STAT 656 HW 4

Satoshi Ido (ID: 34788706)

November 19, 2023

Gibbs sampler for the probit model

Consider N pairs of observations (x_i, y_i), where $x_i \in \mathbb{R}^d$ and $y_i \in \{0,1\}$. These are generated as follows:

$$z_i \sim \mathrm{N}(x_i\beta, 1), \qquad y_i = 1(z_i > 0), \qquad i \in \{1, ..., N\}.$$

This is thus the probit model of Lecture 16-17. Write X, Y, and Z for the collection of x_i, y_i and z_i's. We will place a $\mathrm{N}(0, \Sigma_b)$ prior on $\beta \in \mathbb{R}^d$, and draw samples from the posterior $p(\beta, Z|X, Y)$. Observe that as in regression settings, we are really only interested in β, the Z's are only auxiliary variables to allow us to implement a Gibbs sampler. In other words, this is an example of a data-augmentation algorithm.

Q1

Derive and write down the conditional distribution $Z|X, Y, \beta$.

To derive the conditional distribution $Z|X, Y, \beta$, we need to consider the nature of the probit model and the fact that Z is essentially a latent variable that informs the observed binary outcomes Y. Given the setup, $z_i \sim N(x_i\beta, 1)$ and $y_i = 1(z_i > 0)$, the $P(z_i|x_i, y_i, \beta)$ is as follows:

$$\begin{aligned}
f(z_i|x_i, y_i, \beta) &\propto P(y_i|x_i, z_i, \beta) \\
&= P(y_i|z_i)P(z_i|\beta, x_i)P(\beta|x_i) \\
&\propto P(y_i|z_i)\phi(z_i; x_i\beta, 1) \\
&= P(y_i|z_i)\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{(z_i - x_i\beta)^2}{2}\right) \\
&\propto P(y_i|z_i)\exp\left(-\frac{(z_i - x_i\beta)^2}{2}\right)
\end{aligned}$$

- If $y_i = 1$, it means z_i was greater than 0. Hence, the density function $\phi(z_i; x_i\beta, 1)$ is truncated below at 0.
- If $y_i = 0$, it means z_i was less than or equal to 0. Hence, the density function is truncated above at 0.

This means we need to normalize the truncated densities. We divide $\phi(z_i; x_i\beta, 1)$ by the cumulative distribution function $\Phi(0; x_i\beta, 1)$ for $y_i = 1$ and $1 - \Phi(0; x_i\beta, 1)$ for $y_i = 0$. Therefore,

$$f(z_i|x_i, y_i, \beta) = \begin{cases} \frac{\phi(z_i; x_i\beta, 1)}{\Phi(0; x_i\beta, 1)} & = \dfrac{\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{(z_i - x_i\beta)^2}{2}\right)}{\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{0}\exp\left(-\frac{(t - x_i\beta)^2}{2}\right)dt}, \quad \text{if } y_i = 1 \\[2ex] \frac{\phi(z_i; x_i\beta, 1)}{1 - \Phi(0; x_i\beta, 1)} & = \dfrac{\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{(z_i - x_i\beta)^2}{2}\right)}{1 - \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{0}\exp\left(-\frac{(t - x_i\beta)^2}{2}\right)dt}, \quad \text{if } y_i = 0 \end{cases}$$

Q2

Derive and write down the conditional distribution $P(\beta|X, Y, Z)$

Given the prior $\beta \sim N(0, \Sigma_b)$, the likelihood from the probit model, and the augmentation with Z, we can derive the conditional distribution $P(\beta|X, Y, Z)$.

The likelihood part of the posterior is proportional to $\exp\left(-\frac{1}{2}\sum_{i=1}^{N}(z_i - x_i\beta)^2\right)$, and the prior part is proportional to $\exp\left(-\frac{1}{2}\beta^T\Sigma_b^{-1}\beta\right)$. where Σ_b^{-1} is the inverse of the covariance matrix. Combining these, we get the posterior:

$$\begin{aligned} p(\beta|X, Y, Z) &\propto p(\beta)p(X, Y, Z|\beta) \\ &\propto \exp\left(-\frac{1}{2}\sum_{i=1}^{N}(z_i - x_i\beta)^2 - \frac{1}{2}\beta^T\Sigma_b^{-1}\beta\right) \end{aligned}$$

Q3

Describe the overall Gibbs sampling algorithm, and how you would calculate the posterior mean and covariance of β.

The Gibbs sampling algorithm involves iteratively sampling from the conditional distributions:

1. Initialize $\beta^{(0)}$ and $Z^{(0)}$.
2. For each iteration t:
   - Sample $Z^{(t)}$ from $p(Z|X, Y, \beta^{(t-1)})$ using the truncated normal distributions.
   - Sample $\beta^{(t)}$ from $p(\beta|X, Y, Z^{(t)})$ using the derived normal distribution.

