat{\beta}_{Bayes}$. Plot versus the test data, compute the prediction error, and compare to the OLS prediction error. Explain the results.

```r
bayes_error = function(i=1) { # set argument i for sapply function
  # train = sample(n, size = ceiling(n/2), replace = FALSE)
  X.train = X[train,]; X.test = X[-train,]
  y.train = y[train]; y.test = y[-train]
  g = n = length(y.train)
  S=1000
  Hg = (g/(g+1)) * X.train %*% solve(t(X.train)%*%X.train) %*% t(X.train)
  SSRg = t(y.train) %*% (diag(1, nrow = n) - Hg) %*% y.train
  sig2 = 1/rgamma(S, (nu.0+n)/2, (nu.0*sig2.0+SSRg)/2)

  Vb = g * solve(t(X.train) %*% X.train)/(g+1)
  Eb = Vb %*% t(X.train) %*% y.train

  E = matrix(rnorm(S*p, 0, sqrt(sig2)), S, p)
  # independent Monte Carlo samples from posterior distibution
  beta = colMeans(t( t(E%*%chol(Vb)) + c(Eb)))
  y.pred = X.test %*% beta
  return(list(y.test = y.test, y.pred = y.pred, beta = beta, MSE = MSE(X.test %*% beta, y.test)))
}

set.seed(1)
res.b = bayes_error()
res.b$beta %>%
  t %>%
  kable(digits = 2, caption = 'Bayes estimate')
```

Table 3: Bayes estimate

M	So	Ed	Po1	Po2	LF	M.F	Pop	NW	U1	U2	GDP	Ineq	Prob	Time
0.55	0.01	0.31	1.45	-0.72	0.12	-0.09	-0.16	0.18	-0.16	0.31	0.62	0.81	-0.44	-0.2

```r
data.frame(observed = res.b$y.test, predicted = res.b$y.pred) %>%
  ggplot(aes(x = observed, y = predicted)) +
  geom_point() +
  geom_abline(slope = 1, intercept = 0, color = 'gray',
              lty = 2)
```




```
res.b$MSE # OLS prediction error
```

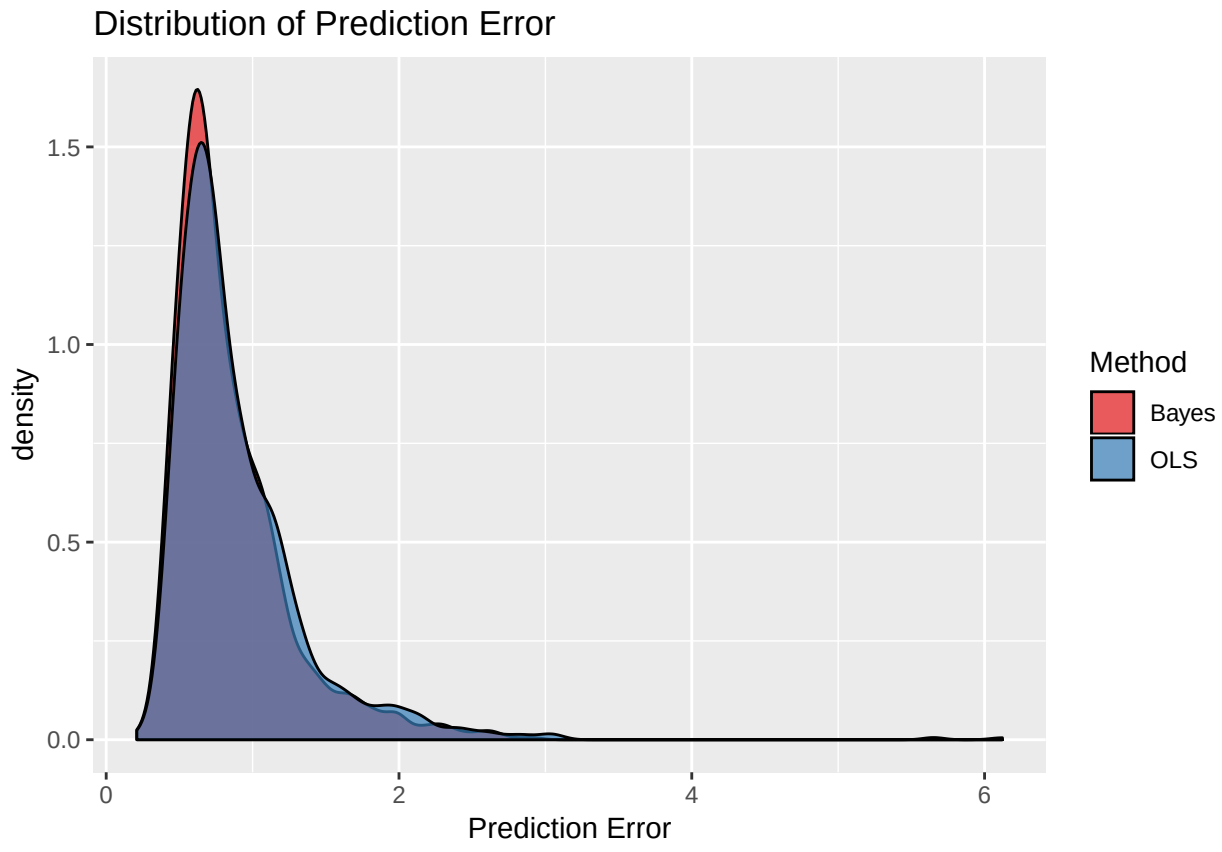
```
## [1] 0.5215692
```

From the results above, we can see that the prediction error of bayes model on test set is a bit smaller than the prediction error of ols on test set based on a randomly split training data. But we still need to check this claim on more training sets.

c) Repeat the procedures in b) many times with different randomly generated test and training sets. Compute the average prediction error for both the OLS and Bayesian methods.

```
k = 1000
ols_bayes_error = function(i=1) { # set argument i for sapply function
  train = sample(n, size = ceiling(n/2), replace = FALSE)
  X.train = X[train,]; X.test = X[-train,]
  y.train = y[train]; y.test = y[-train]
  beta.ols = solve(t(X.train) %*% X.train) %*% t(X.train) %*% y.train
  g = n = length(y.train)
  S=1000
  Hg = (g/(g+1)) * X.train %*% solve(t(X.train)%*%X.train) %*% t(X.train)
  SSRg = t(y.train) %*% (diag(1, nrow = n) - Hg) %*% y.train
  sig2 = 1/rgamma(S, (nu.0+n)/2, (nu.0*sig2.0+SSRg)/2)

  Vb = g * solve(t(X.train) %*% X.train)/(g+1)
  Eb = Vb %*% t(X.train) %*% y.train
```

```
  E = matrix(rnorm(S*p, 0, sqrt(sig2)), S, p)
  # independent Monte Carlo samples from posterior distibution
  beta.b = colMeans(t( t(E%*%chol(Vb)) + c(Eb)))
  return(c(MSE(X.test %*% beta.ols, y.test), MSE(X.test %*% beta.b, y.test)))
}

Error = sapply(1:k, ols_bayes_error)
Error_ols = Error[1,]
Error_Bayes = Error[2,]
dt.error = data_frame(
  er = c(Error_ols, Error_Bayes),
  cat = rep(c('OLS', 'Bayes'), each = k)
)
ggplot(data = dt.error) +
  geom_density(aes(er, fill = cat), alpha = 0.7) +
  scale_fill_brewer(palette = 'Set1') +
  labs(title = 'Distribution of Prediction Error',
       fill = 'Method', x = 'Prediction Error')
```




```
c(mean(Error_ols), mean(Error_Bayes), mean(Error_Bayes <Error_ols)) %>%
  t() %>%
  kable(caption = 'Average Prediction Error',
        col.names = c('OLS', 'Bayes', 'Pr(ols>bayes)'), digits = 5)
```

Table 4: Average Prediction Error

OLS	Bayes	Pr(ols>bayes)
0.89775	0.85277	0.933

By repeating the procedures in b), we get the a set of prediction errors using OLS and Bayesian method. From the distribution and Average, we can see that Bayesian method has a slightly lower prediction error. But the chance of getting a lower error rate based on the same training and test data is very high.

