STAT 656: HW5

Rohan Dekate

Variational Bayes for the probit model

Write an R function to implement the VB algorithm for the probit model given in the slides. This should accept as input a dataset (X, Y) and return the approximations to the posterior distributions over β and the z's. It should also return the sequence of values of the variational free energy from each step of the optimization algorithm. Recall how the algorithm proceeds: start with some arbitrary initial distribution $q(\beta)$ over β, use that to calculate the $q(z_i)$'s, use those to calculate $q(\beta)$, and repeat till these distributions stop changing.

```r
probit_vb <- function(X, y, prior_var = 10, max_iter = 100, tol = 1e-6) {

  N <- nrow(X)
  d <- ncol(X)

  # Prior Precision Matrix (Lambda_0)
  Lambda_0 <- diag(1/prior_var, d)

  # Pre-compute Posterior Covariance (Sigma_beta)
  XtX <- t(X) %*% X
  Lambda_beta <- XtX + Lambda_0
  Sigma_beta <- solve(Lambda_beta)

  # Initialize q(beta)
  mu_beta <- rep(0, d)

  # Storage for ELBO sequence
  elbo_seq <- c()

  # Constants for ELBO calculation
  log_det_Sigma_beta <- determinant(Sigma_beta, logarithm = TRUE)$modulus
  log_det_Sigma_0 <- determinant(diag(prior_var, d), logarithm = TRUE)$modulus
  const_elbo_term <- 0.5 * as.numeric(log_det_Sigma_beta - log_det_Sigma_0)

  for (iter in 1:max_iter) {

    # We need the expected value of z, E[z], given current mu_beta.

    linear_pred <- X %*% mu_beta # Vector of means (m_i)
    q_sign <- 2 * y - 1
    m_scaled <- q_sign * linear_pred
    # Calculate standard normal pdf and cdf
    pdf_val <- dnorm(m_scaled)
    cdf_val <- pnorm(m_scaled)

    # Avoid division by zero for extreme values
    cdf_val <- pmax(cdf_val, 1e-15)

    # Inverse Mills Ratio
    imr <- pdf_val / cdf_val

    # E[z] vector
    E_z <- linear_pred + q_sign * imr

    mu_beta <- Sigma_beta %*% (t(X) %*% E_z)

    # Quadratic term:
    quad_term <- 0.5 * t(mu_beta) %*% Lambda_0 %*% mu_beta

    # Likelihood term (sum of log probabilities of the truncation intervals)
    log_Z_sum <- sum(log(cdf_val))

    elbo <- const_elbo_term - as.numeric(quad_term) + log_Z_sum
    elbo_seq <- c(elbo_seq, elbo)

    # Check convergence
    if (iter > 1 && abs(elbo - elbo_seq[iter-1]) < tol) {
      break
    }
  }

  # Return results
  return(list(
    mu_beta = mu_beta,
    Sigma_beta = Sigma_beta,
    E_z = E_z,
    elbo_seq = elbo_seq
  ))
}
```

Effect of computer-generated reminders

1. 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. Note that the free energy should not decrease over the course of the algorithm. Verify this is the case by plotting it against iteration number.

```r
flu_data <- read.table("flu_data1.txt", header = TRUE, na.strings = ".")
head(flu_data)
```

```
##   treatment vaccinated age copd heartd renal liverd
## 1         1          1  73    0      1     0      0
## 2         0          1  65    0      0     0      0
## 3         0          1  77    1      1     0      0
## 4         1          1  68    0      1     0      0
## 5         1          0  68    0      1     0      0
## 6         1          0  66    0      0     0      0
```

```r
# Report on missing data
n_total <- nrow(flu_data)
flu_data_clean <- na.omit(flu_data)
n_clean <- nrow(flu_data_clean)
n_dropped <- n_total - n_clean

cat(paste("Loaded", n_total, "observations.\n"))
```

```
## Loaded 2901 observations.
```

```r
cat(paste("Dropped", n_dropped, "observations with missing covariates.\n"))
```

```
## Dropped 8 observations with missing covariates.
```

```r
cat(paste("Analyzing", n_clean, "complete observations.\n\n"))
```

```
## Analyzing 2893 complete observations.
```

```r
y_response <- flu_data_clean$vaccinated

X_design <- model.matrix(
  vaccinated ~ treatment + scale(age) + copd + heartd + renal + liverd,
  data = flu_data_clean
)

# Get dimensions
N <- nrow(X_design)
d <- ncol(X_design)
cat(paste("Model has", N, "observations and", d, "beta coefficients (incl. intercept).\n"))
```

```
## Model has 2893 observations and 7 beta coefficients (incl. intercept).
```

```r
print("Predictors (X columns):")
```

```
## [1] "Predictors (X columns):"
```