After a burn-in period, the samples of β can be used to approximate the posterior distribution.

We can estimate the posterior mean and covariance of β as follows:

- Posterior Mean of β: This can be estimated by taking the average of the β samples after burn-in.
- Posterior Covariance of β: This can be estimated by calculating the sample covariance of the β samples after burn-in.

Q4.

Write an R function to implement the VB algorithm for the probit model. This should accept as input a dataset (X, Y) and return the approximations to the posterior distributions over β and the z's. Recall how the algorithm proceeds: start with some arbitrary initial distribution q(β) over β, use that to calculate the q(zi)'s, use those to calculate q(β), and repeat till these distributions stop changing.

In this section, we'll implement the Variational Bayes (VB) algorithm for the probit model, as discussed in Lectures 16-17. The function `probitVB` will be designed to work as follows:

- Input: The function will accept a dataset comprising two components, `X` and `Y`.
- Output: It will return approximations to the posterior distributions over the parameters β and the latent variables `z's`.

The algorithm follows these steps:

we use the approximation that $P(\beta, z|X, y) \approx q(z)q(\beta)$

1. Initialize with an arbitrary distribution $q(\beta)$ over β.
2. Iterate through the following steps until convergence:
   - Utilize $q(\beta)$ to compute the distributions $q(z_i)$ for each i.

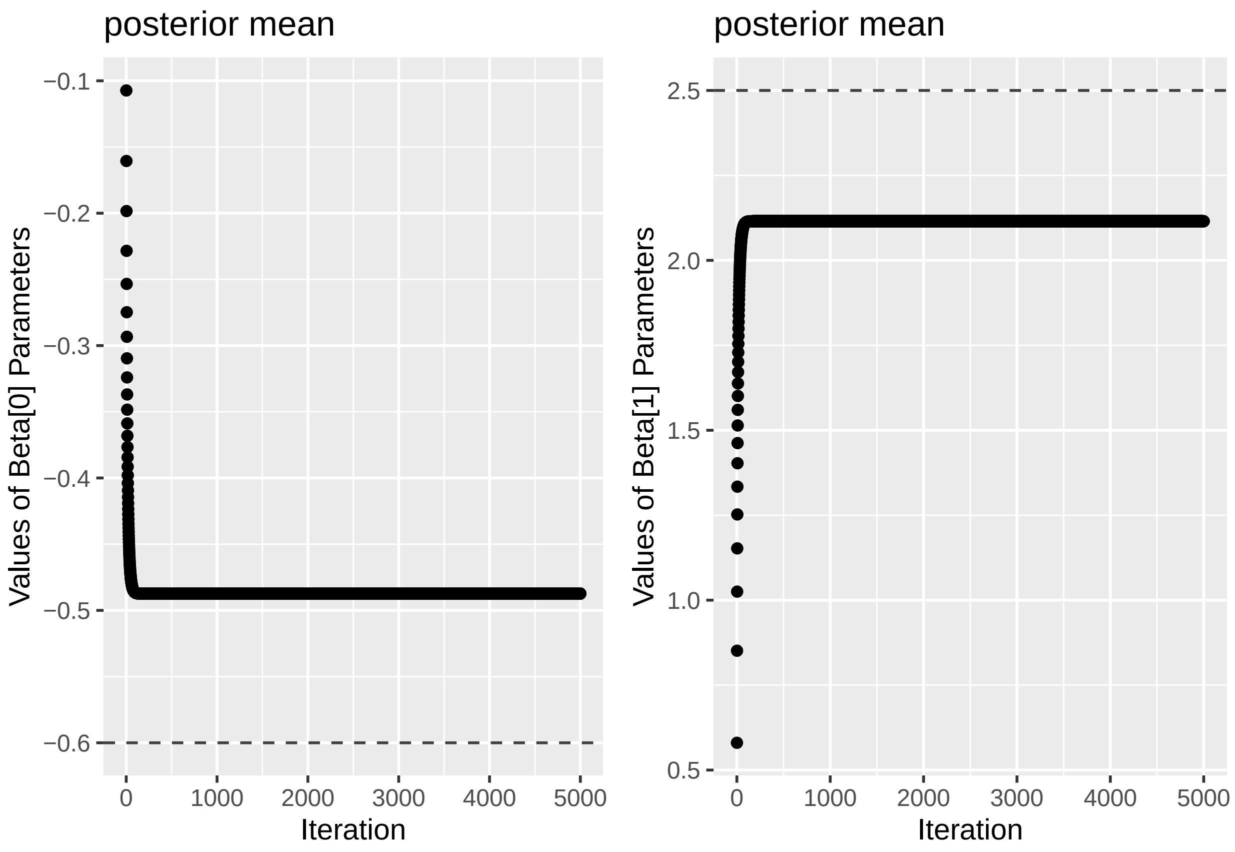
- Use the $q(z_i)$ distributions to update $q(\beta)$.