Problem 2

- Consider our hierarchical means model from before where we had m schools and in school k we observed n_k independent students' scores on some exam. The model we had for this was:

$$Y_{ij} \sim N(\theta_j, \sigma^2), \theta_j \sim N(\mu, \tau^2).$$

Instead of just modeling a mean we now want to model some kind of regression function within each school, but we still believe that there should be some sharing of information. Consider the following model:

$$Y_{ij} \sim N(\beta_j^t X_{ij}, \sigma^2), \beta_j \sim \mathcal{N}_p(\beta_0, \Sigma_0)$$

- Use the general normal prior on β_0, Wishart prior on Σ_0^{-1} and gamma prior on σ^{-2} and derive the Gibbs sampler for this model.
- Let there be 10 schools in this example with each school k having $n_k = 10k$ students.

Generate covariates in the following way:

$$X_{ij,1} = 1, X_{ij,2,...,p} \sim \mathcal{N}_{p-1}(\mathbf{0}, 0.8\mathbf{I} + 0.2\mathbf{1}\mathbf{1}^T)$$

(that is the correlation between the $X's$ is 0.2). Let $\beta_0 = 0, \sigma^2 = 1, \Sigma_0 = I$ and generate $\beta_1, ..., \beta_{10}$ and the Y_{ij} from the model. Run the Gibbs sampler you derived above for 10000 iterations, assess convergence of the different parameters and provide a summary of the posterior.

1. Full conditional distributions

We have m schools and each school k have n_k independent students' scores Y_{ik}, $i = 1, ..., n_k, k = 1, ..., m$. In this model setting, we have unknown quantities $\{\beta_1, ..., \beta_m\}, \beta_0, \Sigma_0, \sigma^2$ which we should do inference about. Joint posterior inference for these parameters can be made by constructing a Gibbs sampler. Notice $\beta_0, \Sigma_0, \sigma^2$ are fixed but unknown parameters in this model, so we should have prior distributions for them.

Suppose we have prior belief that

$$\begin{aligned} \beta_0 &\sim \mathcal{N}_p(\mu_0, \Lambda_0) \\ \Sigma_0^{-1} &\sim Wishart(\eta_0, S_0^{-1}) \\ \sigma^{-2} &\sim Gamma(\nu_0/2, \nu_0\sigma_0^2/2) \end{aligned}$$

Then the kernel of joint posterior distribution can be written as

$$
\begin{aligned}
&p(\beta_1, \ldots, \beta_m, \beta_0, \Sigma_0, \sigma^2 | y_1, \ldots, y_m, X_1, \ldots, X_m) \\
\propto &p(\beta_0, \Sigma_0, \sigma^2) p(\beta_1, \ldots, \beta_m | \beta_0, \Sigma_0, \sigma^2) p(y_1, \ldots, y_m | \beta_1, \ldots \beta_m, \beta_0, \Sigma_0, \sigma^2, X_1, \ldots, X_m) \\
= &p(\beta_0) p(\Sigma_0) p(\sigma^2) \prod_{j=1}^{m} p(\beta_j | \beta_0, \Sigma_0) \prod_{j=1}^{m} \prod_{i=1}^{n_j} p(y_{ij} | \beta_j, \sigma^2, X_{ij})
\end{aligned}
$$

Notice in the equation above:

- Using the same reasoning as in textbook 8.2.1, we can justify the exchangeability and therefore conditional independence in this hierarchical model.
- β_0, Σ_0 connect to y_i only indirectly through β_j. Thus given β_j, y_i is independent of β_0, Σ_0
- There is no obvious reason to depend our prior beliefs of $\beta_0, \Sigma_0, \sigma^2$ on each other, so they are independent.
- X_{ij} is a $p \times 1$ matrix (vector), standing for the p covariates of the ith individual in jth school.

1.1 Full conditional of β_0

From the kernel of the joint posterior density, we know that

$$
\begin{aligned}
&p(\beta_0 | \beta_1, ..., \beta_m, \Sigma_0, \sigma^2, y_1, \ldots, y_m, X_1, \ldots, X_m) \\
\propto &p(\beta_0) \prod_{j=1}^{m} p(\beta_j | \beta_0, \Sigma_0) \\
\propto &\exp(-\frac{(\beta_0 - \mu_0)^T \Lambda_0^{-1} (\beta_0 - \mu_0)}{2}) \times \exp(-\frac{\sum_{j=1}^{m} (\beta_j - \beta_0)^T \Sigma_0^{-1} (\beta_j - \beta_0)}{2}) \\
\propto &\exp(-\frac{\beta_0^T \Lambda_0^{-1} \beta_0 - 2\beta_0^T \Lambda_0^{-1} \mu_0 + \sum_{j=1}^{m} (\beta_0^T \Sigma_0^{-1} \beta_0 - 2\beta_0^T \Sigma_0^{-1} \beta_j)}{2}) \\
= &\exp(-\frac{\beta_0^T (\Lambda_0^{-1} + m\Sigma_0^{-1}) \beta_0 - 2\beta_0^T (\Lambda_0^{-1} \mu_0 + \Sigma_0^{-1} \sum_{j=1}^{m} \beta_j)}{2})
\end{aligned}
$$

From the form of the kernel, we know that the full conditional of β_0 is

$$
\mathcal{N}_p((\Lambda_0^{-1} + m\Sigma_0^{-1})^{-1} (\Lambda_0^{-1} \mu_0 + m\Sigma_0^{-1} \bar{\beta}), (\Lambda_0^{-1} + m\Sigma_0^{-1})^{-1})
$$

where $\bar{\beta} = \frac{1}{m} \sum_{j=1}^{m} \beta_j$ is a $p \times 1$ matrix (vector).

1.2 Full conditional of Σ_0

Since $\Sigma_0^{-1} \sim Wishart(\eta_0, S_0^{-1})$, we know $\Sigma_0 \sim Inverse - Wishart(\eta_0, S_0^{-1})$. From the kernel of the joint posterior density, we know that