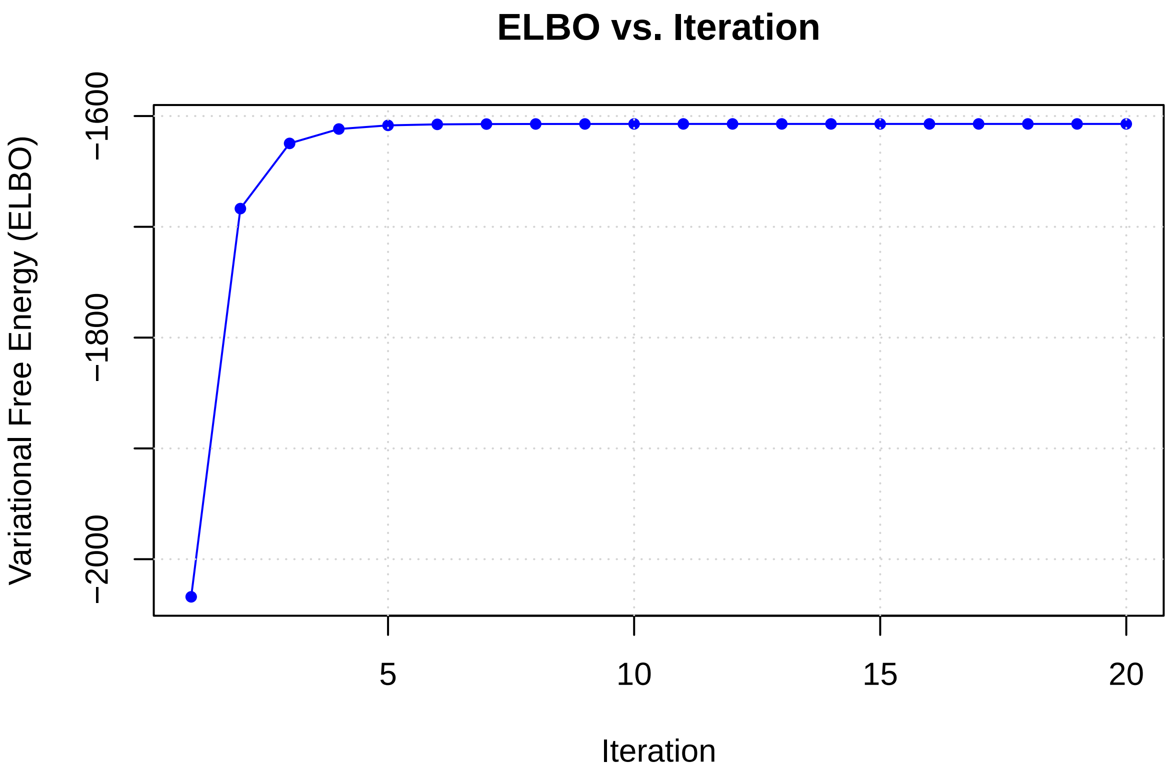
```r
print(colnames(X_design))
```

```
## [1] "(Intercept)" "treatment"   "scale(age)"  "copd"        "heartd"
## [6] "renal"       "liverd"
```