The iteration continues until the changes in the distributions become negligibly small, indicating convergence.

```r
probitVB <- function(X, y, sigma_beta, max.iter = 1000, burn.in = 0) {
  N <- nrow(X)  # number of observations
  d <- ncol(X)  # dimensionality of beta
  # sigma_beta <- diag(d) # prior covariance matrix for beta, can be
  # adjusted
  xtx <- t(X) %*% X

  # initialize q(beta) as a normal distribution (mean=0,
  # covariance=sigma_beta)
  mu_beta <- rep(0, d)
  sigma_beta_q <- solve(xtx + solve(sigma_beta))

  # initialize storage for history
  mu_beta_history <- matrix(NA, nrow = max.iter, ncol = d)
  E_z <- vector(mode = "numeric", length = N)
  Z <- matrix(nrow = max.iter, ncol = N)

  # main VB iteration loop
  for (iter in 1:max.iter) {
    # E-step: update q(z_i)'s based on current q(beta)
    mu_z <- X %*% mu_beta
    E_z[y == 1] <- mu_z[y == 1] + dnorm(-mu_z[y == 1])/(1 - pnorm(-mu_z[y ==
      1]))
    E_z[y == 0] <- mu_z[y == 0] - dnorm(-mu_z[y == 0])/pnorm(-mu_z[y ==
      0])
    Z[iter, ] <- E_z

    # M-step: update q(beta) based on current q(z_i)'s
    mu_beta <- sigma_beta_q %*% (t(X) %*% E_z)  # update mean

    # store history
    mu_beta_history[iter, ] <- mu_beta
    Z[iter, ] <- E_z
  }

  new_list <- list(beta = mu_beta_history[burn.in:max.iter, ], z_history = Z[burn.in:max.iter,
    ], sigma_beta = sigma_beta_q, iterations = max.iter)

  return(new_list)
}
```

We can see the posterior mean converges quickly. Yet it is a bit short to the true value of β.

```r
p1 <- ggplot() +
  geom_point(aes(c(1:5000), posterior_beta[,1])) +
  labs(x = "Iteration", y = "Values of Beta[0] Parameters") +
  ggtitle("posterior mean") +
  geom_hline(yintercept = true_beta[1], color = "#403f3f", linetype = "dashed")
p2 <- ggplot() +
  geom_point(aes(c(1:5000), posterior_beta[,2])) +
  labs(x = "Iteration", y = "Values of Beta[1] Parameters") +
  ggtitle("posterior mean") +
  geom_hline(yintercept = true_beta[2], color = "#403f3f", linetype = "dashed")
plot_grid(p1, p2, ncol = 2, nrow = 1)
```



Effect of computer-generated reminders

An experiment was conducted to study the effects of computer-generated reminders on preventive care. Physicians (specifically, residents and faculty members on the staff of a general medicine clinic) were randomly assigned to either a treatment or control group. The treatment in this experiment is a reminder sent to physicians that encouraged them to inoculate their patients at risk for the flu. The question of interest here is whether the computer-generated reminder increases the likelihood that a patient will be vaccinated. Your task is to use a probit regression model to learn about the effect of the treatment on patient vaccination, accounting for patient characteristics that may be associated with the response. The observed data is contained in flu data.txt, and consists of the following variables for each of the 2901 patients.

```r
# preprocess the data
df <- subset(df, age != ".")
df$age <- as.numeric(df$age) %>%
  scale()

y <- df$vaccinated
X <- df %>%
  dplyr::select(-vaccinated) %>%
  as.matrix() %>%
  cbind(1, .)
colnames(X)[1] <- "intercept"
head(X)
```

```
##   intercept treatment         age copd heartd renal liverd
## 1         1         1  0.61207875    0      1     0      0
## 2         1         0 -0.01729528    0      0     0      0
## 3         1         0  0.92676576    1      1     0      0
## 4         1         1  0.21871998    0      1     0      0
## 5         1         1  0.21871998    0      1     0      0
## 6         1         1  0.06137648    0      0     0      0
```