$$
\begin{aligned}
&p(\Sigma_0|\beta_1,...,\beta_m,\beta_0,\sigma^2,y_1,\ldots,y_m,X_1,\ldots,X_m)\\
\propto& p(\Sigma_0)\prod_{j=1}^{m}p(\beta_j|\beta_0,\Sigma_0)\\
\propto&|\Sigma_0|^{-\frac{\eta_0+p+1}{2}}\exp(-tr(S_0\Sigma_0^{-1}/2))\times|\Sigma_0|^{-\frac{m}{2}}\exp(-\frac{\sum_{j=1}^{m}(\beta_j-\beta_0)^T\Sigma_0^{-1}(\beta_j-\beta_0)}{2})\\
=&|\Sigma_0|^{-\frac{\eta_0+m+p+1}{2}}\exp(-tr(S_0\Sigma_0^{-1}/2))\times\exp(-\frac{\sum_{j=1}^{m}tr[(\beta_j-\beta_0)^T\Sigma_0^{-1}(\beta_j-\beta_0)]}{2})\\
=&|\Sigma_0|^{-\frac{\eta_0+m+p+1}{2}}\exp(-tr(S_0\Sigma_0^{-1}/2))\times\exp(-\frac{\sum_{j=1}^{m}tr[(\beta_j-\beta_0)(\beta_j-\beta_0)^T\Sigma_0^{-1}]}{2})\\
=&|\Sigma_0|^{-\frac{\eta_0+m+p+1}{2}}\exp(-tr(S_0\Sigma_0^{-1}/2))\times\exp(-\frac{tr[\sum_{j=1}^{m}(\beta_j-\beta_0)(\beta_j-\beta_0)^T\Sigma_0^{-1}]}{2})\\
=&|\Sigma_0|^{-\frac{\eta_0+m+p+1}{2}}\exp(-tr(S_0\Sigma_0^{-1}/2))\times\exp(-\frac{tr(S_\beta\Sigma_0^{-1})}{2})\\
=&|\Sigma_0|^{-\frac{\eta_0+m+p+1}{2}}\exp(-\frac{tr((S_0+S_\beta)\Sigma_0^{-1})}{2})
\end{aligned}
$$

where $S_\beta=\sum_{j=1}^{m}(\beta_j-\beta_0)(\beta_j-\beta_0)^T$.

From the form of the kernel, we know that the full conditional of Σ_0 is

$$Inverse-Wishart(\eta_0+m,(S_0+S_\beta)^{-1})$$

The full conditional of Σ_0^{-1} is $Wishart(\eta_0+m,(S_0+S_\beta)^{-1})$.

1.3 Full conditional of σ^2

Since $\sigma^{-2}\sim Gamma(\nu_0,\nu_0\sigma_0^2)$, we know $\sigma^2\sim Inverse-Gamma(\nu_0,\nu_0\sigma_0^2)$. From the kernel of the joint posterior density, we know that

$$
\begin{aligned}
&p(\sigma^{-2}|\beta_1,...,\beta_m,\beta_0,\Sigma_0,y_1,\ldots,y_m,X_1,\ldots,X_m)\\
\propto&p(\sigma^{-2})\prod_{j=1}^{m}\prod_{i=1}^{n_j}p(y_{ij}|\beta_j,\sigma^2,X_{ij})\\
\propto&(\sigma^{-2})^{\frac{\nu_0}{2}-1}\exp(-\frac{\nu_0\sigma_0^2}{2}\sigma^{-2})\prod_{j=1}^{m}\prod_{i=1}^{n_j}\left\{(\sigma^{-2})^{\frac{1}{2}}\exp(-\frac{(y_{ij}-\beta_j^TX_{ij})^2}{2}\sigma^{-2})\right\}\\
=&(\sigma^{-2})^{\frac{\nu_0+\sum_{j=1}^{m}n_j}{2}-1}\exp(-\frac{\nu_0\sigma_0^2+\prod_{j=1}^{m}\prod_{i=1}^{n_j}(y_{ij}-\beta_j^TX_{ij})^2}{2}\sigma^{-2})
\end{aligned}
$$

From the form of the kernel, we know the full conditional of σ^{-2} is

$$Gamma(\frac{\nu_0+\sum_{j=1}^{m}n_j}{2},\frac{\nu_0\sigma_0^2+\prod_{j=1}^{m}\prod_{i=1}^{n_j}(y_{ij}-\beta_j^TX_{ij})^2}{2})$$

1.4 Full conditional of β_j

Collecting the terms in the joint posterior density that depend on β_j shows that the full conditional distribution of β_j must be proportional to

$$
\begin{aligned}
&p(\beta_j|\beta_{-j},\beta_0,\Sigma_0,\sigma^2,y_1,\ldots,y_m,X_1,\ldots,X_m)\\
\propto& p(\beta_j|\beta_0,\Sigma_0)\prod_{i=1}^{n_j}p(y_{ij}|\beta_j,\sigma^2,X_{ij})\\
\propto& \exp(-\frac{(\beta_j-\beta_0)^T\Sigma_0^{-1}(\beta_j-\beta_0)}{2})\exp(-\frac{\sum_{i=1}^{n_j}(y_{ij}-\beta_j^TX_{ij})^2}{2\sigma^2})\\
=& \exp(-\frac{(\beta_j-\beta_0)^T\Sigma_0^{-1}(\beta_j-\beta_0)}{2})\exp(-\underbrace{\frac{\sum_{i=1}^{n_j}(y_{ij}-\beta_j^TX_{ij})(\sigma^2)^{-1}(y_{ij}-\beta_j^TX_{ij})^T}{2}}_{\textit{A scalar can be viewed as a } 1\times1 \textit{ matrix}})\\
\propto& \exp(-\frac{\beta_j^T\Sigma_0^{-1}\beta_j-2\beta_j^T\Sigma_0^{-1}\beta_0+\sum_{i=1}^{n_j}(\beta_j^TX_{ij}\sigma^{-2}X_{ij}^T\beta_j-2\beta_j^TX_{ij}\sigma^{-2}y_{ij})}{2})\\
=& \exp(-\frac{\beta_j^T(\Sigma_0^{-1}+\sum_{i=1}^{n_j}X_{ij}X_{ij}^T/\sigma^2)\beta_j-2\beta_j^T(\Sigma_0^{-1}\beta_0+\sum_{i=1}^{n_j}X_{ij}y_{ij}/\sigma^2)}{2})
\end{aligned}
$$

where X_{ij} is a $p\times1$ matrix (vector), y_{ij} is a 1×1 matrix (scalar). From the form of the kernel, we know the full conditional of β_j is

$$
\mathcal{N}_p((\Sigma_0^{-1}+\sum_{i=1}^{n_j}X_{ij}X_{ij}^T/\sigma^2)^{-1}(\Sigma_0^{-1}\beta_0+\sum_{i=1}^{n_j}X_{ij}y_{ij}/\sigma^2),(\Sigma_0^{-1}+\sum_{i=1}^{n_j}X_{ij}X_{ij}^T/\sigma^2)^{-1})
$$

2. Gibbs sampler

Based on the full conditionals we have derived, we can easily define some update function in R. Once, we have the data, we can use these functions to get our MCMC samples from the joint posterior distribution.

Posterior approximation proceeds by iterative sampling of each unknown quantity from its full conditional distribution. This Gibbs sampling procedure is as follows:

0. give some 'reasonable' initial points of each unknown quantity $\left\{\beta_0^{(0)},\Sigma_0^{(0)},\sigma^{2(0)},\beta_1^{(0)},...,\beta_m^{(0)}\right\}$;
1. sample $\beta_0^{(s+1)}\sim p(\beta_0|\Sigma_0^{(s)},\beta_1^{(s)},...,\beta_m^{(s)})$;
2. sample $\Sigma_0^{(s+1)}\sim p(\Sigma_0|\beta_0^{(s+1)},\beta_1^{(s)},...,\beta_m^{(s)})$;
3. sample $\sigma^{2(s+1)}\sim p(\sigma^2|\beta_1^{(s)},...,\beta_m^{(s)},y_1,...,y_m,X_1,...,X_m)$;
4. for each $j\in\{1,...,m\}$, sample $\beta_j^{(s+1)}\sim p(\beta_j|\beta_0^{(s+1)},\Sigma_0^{(s+1)},\sigma^{2(s+1)},y_j,X_j)$.

