Simple Logit Model

Simple Logit Models

First, we generate a binary dataset with simple inherent structure, abundant data and enough features.

```r
set.seed(123)

n <- 2000 # abundant data
p <- 20 # enough features

# For each feature, we sample data independently from N(0, I_n) to avoid collinearity
X <- matrix(NA, n, p)
for (j in 1:p) {
  X[, j] <- rnorm(n)
}
colnames(X) <- paste0("x", 1:p)

# Set the underlying true beta
beta_true <- c(1.9, -0.9, 0.4, 2.1, -1.1, rep(0, p-5))
intercept_true <- 0 # set beta0 individually as we did not define the all-one column in X
Z_true <- c(rep(1, 5), rep(0, p-5))

# Construct the logistic model
eta <- as.vector(intercept_true + X %*% beta_true)
p_true <- 1 / (1 + exp(-eta))
# generate binary response
y <- rbinom(n = n, size = 1, prob = p_true)
```

Then we perform independent Metropolis Hastings, with Laplace proposal.

```r
library(MASS)
library(mvtnorm)

# compute the log posterior of beta to compute the acceptance probability
logpost_b <- function(b, z, X, y, sigma0=100) {
  eta <- as.vector(X %*% (z * b))
  # simplified logistic log-likelihood
  loglik <- sum(y * eta - log1p(exp(eta)))
  # Gaussian prior on beta
  logprior <- -0.5 * sum(b^2 / sigma0^2)
  return(loglik + logprior)
}

# general proposal by Laplace approximation when z is a all one vector
general_proposal <- function(X, y, c_scale = 1.2) {
  n <- nrow(X); p <- ncol(X)
```

```r
  # MLE from logistic regression as mean of the proposal
  dat <- data.frame(y = y, X)
  fit_glm <- glm(y ~ ., data = dat, family = binomial(link = "logit"))
  beta_mle <- coef(fit_glm)[-1] # drop intercept

  # negative Hessian of MLE
  eta_hat <- as.vector(X %*% beta_mle)
  p_hat <- plogis(eta_hat)
  W <- diag(p_hat * (1 - p_hat))
  H_ll <- t(X) %*% W %*% X

  # proposal covariance, with some prior curvature
  Sigma_q <- c_scale * solve(H_ll)

  list(
    betahat = as.vector(beta_mle),
    Sigma_q = Sigma_q
  )
}

# a single update for beta
mh_update_b <- function(b_curr, z_curr, X, y, sigma0, prop) {

  # propose b* ~ N(betahat, Sigma_q)
  b_prop <- as.vector(
    MASS::mvrnorm(1, mu = prop$betahat, Sigma = prop$Sigma_q)
  )

  # log posterior at current and proposed values
  logpi_curr <- logpost_b(b_curr, z_curr, X, y, sigma0)
  logpi_prop <- logpost_b(b_prop, z_curr, X, y, sigma0)

  # log proposal at current and proposed points
  logq_curr <- mvtnorm::dmvnorm(
    b_curr,
    mean = prop$betahat,
    sigma = prop$Sigma_q,
    log = TRUE
  )
  logq_prop <- mvtnorm::dmvnorm(
    b_prop,
    mean = prop$betahat,
    sigma = prop$Sigma_q,
    log = TRUE
  )

  # log acceptance probability
  log_alpha <- (logpi_prop + logq_curr) - (logpi_curr + logq_prop)
  if (log(runif(1)) < log_alpha) {
    return(b_prop)   # accept
  } else {
    return(b_curr)   # reject
  }
```

```
}

# a single update for z
flip_z <- function(z_curr, b_curr, X, y) {
  p <- length(z_curr)

  # log likelihood with variable z
  loglik_given_z <- function(z_vec) {
    eta <- as.vector(X %*% (z_vec * b_curr))
    sum(y * eta - log1p(exp(eta)))
  }

  for (j in sample.int(p)) {
    # compute the full conditional probability
    z0 <- z_curr; z0[j] <- 0
    z1 <- z_curr; z1[j] <- 1

    ll0 <- loglik_given_z(z0)
    ll1 <- loglik_given_z(z1)

    log_odds <- ll1 - ll0
    p1 <- plogis(log_odds)
    # sample z from bern
    z_curr[j] <- rbinom(1, size = 1, prob = p1)
  }
  return(z_curr)
}

run_mcmc <- function(X, y,
                     sigma0 = 100,
                     n_iter = 5000,
                     burn_in = 1000,
                     c_scale = 1.2) {
  set.seed(111)

  n <- nrow(X); p <- ncol(X)

  # construct the proposal
  prop <- general_proposal(X, y, c_scale)

  # initialize beta and z
  b_curr <- rep(0, p)
  z_curr <- rep(1, p)

  # matrices to store samples
  B_save <- matrix(NA, nrow = n_iter, ncol = p)
  Z_save <- matrix(NA, nrow = n_iter, ncol = p)

  # to store the number of accept
  acc_b <- 0

  for (iter in 1:n_iter) {
    # update b by independence MH
```

```
    b_new <- mh_update_b(b_curr, z_curr, X, y, sigma0, prop)
    if (!all(b_new == b_curr)) acc_b <- acc_b + 1
    b_curr <- b_new

    # update z by Gibbs sampling
    z_curr <- flip_z(z_curr, b_curr, X, y)

    B_save[iter, ] <- b_curr
    Z_save[iter, ] <- z_curr

    if (iter %% 10000 == 0) {
      cat("iter =", iter, "\n")
    }
  }

  list(
    B = B_save[(burn_in+1):n_iter, , drop = FALSE],
    Z = Z_save[(burn_in+1):n_iter, , drop = FALSE],
    acc_rate_b = acc_b / n_iter,
    proposal = prop
  )
}
```

```
library(coda)

t_mh <- system.time(
  res <- run_mcmc(X, y, n_iter = 15000, burn_in = 5000)
)
```