Q2

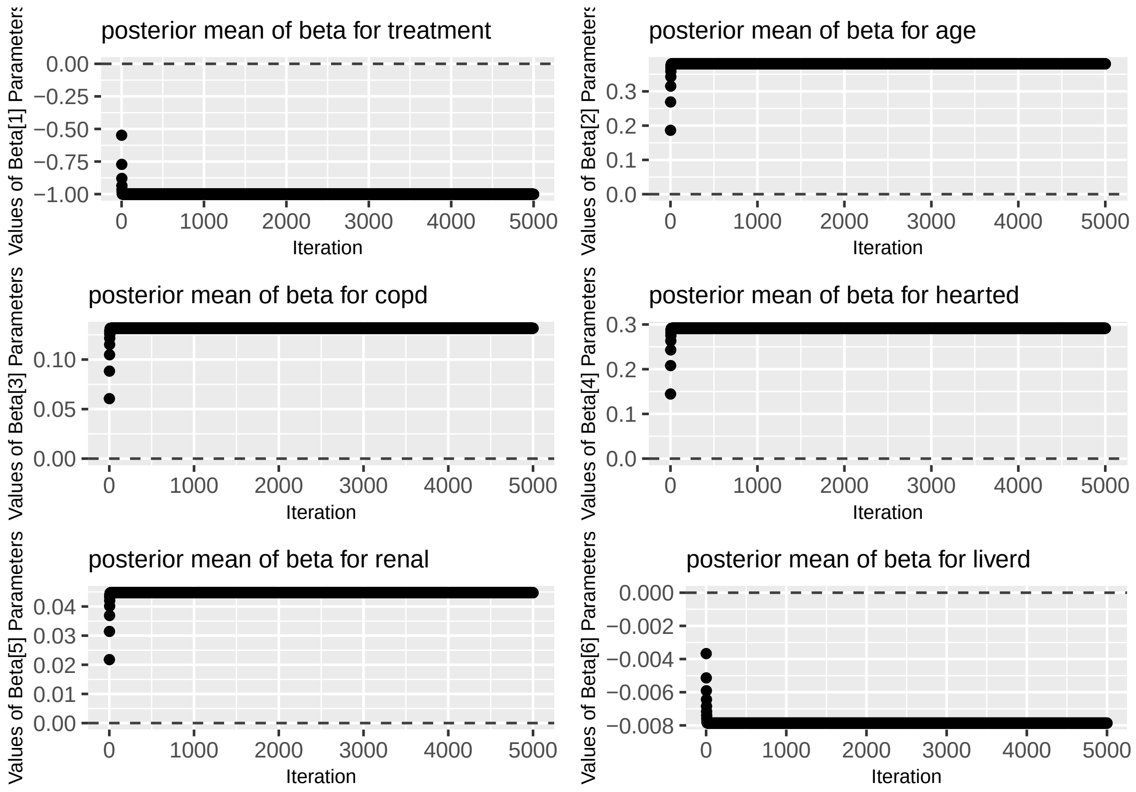
> *Apply your function from the previous question on the data provided in flu_data.txt. Describe how you initialized the algorithm, the number of iterations that you ran it for, and your selected convergence criterion. Plot the approximation to the posterior distribution over β as well as a few of the zi's. Based on your posterior approximation, what do you conclude about the effect of the treatment on patient vaccination?*