2.1 Define update funtions of Gibbs sampler

Here we define update functions in R:

```
# Notes:
# 1. prior hyperparameters and data are defined in global environment
# 2. update functions only take parameters updated in the sampling process
# 3. betas is a p x m matrix, each column is a beta_j for school j
update_beta0 = function(Sig0, betas) {
  Cov = solve(solve(Lam0) + m * solve(Sig0))
  Mean = Cov %*% (solve(Lam0) %*% mu0 + solve(Sig0) %*% rowSums(betas))
  return( c(rmvnorm(1, Mean, Cov)) )
}
```

```
update_Sig0 = function(beta0, betas) {
  beta_scale = betas - beta0 %*% t(rep(1, m))
  Sbeta = beta_scale %*% t(beta_scale)
  # the second argument is S (Cov) not S^-1 (Pre)
  return( riwish(eta0+m, (S0+Sbeta)) )
}

update_sig2 = function(betas) {
  a = v0 + n
  b = v0 * sig0.2
  for(i in 1:m) {
    ind = (school == i) # indicator of data from each school
    Xi = X[ind,]
    yi = y[ind]
    b = b + sum((yi - Xi %*% betas[,i])^2)
  }
  return( 1/rgamma(1, a/2, b/2))
}

update_betas = function(beta0, Sig0, sig2) {
  betas = matrix(NA, nrow = p, ncol = m)
  for(i in 1:m) {
    ind = (school == i) # indicator of data from each school
    Xi = X[ind,]
    yi = y[ind]
    Cov = solve(solve(Sig0) + t(Xi) %*% Xi / sig2)
    Mean = Cov %*% (solve(Sig0) %*% beta0 + t(Xi) %*% yi / sig2)
    betas[,i] = rmvnorm(1, Mean, Cov)
  }
  return(betas)
}
```

2.2 Generate data and parameters

```
# data settings
m = 10
p = 10
nk = 1:m * 10
n = sum(nk)
school = map(1:m , function(x) rep(x, x * 10)) %>% flatten_int()

# oracle values
beta0_or = rep(0, p)
sig2_or = 1
Sig0_or = diag(1, nrow = p)

# generate beta's and data
set.seed(1)
# betas: p x m matrix, each column is a beta_j for school j
betas_or = t(rmvnorm(m, beta0_or, Sig0_or))
# X: n x p matrix, n = n_1+...+n_m (sorted as school 1 to m)
X = cbind(1,
```

```
          rmvnorm(n, rep(0, p-1),
                  0.8*diag(1, p-1) + 0.2 * rep(1, p-1)%*%t(rep(1, p-1))) )
y = rep(NA, n)
for(i in 1:m) {
  ind = (school == i)
  y[ind] = rnorm(nk[i], X[ind,] %*% betas_or[,i], sqrt(sig2_or))
}
```

2.3 Implement Gibbs sampler

To implement Gibbs sampler, first we should choose and justify priors for β_0, Σ_0^{-1} and σ^2.

For β_0, it's the center of β_j. I believe it's natural to assume all $\beta' s$ are evenly distributed around 0, and parameter β for each covariate should be independent since there is no obvious evidence for us to claim any dependence between the possible 'influence' of each covariate to the response. Therefore, we have $\mu_0 = \mathbf{0}_{p\times 1}, \Lambda_0 = I_{p\times p}$, making $\beta_0 \sim \mathcal{N}_p(\mathbf{0}, I)$

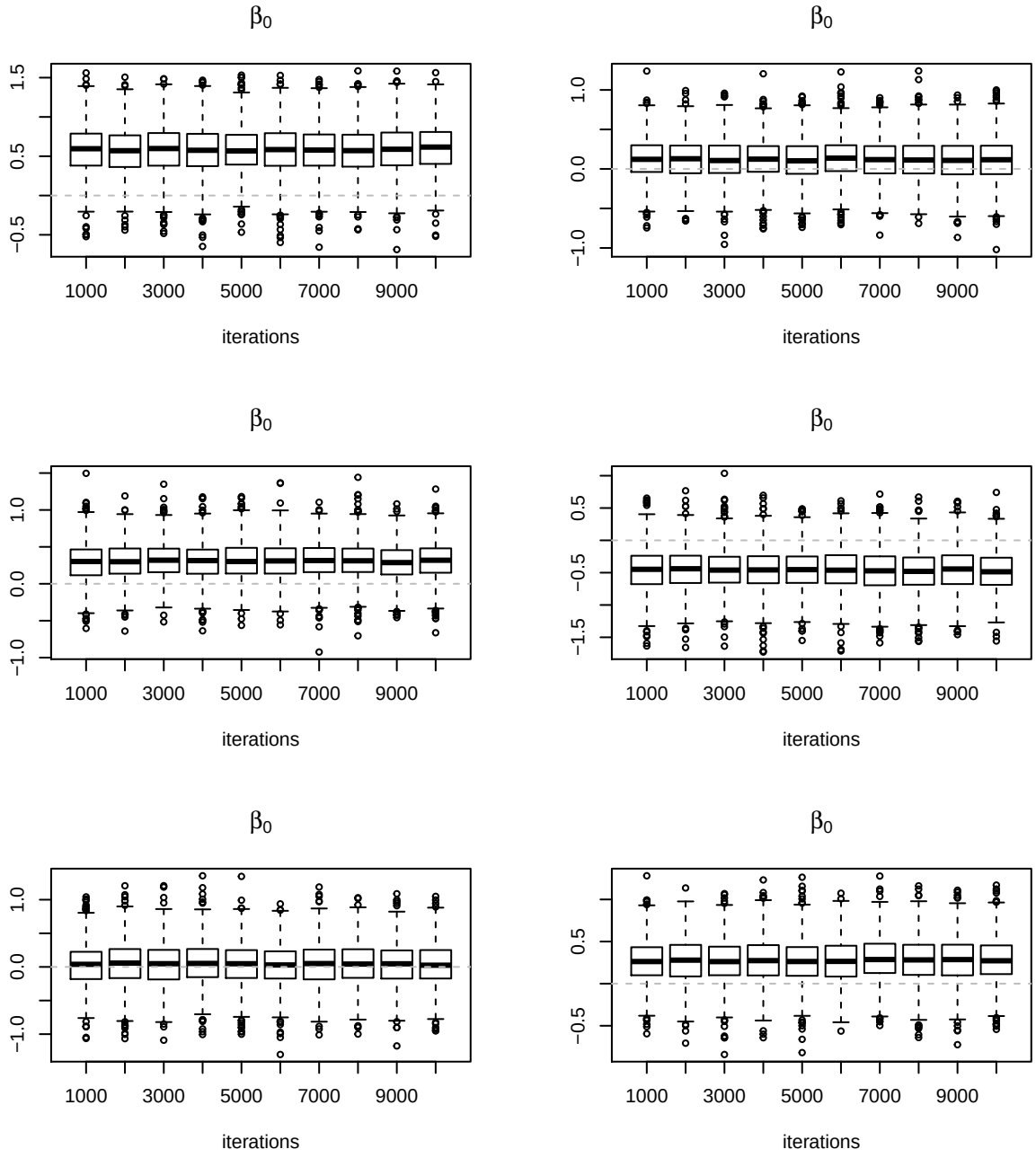
For Σ_0, it describes the variation in the $\beta_j' s$ around their center β_0. Since there is not much information, I will also let $S_0 = (\eta_0 - p - 1)I_{p\times p}$ (then the expectation of Σ_0 is $I_{p\times p}$), to capture a prior belief of relatively small and independent variation for each element of $\beta_j' s$. I'm fairly sure about my prior belief, so I set $\eta_0 = 100$, making $\Sigma_0^{-1} \sim Wishart(100, (100 - p - 1)I_{p\times p})$.