- **Prior:** I used a weakly informative Gaussian prior on β with mean $\mathbf{0}$ and covariance $\Sigma_0 = 10I$ (variance of 10 for each coefficient).
- **Variational Parameters:** The algorithm was initialized by setting the mean of the variational distribution for β (μ_β) to a vector of **zeros**. The covariance matrix Σ_β is constant in this specific algorithm (determined solely by data X and prior Σ_0), so it was pre-calculated before the first iteration.
- **Iterations:** The algorithm was configured to run for a maximum of **100 iterations**.
- **Convergence Criterion:** I defined convergence based on the **Variational Free Energy (ELBO)**. The algorithm stops early if the absolute difference in the ELBO between consecutive iterations is less than a tolerance of 10^{-6}.

```r
vb_result <- probit_vb(X_design, y_response, prior_var = 10, max_iter = 100, tol = 1e-6)
```

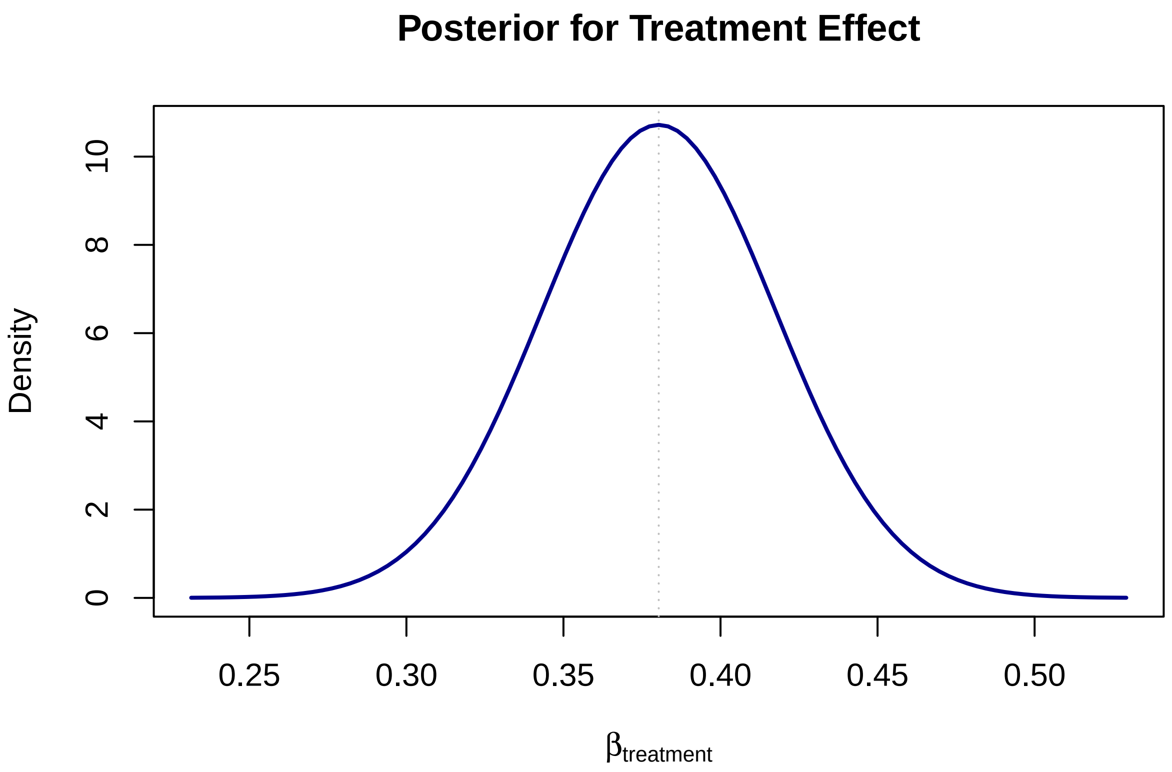
```r
# Plot the ELBO
plot(vb_result$elbo_seq, type = "o", col = "blue", pch = 16, cex = 0.8,
     xlab = "Iteration", ylab = "Variational Free Energy (ELBO)",
     main = "ELBO vs. Iteration")
grid()
```



2. Plot the approximation to the posterior distribution over β as well as a few of the z_i's. Based on your posterior approximation, what do you conclude about the effect of the treatment on patient vaccination?