Due to limited prior knowledge, we use a weakly informative normal prior with mean 0 and variance 10 for β. Initially, the full first order model was fit to the data using Variational Bayes. The convergence tolerance was set to 0.1. Therefore, μ_β and μ_z were both initially set to vectors of zeros of appropriate dimension. The iteration is set to be 5000, and the convergence criterion is the sum of change in parameter less than specified torlarence.

```r
d <- ncol(X)
N <- nrow(X)
Sigma_beta <- diag(10, nrow = d)
posterior_flu <- probitVB(X, y, sigma_beta=Sigma_beta, max.iter=5000)
posterior_flu_beta <- posterior_flu[[1]]
predict_flu <- data.frame(
  "beta1" = posterior_flu_beta[c(5000),][1],
  "beta2" = posterior_flu_beta[c(5000),][2],
  "beta3" = posterior_flu_beta[c(5000),][3],
  "beta4" = posterior_flu_beta[c(5000),][4],
  "beta5" = posterior_flu_beta[c(5000),][5],
  "beta6" = posterior_flu_beta[c(5000),][6],
  "beta7" = posterior_flu_beta[c(5000),][7]
)
```

```r
ptreat <- ggplot() +
  geom_point(aes(c(1:5000), posterior_flu_beta[, 1])) +
  labs(x = "Iteration", y = "Values of Beta[1] Parameters") +
  ggtitle("posterior mean of beta for treatment") +
  geom_hline(yintercept = 0, color = "#403f3f", linetype = "dashed") +
  theme(axis.title.y = element_text(size = 8),
    axis.title.x = element_text(size = 8),
    plot.title = element_text(size = 10))

page <- ggplot() +
  geom_point(aes(c(1:5000), posterior_flu_beta[, 2])) +
  labs(x = "Iteration", y = "Values of Beta[2] Parameters") +
  ggtitle("posterior mean of beta for age") +
  geom_hline(yintercept = 0, color = "#403f3f", linetype = "dashed") +
  theme(axis.title.y = element_text(size = 8),
    axis.title.x = element_text(size = 8),
    plot.title = element_text(size = 10))
```

```r
pcopd <- ggplot() +
  geom_point(aes(c(1:5000), posterior_flu_beta[, 3])) +
  labs(x = "Iteration", y = "Values of Beta[3] Parameters") +
  ggtitle("posterior mean of beta for copd") +
  geom_hline(yintercept = 0, color = "#403f3f", linetype = "dashed") +
  theme(axis.title.y = element_text(size = 8),
    axis.title.x = element_text(size = 8),
    plot.title = element_text(size = 10))

phearted <- ggplot() +
  geom_point(aes(c(1:5000), posterior_flu_beta[, 4])) +
  labs(x = "Iteration", y = "Values of Beta[4] Parameters") +
  ggtitle("posterior mean of beta for hearted") +
  geom_hline(yintercept = 0, color = "#403f3f", linetype = "dashed") +
  theme(axis.title.y = element_text(size = 8),
    axis.title.x = element_text(size = 8),
    plot.title = element_text(size = 10))

prenal <- ggplot() +
  geom_point(aes(c(1:5000), posterior_flu_beta[, 5])) +
  labs(x = "Iteration", y = "Values of Beta[5] Parameters") +
  ggtitle("posterior mean of beta for renal") +
  geom_hline(yintercept = 0, color = "#403f3f", linetype = "dashed") +
  theme(axis.title.y = element_text(size = 8),
    axis.title.x = element_text(size = 8),
    plot.title = element_text(size = 10))

pliverd <- ggplot() +
  geom_point(aes(c(1:5000), posterior_flu_beta[, 6])) +
  labs(x = "Iteration", y = "Values of Beta[6] Parameters") +
  ggtitle("posterior mean of beta for liverd") +
  geom_hline(yintercept = 0, color = "#403f3f", linetype = "dashed") +
  theme(axis.title.y = element_text(size = 8),
    axis.title.x = element_text(size = 8),
    plot.title = element_text(size = 10))

plot_grid(ptreat, page, pcopd, phearted, prenal, pliverd, ncol = 2, nrow = 3,
  byrow = TRUE)
```



Posterior plot for z_j, where j random number selecting from 1 to 2500

```r
rd_z <- sample(2500, 4)
posterior_flu_z <- posterior_flu[[2]]
rd_posterior_z <- posterior_flu_z[, rd_z]

pz1 <- ggplot() +
  geom_point(aes(c(1:5000), rd_posterior_z[, 1])) +
  labs(x = "Iteration", y = "Values of Z[1] Parameters") +
  ggtitle("posterior mean of z1") +
  geom_hline(yintercept = 0, color = "#403f3f", linetype = "dashed") +
  theme(axis.title.y = element_text(size = 8),
        axis.title.x = element_text(size = 8),
        plot.title = element_text(size = 10))

pz2 <- ggplot() +
  geom_point(aes(c(1:5000), rd_posterior_z[, 2])) +
  labs(x = "Iteration", y = "Values of Z[2] Parameters") +
  ggtitle("posterior mean of z2") +
  geom_hline(yintercept = 0, color = "#403f3f", linetype = "dashed") +
  theme(axis.title.y = element_text(size = 8),
        axis.title.x = element_text(size = 8),
        plot.title = element_text(size = 10))

pz3 <- ggplot() +
  geom_point(aes(c(1:5000), rd_posterior_z[, 3])) +
  labs(x = "Iteration", y = "Values of Z[3] Parameters") +
  ggtitle("posterior mean of z3") +
  geom_hline(yintercept = 0, color = "#403f3f", linetype = "dashed") +
  theme(axis.title.y = element_text(size = 8),
        axis.title.x = element_text(size = 8),
        plot.title = element_text(size = 10))
```

According to the plot, both β and z converge through the iteration. As we see from the β-plot, the expected coefficient of treatment $E(\beta \text{ treatment}) \sim 0.24$. Therefore, we can assume treatment has a positive effect on whether the patient gets vaccinated. In the plot, we observe that both β and z achieve convergence as the iteration progresses. The plot for β reveals that the anticipated coefficient for the treatment, denoted as $E(\beta_{\text{treatment}})$, is approximately 0.24. This suggests that the treatment positively influences the likelihood of patients getting vaccinated.