For σ^2, I believe the variantion within schools should neither be too small nor too big, so I set $\sigma_0^2 = 2.25$. But I'm not sure whether such a guess would be appropriate, so I set $\nu_0 = 2$ for my weak prior belief, thus making $\sigma^{-2} \sim Gamma(1, 2.25)$.

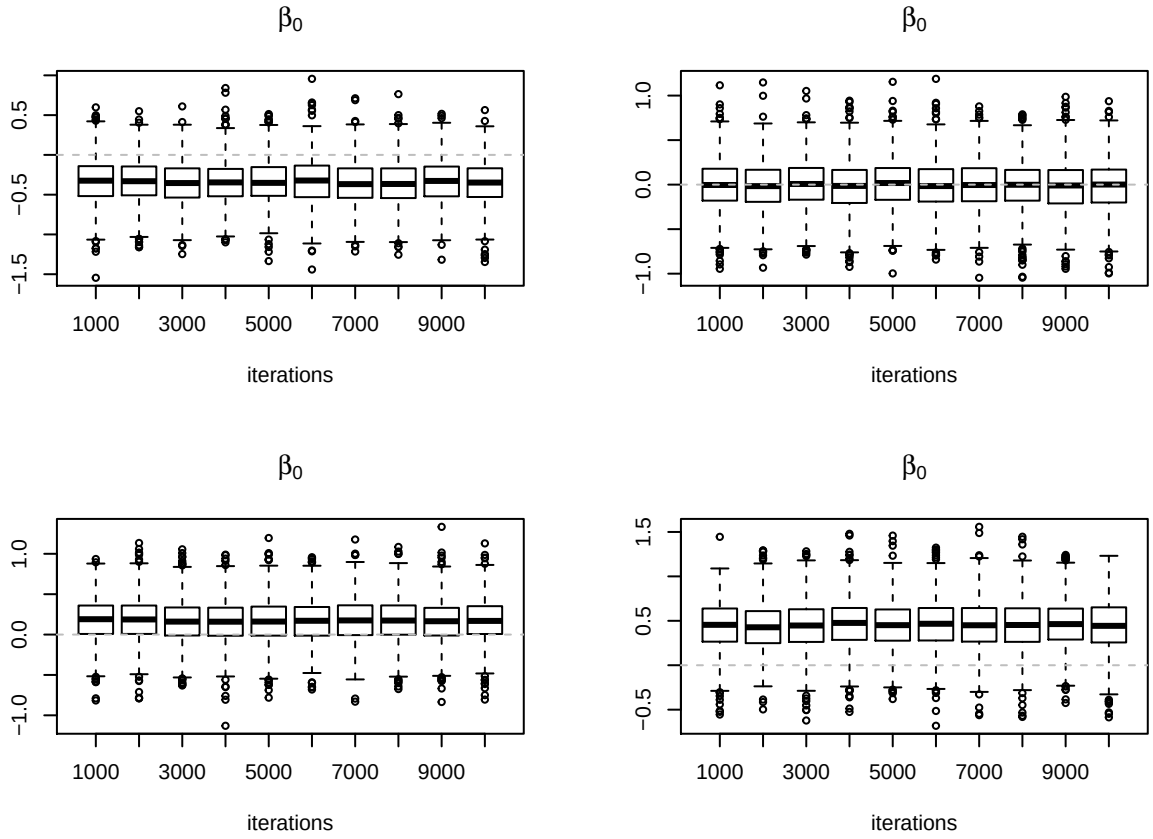
With such priors, we can implement our Gibbs sampler as follows:

```
# set hyperparameters
mu0 = rep(0, p)
Lam0 = diag(1, nrow = p)
eta0 = 20
S0 = (eta0 - p - 1) * diag(1, nrow = p)
v0 = 2
sig0.2 = 2.25

# initial values
beta0 = rep(0, p)
Sig0 = diag(1, p)
sig2 = 1
betas = matrix(0, nrow = p, ncol = m)

S = 10000
beta0.sam = array(NA, dim = c(p, 1, S))
Sig0.sam = array(NA, dim = c(p, p, S))
sig2.sam = array(NA, dim = c(1, 1, S))
betas.sam = array(NA, dim = c(p, m, S))
for(i in 1:S) {
  beta0 = update_beta0(Sig0, betas) # p x 1
  beta0.sam[,,i] = beta0
  Sig0 = update_Sig0(beta0, betas) # p x p
  Sig0.sam[,,i] = Sig0
  sig2 = update_sig2(betas) # 1 x 1
  sig2.sam[,,i] = sig2
  betas = update_betas(beta0, Sig0, sig2) # p x m
```

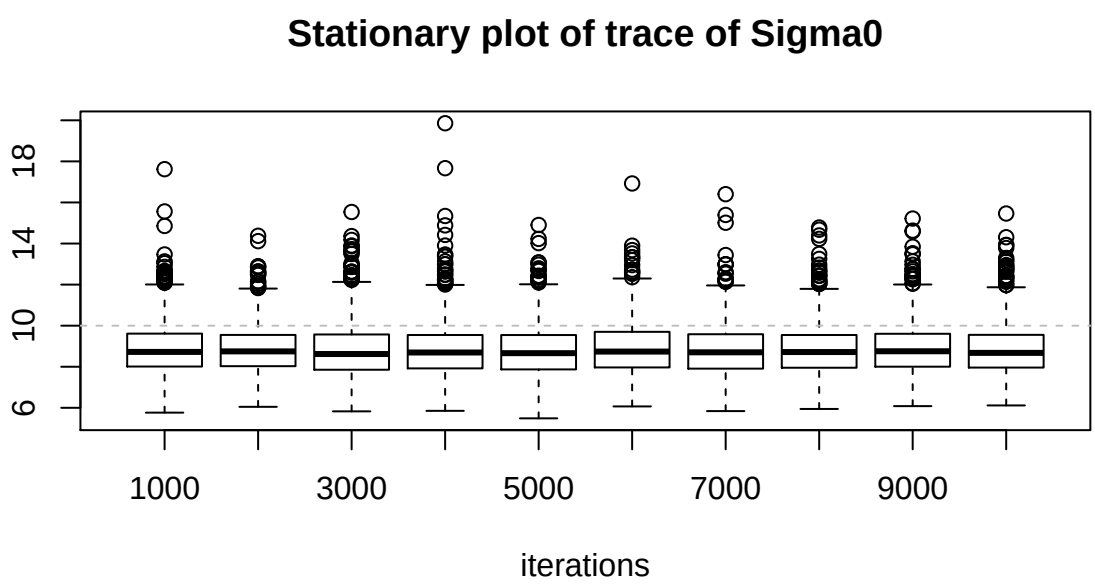
```
  betas.sam[,,i] = betas
}

# stationary plot
stationarity.plot = function(x,...){
  S = length(x)
  scan = 1:S
  ng = min( round(S/100),10)
  group = S*ceiling( ng*scan/S) /ng
  boxplot(x~group,...) }



# beta 0 (every value) vs beta0_or
par(mfrow = c(3,2))
for(i in 1:m) {
  stationarity.plot(beta0.sam[i,,], main = expression(beta[0]), xlab = 'iterations')
  abline(h = 0, col = 'gray', lty = 2)
}
```