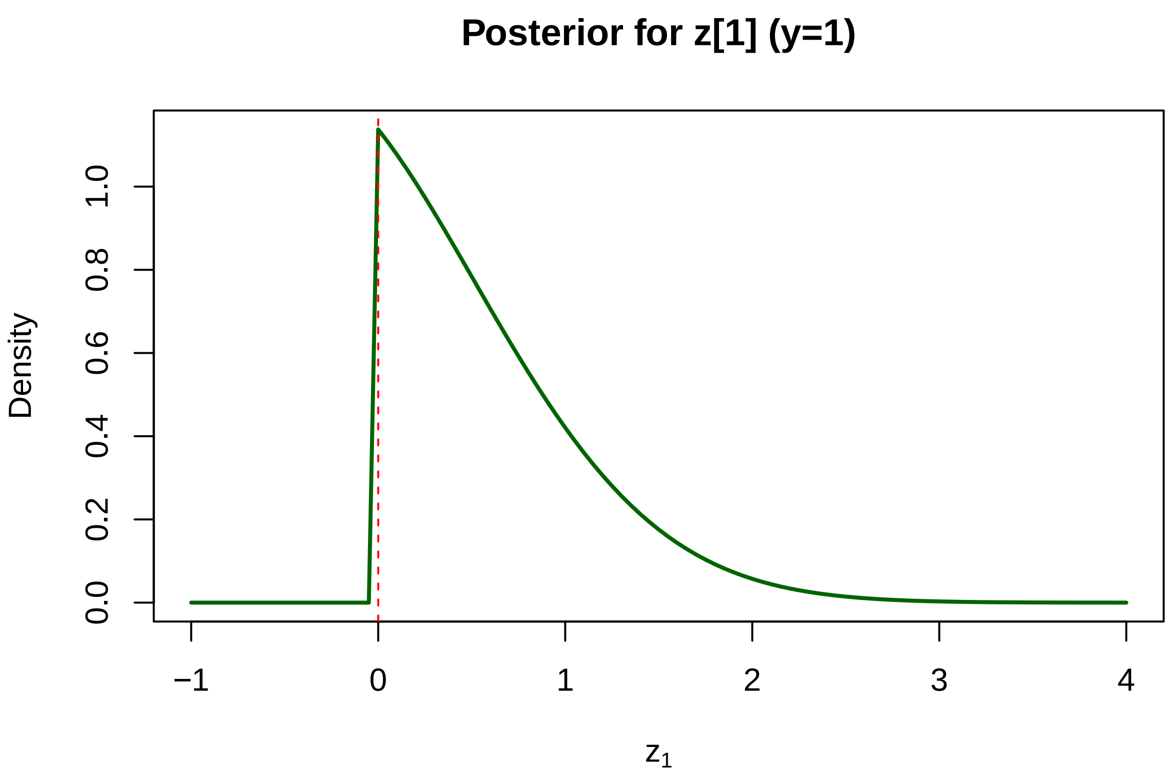
```r
# Plot 1: Posterior for Beta (Treatment)
trt_index <- which(colnames(X_design) == "treatment")
mu_trt <- vb_result$mu_beta[trt_index]
sd_trt <- sqrt(vb_result$Sigma_beta[trt_index, trt_index])

curve(dnorm(x, mean = mu_trt, sd = sd_trt),
      from = mu_trt - 4*sd_trt, to = mu_trt + 4*sd_trt,
      col = "darkblue", lwd = 2,
      xlab = expression(beta[treatment]), ylab = "Density",
      main = "Posterior for Treatment Effect")
abline(v = 0, col = "red", lty = 2)
abline(v = mu_trt, col = "gray", lty = 3)
```