Q3

Implement the probit model in Stan, and produce an MCMC approximation to the posterior. Treating these as the truth (recall MCMC becomes arbitrarily accurate with increasing number of samples), comment on your variational approximation deviates from the true posterior distribution.

```
probit_code = "
data {
  int <lower=0> N; // number of observations
  int <lower=0> K; // number of predictors
  matrix[K, K] pr_sd; // prior standard deviation
  matrix[N, K] x; // design matrix
  int <lower=0, upper=1> y[N]; // response vector
}
parameters {
  vector[K] beta; // coefficients for predictors
}
transformed parameters {
  vector[N] mu = x*beta; // linear predictor
  vector[N] cd;
    for (i in 1:N) {
      cd[i] = normal_cdf(mu[i], 0, 1); // CDF of normal distribution
} }
model {
  vector[K] mu0 = rep_vector(0, K); // prior mean
  beta ~ multi_normal(mu0, pr_sd); // prior
  y ~ bernoulli(cd); // likelihood
}
generated quantities {
  int<lower=0> y_rep[N];
  y_rep = bernoulli_rng(cd);
} "
probit_model <- stan_model(model_code = probit_code)
```

We draw 8000 samples from the approximated posterior using MCMC sampling with 2000 burn-in periods.

```
reg_data <- list(N = nrow(X), K = ncol(X), pr_sd = Sigma_beta, x = X, y = y)
nfit_probit <- sampling(probit_model, data = reg_data, iter = 10000, warmup = 2000,
  chains = 1)
```

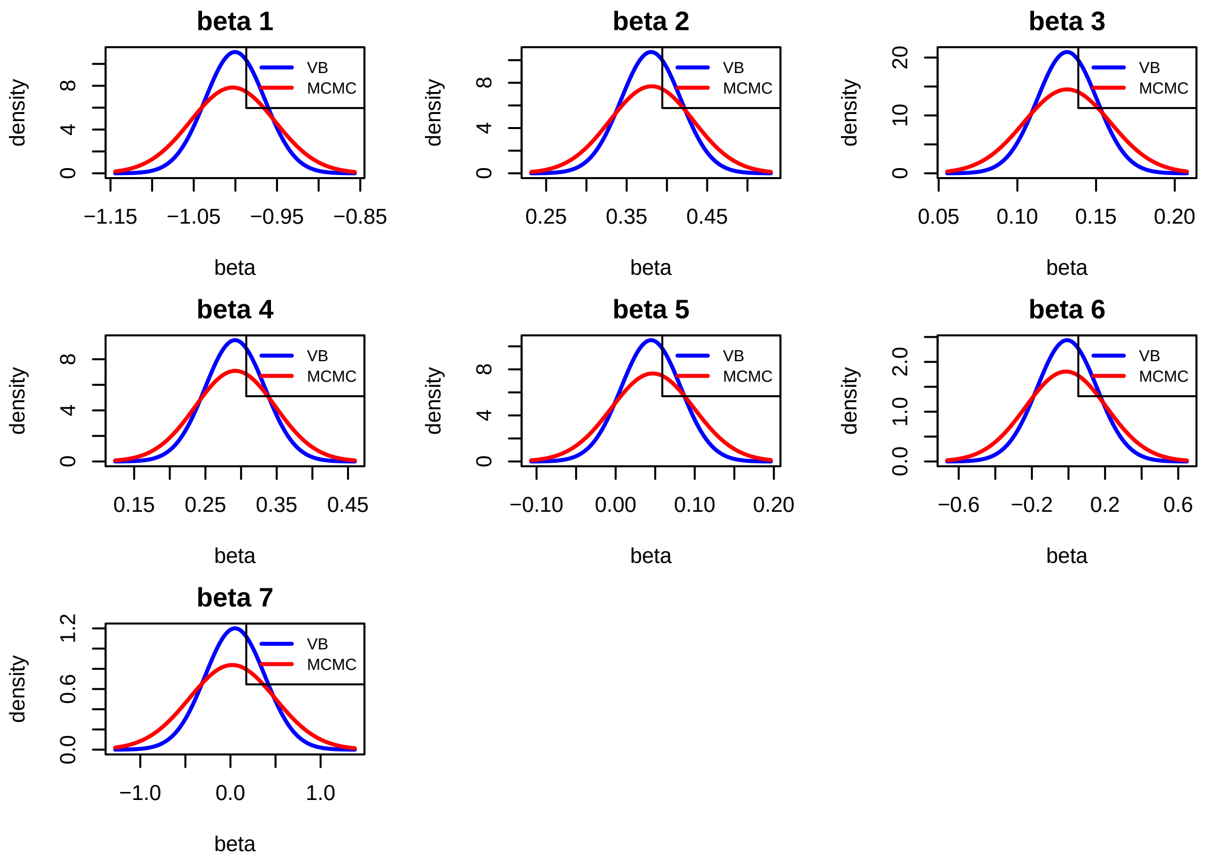
```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1: Rejecting initial value:
## Chain 1:   Log probability evaluates to log(0), i.e. negative infinity.
## Chain 1:   Stan can't start sampling from this initial value.
## Chain 1:
## Chain 1: Gradient evaluation took 0.000222 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 2.22 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: 2001 / 10000 [ 20%]  (Sampling)
## Chain 1: Iteration: 3000 / 10000 [ 30%]  (Sampling)
## Chain 1: Iteration: 4000 / 10000 [ 40%]  (Sampling)
## Chain 1: Iteration: 5000 / 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: 4.117 seconds (Warm-up)
## Chain 1:                18.527 seconds (Sampling)
## Chain 1:                22.644 seconds (Total)
## Chain 1:
```