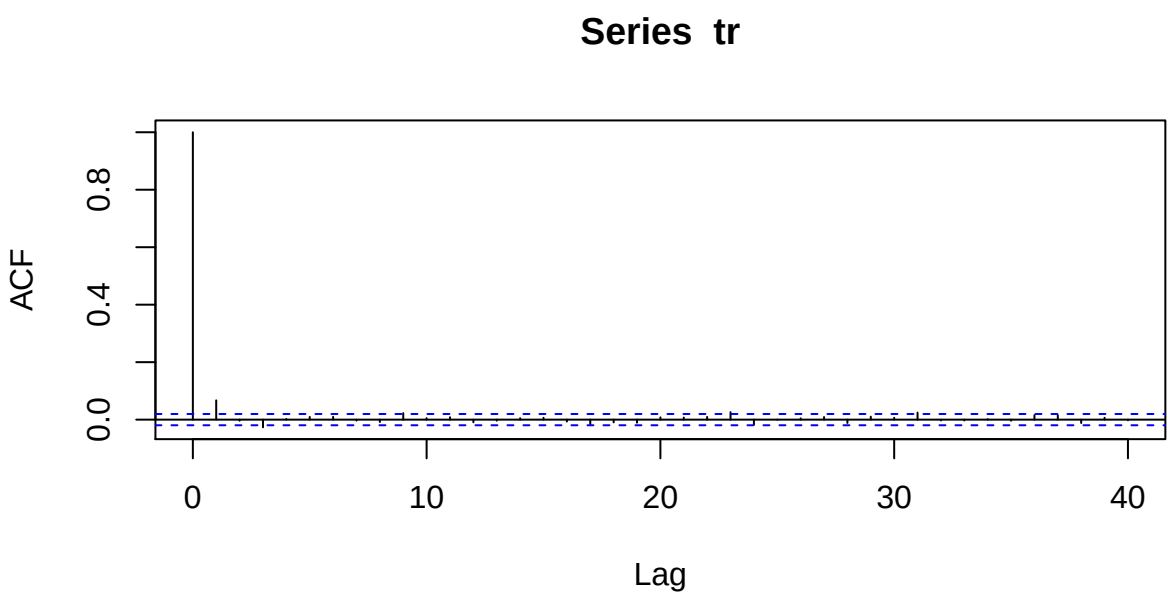



```
# trace of Sig0, Sig0 or, effectivesamplesize
par(mfrow = c(2,1))
```

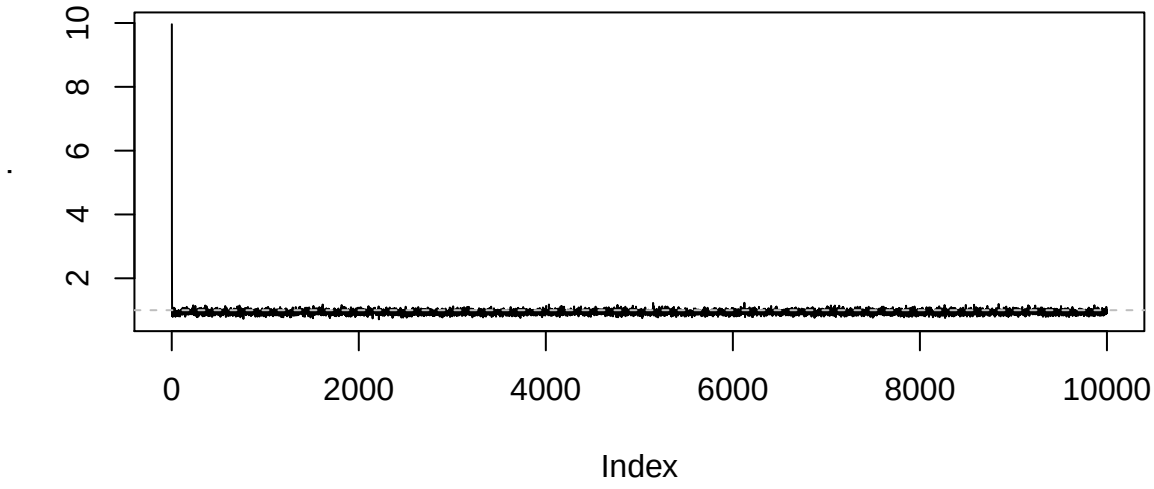



```
tr = apply(Sig0.sam, 3, function(x) diag(x) %>% sum)
stationarity.plot(tr, main = 'Stationary plot of trace of Sigma0', xlab = 'iterations')
abline(h = 10, col = 'gray', lty =2)
acf(tr)
```

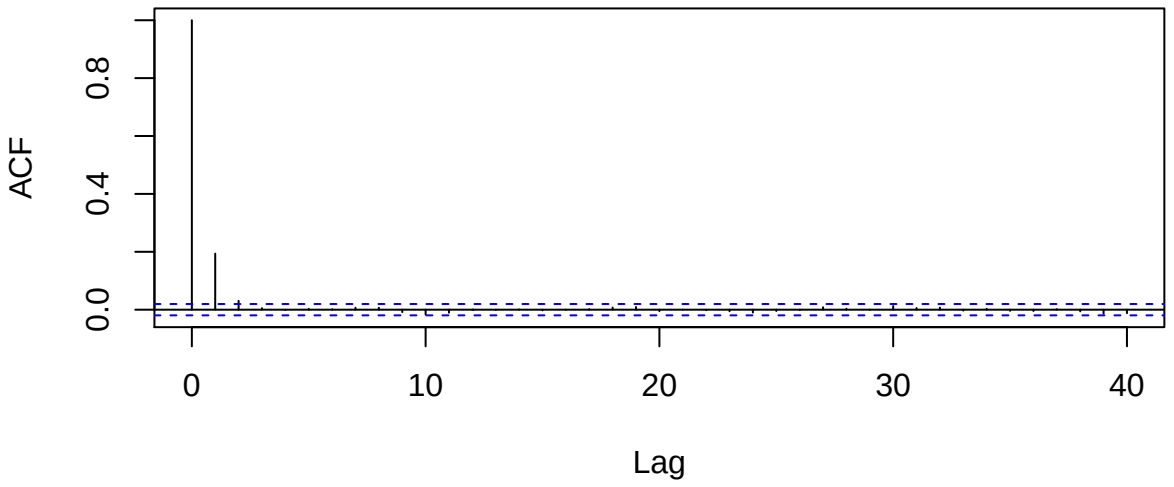




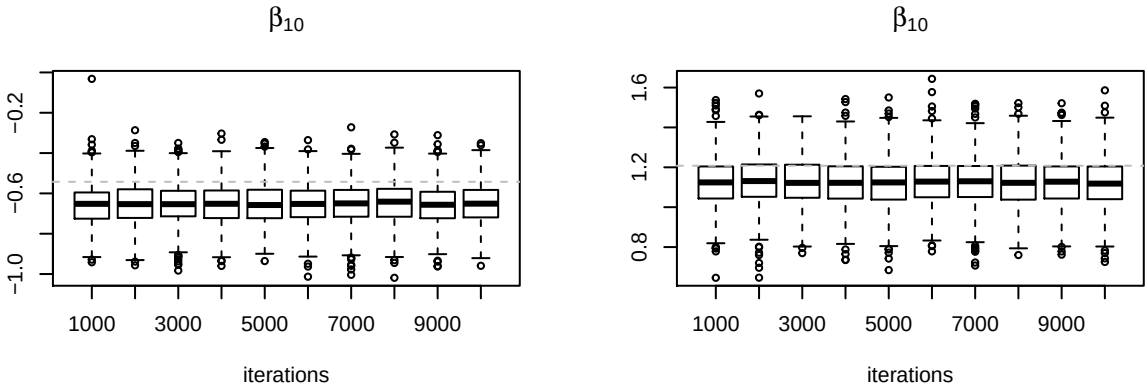

```
# eigenmax = apply(Sig0.sam, 3, function(x) eigen(x)$values  %>% max)


# sigma^2 sigma2_or
par(mfrow = c(2,1))
sig2.sam[1,,] %>% plot(type = 'l')
abline(h = 1, col = 'gray', lty =2)
acf(sig2.sam[1,,], main = 'sigma.squared')
```