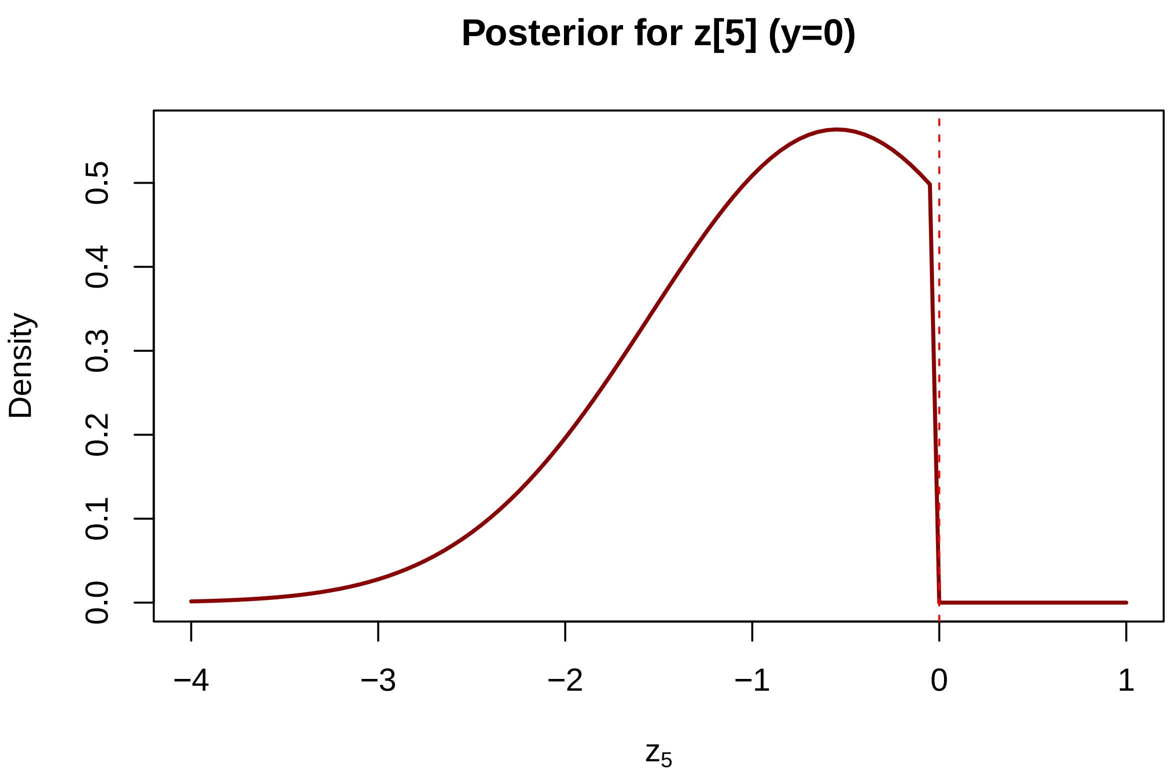
Since the entire 95% credible interval lies strictly above zero, we can conclude with high confidence that **the computer-generated reminder (treatment) has a positive effect on patient vaccination.**

```r
# Plot 2: Posterior for z_1 (Vaccinated, y=1)
i1 <- 1
mu_z1_raw <- sum(X_design[i1, ] * vb_result$mu_beta)
library(truncnorm) # Using for plotting density
curve(dtruncnorm(x, a = 0, b = Inf, mean = mu_z1_raw, sd = 1),
      from = -1, to = 4,
      col = "darkgreen", lwd = 2,
      xlab = expression(z[1]), ylab = "Density",
      main = paste0("Posterior for z[1] (y=", y_response[i1], ")"))
abline(v = 0, col = "red", lty = 2)
```



The posterior mass of z is entirely positive.

```r
# Plot 3: Posterior for z_5 (Not Vaccinated, y=0)
i5 <- 5
mu_z5_raw <- sum(X_design[i5, ] * vb_result$mu_beta)
curve(dtruncnorm(x, a = -Inf, b = 0, mean = mu_z5_raw, sd = 1),
      from = -4, to = 1,
      col = "darkred", lwd = 2,
      xlab = expression(z[5]), ylab = "Density",
      main = paste0("Posterior for z[5] (y=", y_response[i5], ")"))
abline(v = 0, col = "red", lty = 2)
```



The posterior mass of z is entirely negative.