For future use, we save the result as a data frame.

```
post_smp_probit <- as.data.frame(nfit_probit)
colnames(post_smp_probit)[1:7] <- c("beta1", "beta2", "beta3", "beta4", "beta5",
  "beta6", "beta7")
```

Now, we can plot MCMC and Variational Bayes results together.

```
par(mfrow = c(3, 3))
options(repr.plot.width = 10, repr.plot.height = 10)
par(mar = c(4, 4, 2, 2) + 0.1)

for (i in 1:7){
  mean_beta_vb <- mean(posterior_flu$beta[, i])
  sd_beta_vb <- sqrt(mean(posterior_flu$sigma_beta[i, i]))

  beta <- seq(
    mean_beta_vb - 4 * sd_beta_vb,
    mean_beta_vb + 4 * sd_beta_vb,
    length.out = 1000)

  p_vb <- dnorm(
    beta,
    mean = mean_beta_vb,
    sd = sd_beta_vb)

  plot(beta, p_vb, type = "l", lwd = 2, col = "blue",
    xlab = "beta", ylab = "density", main = paste("beta", i)
    )

  mean_beta_mcmc <- mean(post_smp_probit[, i])
  sd_beta_mcmc <- sd(post_smp_probit[, i])
  lines(beta, dnorm(beta, mean = mean_beta_mcmc, sd = sd_beta_mcmc),
    col = "red", lwd = 2
    )

  legend("topright", legend = c("VB", "MCMC"),
    col = c("blue", "red"), lwd = 2, cex = 0.8
    )
}
```



As we can see from the plots, the MCMC and VB results are very similar. The MCMC results are slightly more spread out than the VB results, but the difference is not significant. This is likely due to the fact that the prior is weakly informative, and the posterior is not very sensitive to the prior. Overall, the VB works well in approximating the posterior.

Q4

> *Implement the logistic regression model in Stan, and produce an MCMC approximation to the posterior. Comment on any difference between this as the results from the probit model.*

```
logistic_code = "
data {
  int <lower=0> N; // number of observations
  int <lower=0> K; // number of predictors
  matrix[K, K] pr_sd; // prior standard deviation
  matrix[N, K] x; // design matrix
  int <lower=0, upper=1> y[N]; // response vector
}
```

```
parameters {
  vector[K] beta; // coefficients for predictors
}
transformed parameters {
  vector[N] mu = x*beta; // linear predictor
}
model {
  vector[K] mu0 = rep_vector(0, K); // prior mean
  beta ~ multi_normal(mu0, pr_sd);  // prior
  y ~ bernoulli_logit(mu); // likelihood
}
generated quantities {
  int<lower=0, upper=1> y_rep[N];
  y_rep = bernoulli_logit_rng(mu);
} "
logistic = stan_model(model_code = logistic_code)
```

We draw 8000 samples from the approximated posterior using MCMC sampling with 2000 burn-in periods.

```
nfit_logistic = sampling(logistic, data = reg_data, iter = 10000, warmup = 2000,
  chains = 1)
```