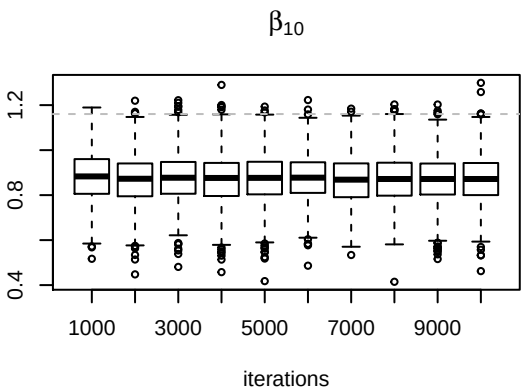
sigma.squared

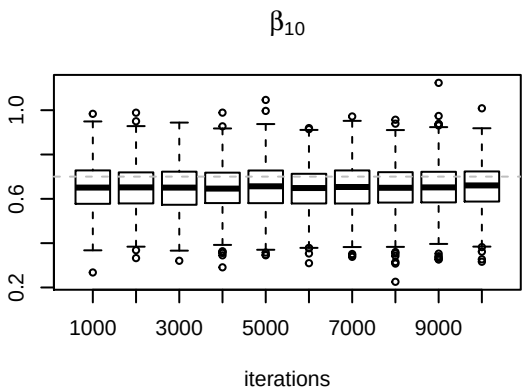


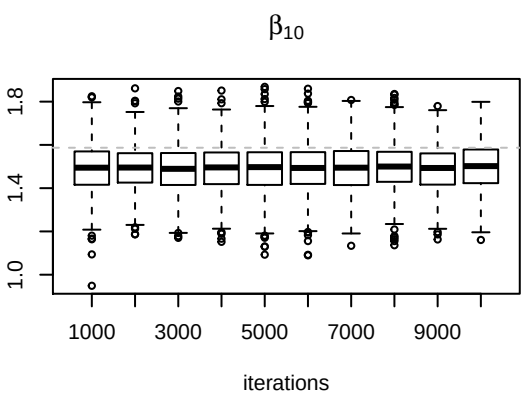

```
# beta 10 and bar beta

beta10 = t(betas.sam[,10,])
par(mfrow = c(3,2))
for(i in 1:m) {
  stationarity.plot(beta10[,i], main = expression(beta[10]), xlab = 'iterations')
  abline(h = betas_or[i,10], col = 'gray', lty = 2)
}
```

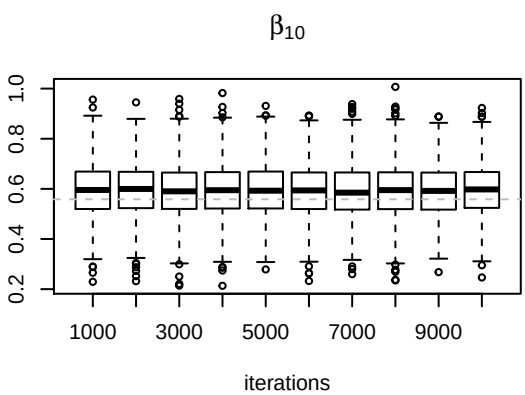

β10
−1.0
−0.6
−0.2
1000
3000
5000
7000
9000
iterations

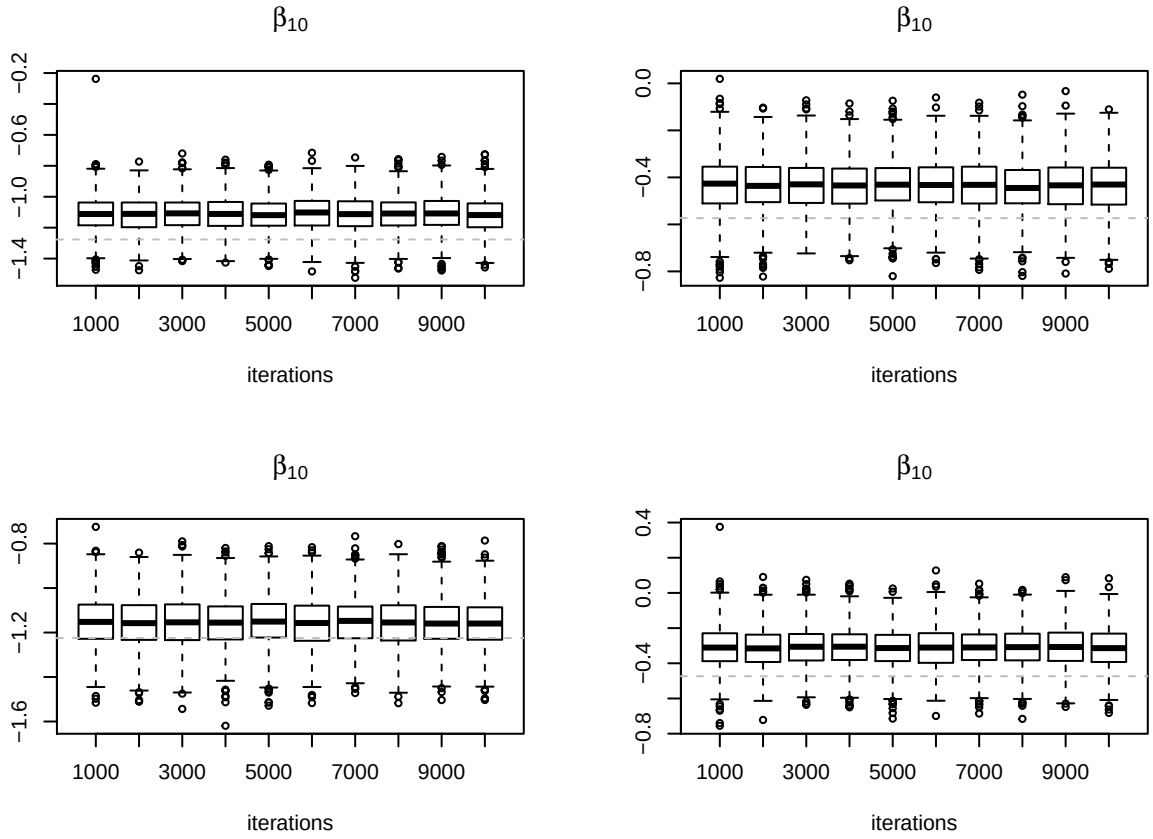
β10
0.8
1.2
1.6
1000
3000
5000
7000
9000
iterations


β10
0.4
0.8
1.2
1000
3000
5000
7000
9000
iterations


β10
0.2
0.6
1.0
1000
3000
5000
7000
9000
iterations


β10
1.0
1.4
1.8
1000
3000
5000
7000
9000
iterations


β10
0.2
0.4
0.6
0.8
1.0
1000
3000
5000
7000
9000
iterations



To evaluate the convergence of Markov Chain, I used stationary plot and traceplot to check whether our Gibbs sampler has achieved stationarity. From the stationary plot of β_0, $tr(\Sigma_0)$ and β_{10} (take β_{10} as a example since there is too many of them) and the traceplot of σ^2, we can see that there is strong evidence that our Gibbs samples of parameters have achieved stationarity or has converged.