3. Compare the latter with an MCMC approximation to the posterior (either using Stan or using your code from the last assignment). Investigate how your variational approximation deviates from the MCMC posterior distribution, looking at the posterior mean, variance and covariance of some variables. Comment on your conclusions.

```r
library(rstan)
```

```
## Loading required package: StanHeaders

##
## rstan version 2.32.7 (Stan version 2.32.2)

## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
## For within-chain threading using `reduce_sum()` or `map_rect()` Stan functions,
## change `threads_per_chain` option:
## rstan_options(threads_per_chain = 1)
```

```r
library(parallel)

# Set Stan to run on multiple cores
options(mc.cores = parallel::detectCores())

stan_model_code <- "
data {
  int<lower=0> N;          // Number of observations
  int<lower=0> d;          // Number of predictors (including intercept)
  matrix[N, d] X;          // Design matrix
  int<lower=0, upper=1> y[N]; // Binary response vector
}
parameters {
  vector[d] beta;          // Vector of coefficients
}
model {
  // Define the linear predictor
  vector[N] eta = X * beta;

  // Prior: beta ~ N(0, 10*I)
  beta ~ normal(0, sqrt(10));

  // Likelihood:
  y ~ bernoulli(Phi(eta));
}
"
```

```
# Create the data list for Stan
stan_data_list <- list(
  N = N,
  d = d,
  X = X_design,
  y = y_response
)

# Compile Stan model
stan_model <- stan_model(model_code = stan_model_code)
```

```
## Trying to compile a simple C file
```

```
## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## using C compiler: ‘Apple clang version 17.0.0 (clang-1700.4.4.1)’
## using SDK: ‘MacOSX26.1.sdk’
## clang -arch arm64 -std=gnu2x -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Li
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/library/StanHeade
## In file included from /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/library/RcppEigen
## In file included from /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/library/RcppEigen
## /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/library/RcppEigen/include/Eigen/src/Cor
##   679 | #include <cmath>
##       |          ^~~~~~~
## 1 error generated.
## make: *** [foo.o] Error 1
```

```
# Run NUTS sampler
stan_fit <- sampling(
  stan_model,
  data = stan_data_list,
  iter = 2000,      # 1000 warmup + 1000 post-warmup
  warmup = 1000,
  chains = 4,       # 4 chains * 1000 = 4000 total samples
  seed = 42
```

```
)
```

```
# Extract MCMC samples
mcmc_samples <- rstan::extract(stan_fit)$beta

mcmc_means <- colMeans(mcmc_samples)
mcmc_sds   <- apply(mcmc_samples, 2, sd)
vb_means   <- as.numeric(vb_result$mu_beta)
vb_sds     <- sqrt(diag(vb_result$Sigma_beta))


comparison_df <- data.frame(
  Param = colnames(X_design),
  MCMC_Mean = mcmc_means,
  VB_Mean = vb_means,
  Diff_Mean = vb_means - mcmc_means,
  MCMC_SD = mcmc_sds,
  VB_SD = vb_sds,
  Ratio_SD = vb_sds / mcmc_sds
)

numeric_cols <- sapply(comparison_df, is.numeric)
comparison_df_rounded <- comparison_df
comparison_df_rounded[numeric_cols] <- round(comparison_df[numeric_cols], 4)
print(comparison_df_rounded)
```

```
##                   Param MCMC_Mean VB_Mean Diff_Mean MCMC_SD  VB_SD Ratio_SD
## (Intercept) (Intercept)   -1.0005 -1.0006   -0.0002  0.0504 0.0360   0.7143
## treatment     treatment    0.3816  0.3803   -0.0013  0.0505 0.0372   0.7366
## scale(age)   scale(age)    0.1318  0.1316   -0.0002  0.0278 0.0190   0.6847
## copd               copd    0.2908  0.2916    0.0007  0.0562 0.0420   0.7481
## heartd           heartd    0.0432  0.0447    0.0015  0.0519 0.0379   0.7302
## renal             renal   -0.0135 -0.0079    0.0056  0.2232 0.1638   0.7337
## liverd           liverd    0.0105  0.0507    0.0402  0.4620 0.3323   0.7194
```

- The `Diff_Mean` column is close to 0, indicating that the VB algorithm successfully located the mode/mean of the posterior distribution.
- `VB_SD` are less than `MCMC_SD`. To minimize KL divergence, the approximating distribution q tends to focus on the mode of the true posterior (p) and avoids areas where p is small.