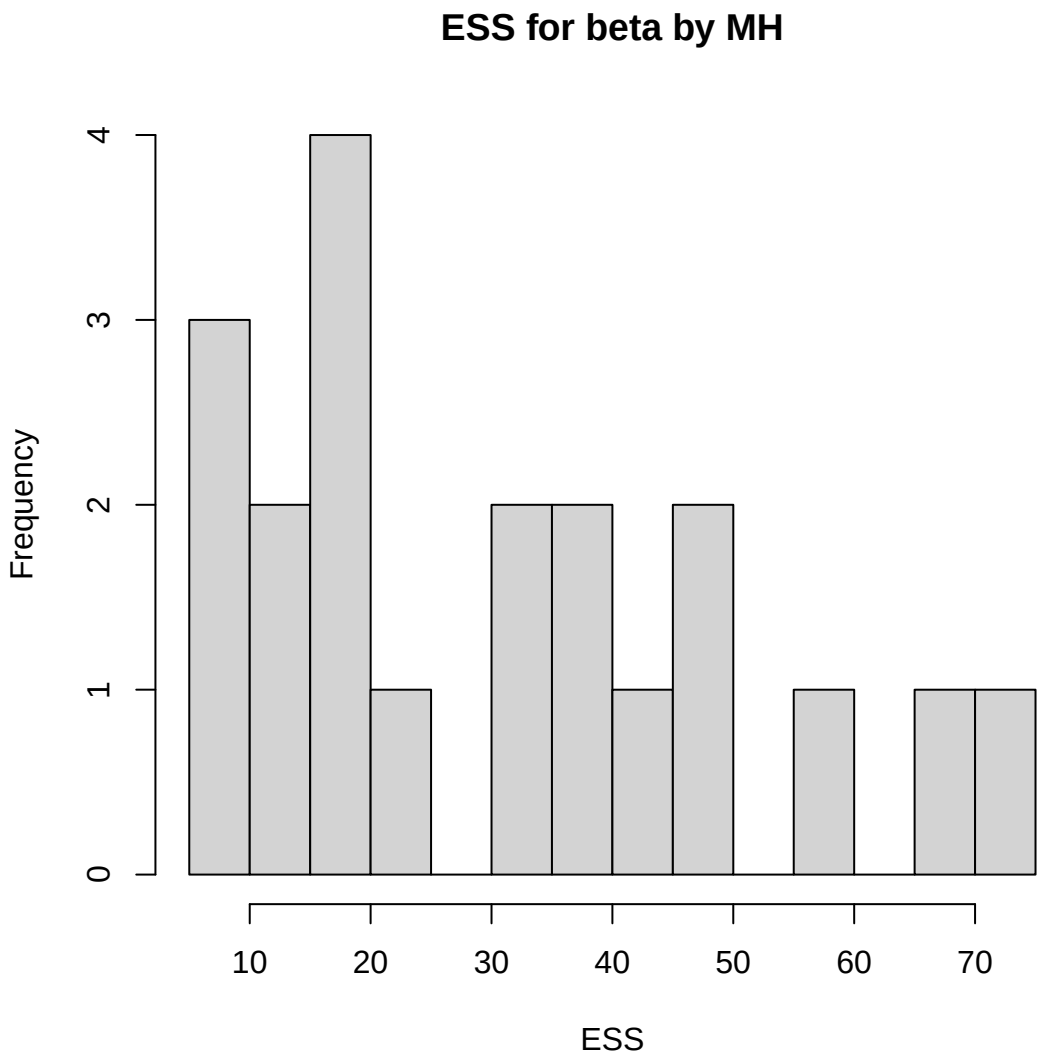
```
## iter = 10000
```

```
t_mh["elapsed"]
```

```
## elapsed
##  27.297
```

```
# extract the result
beta_mh <- res$B
Z_mh <- res$Z
beta_eff_mh <- beta_mh * Z_mh
```

```
ess_beta_mh <- effectiveSize(as.mcmc(beta_mh))
hist(ess_beta_mh, breaks = 20, main = "ESS for beta by MH", xlab = "ESS")
```




```r
# compute 95% credible interval coverage for beta
CI_beta_mh <- apply(beta_eff_mh, 2, quantile, c(0.025, 0.975))
in_CI_beta_mh <- (beta_true >= CI_beta_mh[1, ]) & (beta_true <= CI_beta_mh[2, ])
mean(in_CI_beta_mh)  # coverage of beta
```