Also, we can see that all the Markov Chains seem to have a good mixing judging from the low autocorrelation of parameters and large effective sample size (provided in later tables). From the effective sample sizes of each parameter displayed in the tables below, we know that our Gibbs sampler for this model and prior distribution performs quite well.

```
# beta 0
cbind(0, apply(beta0.sam, 1:2, mean),
      apply(beta0.sam, 1:2, function(x) quantile(x, 0.025)),
      apply(beta0.sam, 1:2, function(x) quantile(x, 0.975)),
      apply(beta0.sam, 1:2, effectiveSize)) %>%
  kable(digits = 4,
        col.names = c('True', 'Pos. Mean', '2.5%', '97.5%', 'Effective Sample Size'),
        caption = 'Inference about $\\beta_{0,k}, k =1,...,p$')
```

Table 5: Inference about $\beta_{0,k}, k = 1, ..., p$

True	Pos. Mean	2.5%	97.5%	Effective Sample Size
0	0.5803	-0.0374	1.1639	10067.309
0	0.1213	-0.4016	0.6395	8716.397
0	0.3079	-0.1995	0.8084	8458.528
0	-0.4601	-1.0954	0.1905	9190.067
0	0.0403	-0.6016	0.6728	8770.990
0	0.2751	-0.2516	0.7957	8497.278
0	-0.3406	-0.8801	0.2173	9235.560
0	-0.0061	-0.5630	0.5454	8196.812
0	0.1713	-0.3683	0.7079	8909.665
0	0.4519	-0.0958	0.9936	8649.523

```
# trace of Sig0
c(m, mean(tr),
      quantile(tr, c(0.025,0.975)),
      effectiveSize(tr)) %>%
  t %>%
  kable(digits = 4,
        col.names = c('True', 'Pos. Mean', '2.5%', '97.5%', 'Effective Sample Size'),
        caption = 'Inference about $tr(\\Sigma_0)$')
```

Table 6: Inference about $tr(\Sigma_0)$

True	Pos. Mean	2.5%	97.5%	Effective Sample Size
10	8.8596	6.7369	11.904	9361.395

```
# sigma^2
c(1, mean(sig2.sam[1,,]),
  quantile(sig2.sam[1,,], c(0.025, 0.975)),
  effectiveSize(sig2.sam[1,,])) %>%
  t %>%
  kable(digits = 4,
        col.names = c('True', 'Pos. Mean', '2.5%', '97.5%', 'Effective Sample Size'),
        caption = 'Inference about $\\sigma^2$')
```

Table 7: Inference about σ^2

True	Pos. Mean	2.5%	97.5%	Effective Sample Size
1	0.9363	0.8208	1.0649	6752.199

```
# beta10
cbind(betas_or[,10], apply(beta10, 2, mean),
      apply(beta10, 2, function(x) quantile(x, 0.025)),
      apply(beta10, 2, function(x) quantile(x, 0.975)),
      apply(beta10, 2, effectiveSize)) %>%
  kable(digits = 4,
        col.names = c('True', 'Pos. Mean', '2.5%', '97.5%', 'Effective Sample Size'),
        caption = 'Inference about $\\beta_{10,k}, k = 1,...,p$')
```

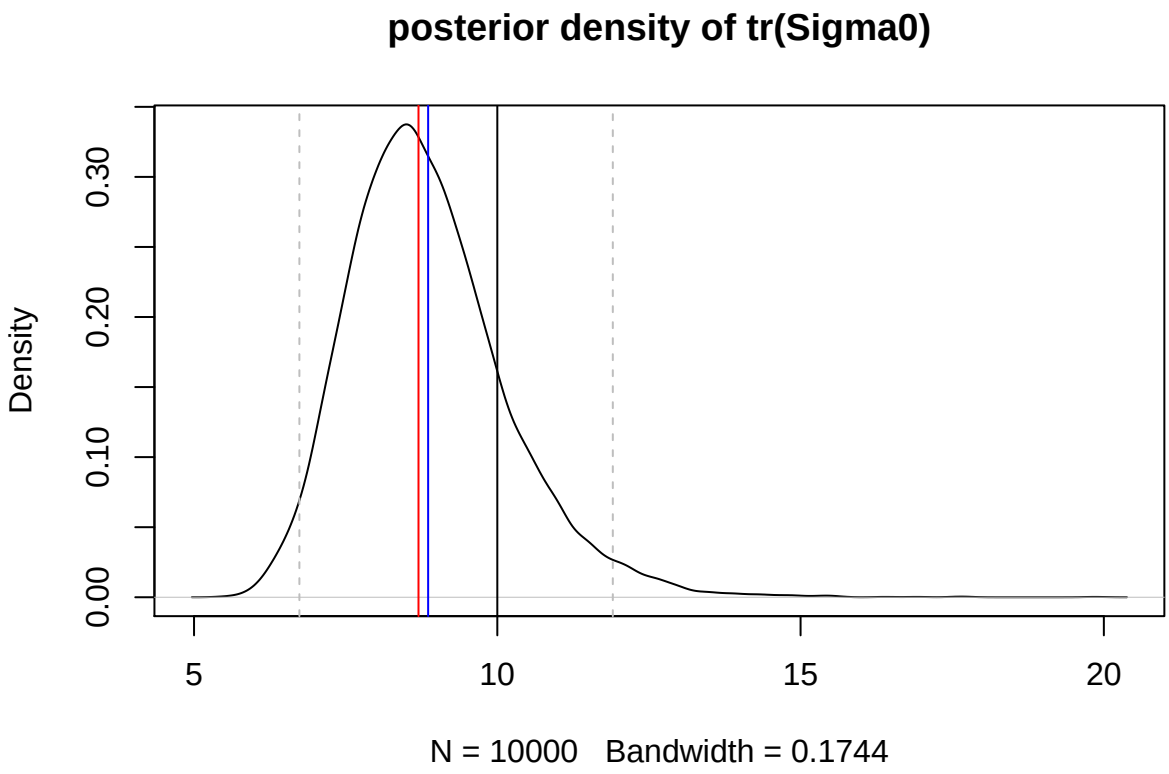
Table 8: Inference about $\beta_{10,k}, k = 1, ..., p$

True	Pos. Mean	2.5%	97.5%	Effective Sample Size
-0.5425	-0.6518	-0.8480	-0.4525	10000.000
1.2079	1.1260	0.8896	1.3665	9386.700
1.1604	0.8729	0.6602	1.0823	10126.890
0.7002	0.6514	0.4519	0.8519	9532.958
1.5868	1.4936	1.2816	1.6999	9358.494
0.5585	0.5938	0.3813	0.7983	9699.321
-1.2766	-1.1124	-1.3388	-0.8909	10000.000
-0.5733	-0.4339	-0.6531	-0.2165	9636.275
-1.2246	-1.1557	-1.3749	-0.9372	10000.000
-0.4734	-0.3101	-0.5395	-0.0771	11568.060

The the summaries of inference about the posterior distribution for each parameter (or summary statistic of parameters) are provided as above. We can see that the posterior confidence intervals all include the true values. Also the posterior means of each parameter seem to perform well in estimating the true (oracle) values.

I choose to draw the density plot for 1 summary statistics since there are too many of them.

```
plot(density(tr), main = 'posterior density of tr(Sigma0)')
abline(v = 10, col = 1)
abline(v = quantile(tr, c(0.025, 0.975)), col = 'gray', lty =2)
abline(v = quantile(tr, c(0.5)), col = 'red')
abline(v = mean(tr), col = 'blue')
```



As is shown in the posterior distribution of $tr(\Sigma_0)$, the true value (black) is included in the confidence interval (gray) and fairly close to the posterior mean (blue) and median (red).