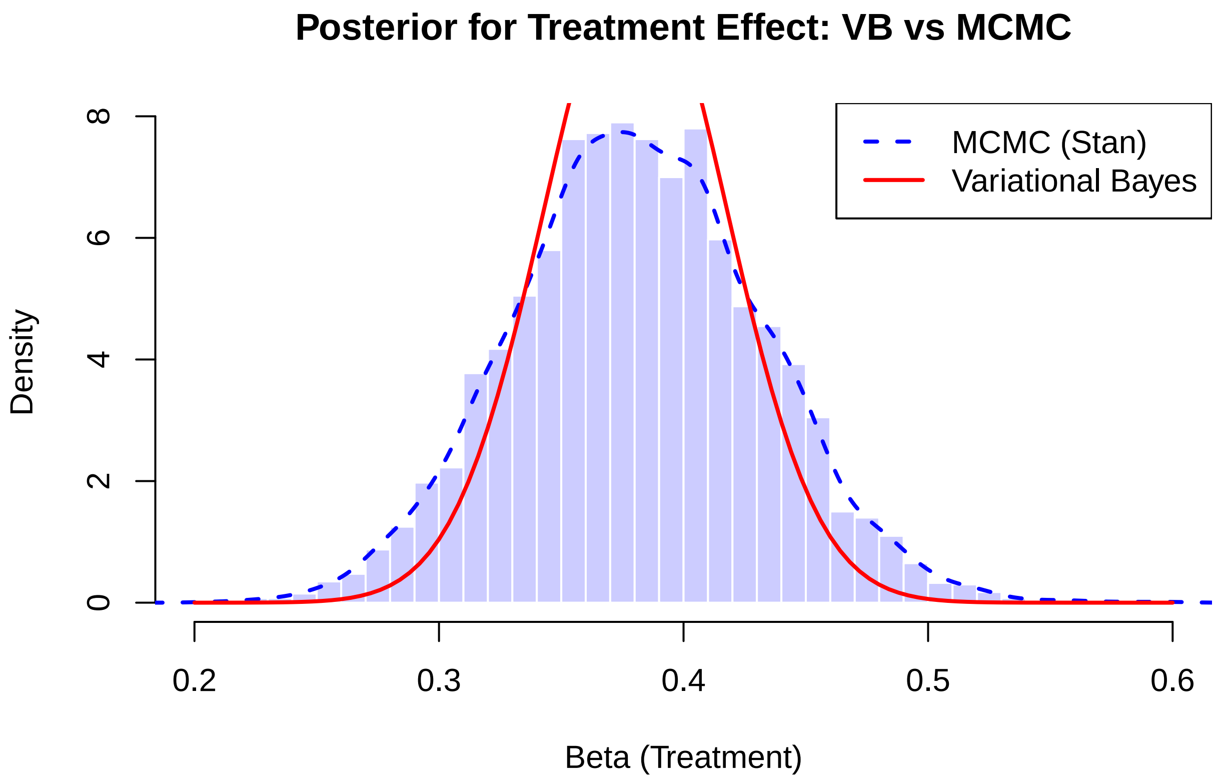
```
trt_idx <- which(colnames(X_design) == "treatment")

par(mfrow = c(1, 1), mar = c(5, 4, 4, 1))
hist(mcmc_samples[, trt_idx], breaks = 40, freq = FALSE,
     col = rgb(0, 0, 1, 0.2), border = "white",
     main = "Posterior for Treatment Effect: VB vs MCMC",
     xlab = "Beta (Treatment)")

# Add MCMC density line
lines(density(mcmc_samples[, trt_idx]), col = "blue", lwd = 2, lty = 2)

# Add VB density line (Gaussian approximation)
curve(dnorm(x, mean = vb_result$mu_beta[trt_idx], sd = sqrt(vb_result$Sigma_beta[trt_idx, trt_idx])),
      add = TRUE, col = "red", lwd = 2)
```

```
legend("topright", legend = c("MCMC (Stan)", "Variational Bayes"),
       col = c("blue", "red"), lwd = 2, lty = c(2, 1))
```


Posterior for Treatment Effect: VB vs MCMC
Density
Beta (Treatment)
MCMC (Stan)
Variational Bayes