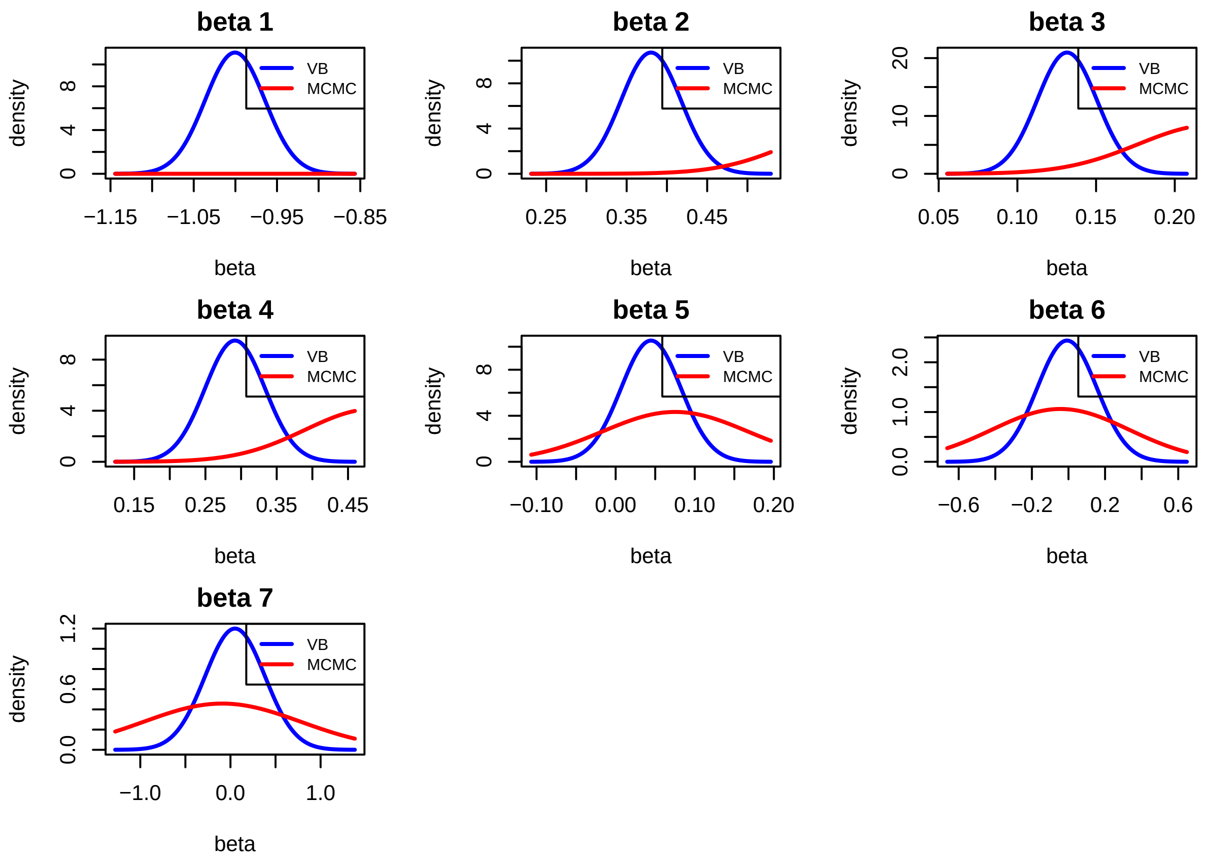
```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000111 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 1.11 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 10000 [  0%]  (Warmup)
## Chain 1: Iteration: 1000 / 10000 [ 10%]  (Warmup)
## Chain 1: Iteration: 2000 / 10000 [ 20%]  (Warmup)
## Chain 1: Iteration: 2001 / 10000 [ 20%]  (Sampling)
## Chain 1: Iteration: 3000 / 10000 [ 30%]  (Sampling)
## Chain 1: Iteration: 4000 / 10000 [ 40%]  (Sampling)
## Chain 1: Iteration: 5000 / 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: 1.984 seconds (Warm-up)
## Chain 1:                8.183 seconds (Sampling)
## Chain 1:                10.167 seconds (Total)
## Chain 1:
```

For future use, we save this result as a data frame as well.

```
post_smp_logistic = as.data.frame(nfit_logistic)
colnames(post_smp_logistic)[1:7] <- colnames(post_smp_probit)[1:7]
```

```
par(mfrow = c(3, 3))
options(repr.plot.width = 10, repr.plot.height = 10)
par(mar = c(4, 4, 2, 2) + 0.1)
```

```r
for (i in 1:7){
  mean_beta_vb <- mean(posterior_flu$beta[, i])
  sd_beta_vb <- sqrt(mean(posterior_flu$sigma_beta[i, i]))

  beta <- seq(
    mean_beta_vb - 4 * sd_beta_vb,
    mean_beta_vb + 4 * sd_beta_vb,
    length.out = 1000)

  p_vb <- dnorm(
    beta,
    mean = mean_beta_vb,
    sd = sd_beta_vb)

  plot(beta, p_vb,
    type = "l",
    lwd = 2,
    col = "blue",
    xlab = "beta",
    ylab = "density",
    main = paste("beta", i))

  mean_beta_mcmc <- mean(post_smp_logistic[, i])
  sd_beta_mcmc <- sd(post_smp_logistic[, i])

  lines(beta, dnorm(beta,
    mean = mean_beta_mcmc,
    sd = sd_beta_mcmc),
    col = "red", lwd = 2)

  legend("topright", legend = c("VB", "MCMC"),
    col = c("blue", "red"), lwd = 2, cex = 0.8
  )
}
```



Compared to probit model, the logistic model has more uncertainty on all these parameter. From the plots, we can see the VB method and MCMC method shows very different patterns among some variables such as beta1, beta2, beta3, and beta4. beta5 and beta6 have the close mean and different variances. The Probit model uses the normal cumulative distribution function, whereas the logistic model uses the logistic function. This can lead to differences in coefficient estimates. Yet, the choice between Logistic Regression and Probit models does not affect the regression analysis conclusions, only the interpretations.