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

```r
# compute evaluation of selcetion
pip <- colMeans(Z_mh)

selected <- as.integer(pip > 0.5)

TP <- sum(selected == 1 & Z_true == 1)
FP <- sum(selected == 1 & Z_true == 0)
TN <- sum(selected == 0 & Z_true == 0)
FN <- sum(selected == 0 & Z_true == 1)
```

```r
TPR <- TP / (TP + FN)  # true positive rate
FPR <- FP / (FP + TN)  # false positive rate

TPR
```

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

```r
FPR
```

```
## [1] 0.3333333
```

Now we perform the P'olya-Gamma Data Augmentation

```r
library(BayesLogit)
# update (omega, beta) by PG
pg_update_beta <- function(b_curr, z_curr, X, y, sigma0 = 100) {
  n <- nrow(X); p <- ncol(X)

  # define prior parameters
  b0    <- rep(0, p)
  B_inv <- diag(1 / sigma0^2, p)

  # sample omega from PG(1, eta)
  eta <- as.vector(X %*% (z_curr * b_curr))
  omega <- BayesLogit::rpg(num = n, h = 1, z = eta)

  # compute kappa, X_z, V and m
  kappa <- y - 0.5
  X_z   <- sweep(X, 2, z_curr, "*")
  WX    <- X_z * omega
  XtWX  <- t(X_z) %*% WX
  V_omega <- solve(XtWX + B_inv)
  m_omega <- V_omega %*% (t(X_z) %*% kappa)

  # sample beta from N(m, V)
  b_new <- as.vector(
    MASS::mvrnorm(1, mu = as.vector(m_omega), Sigma = V_omega)
  )

  return(b_new)
}

# PG Gibbs sampling
run_mcmc_pg <- function(X, y,
                        sigma0  = 100,
                        n_iter  = 5000,
                        burn_in = 1000) {
  set.seed(111)

  n <- nrow(X); p <- ncol(X)

  # initialize
```

```
  b_curr <- rep(0, p)
  z_curr <- rep(1, p)
  #store samples
  B_save <- matrix(NA, nrow = n_iter, ncol = p)
  Z_save <- matrix(NA, nrow = n_iter, ncol = p)

  for (iter in 1:n_iter) {
    # update beta by PGDA
    b_curr <- pg_update_beta(b_curr, z_curr, X, y, sigma0 = sigma0)

    # update z by Gibbs with the same method as independent MH
    z_curr <- flip_z(z_curr, b_curr, X, y)

    B_save[iter, ] <- b_curr
    Z_save[iter, ] <- z_curr

    if (iter %% 10000 == 0) {
      cat("iter =", iter, "\n")
    }
  }

  list(
    B = B_save[(burn_in + 1):n_iter, , drop = FALSE],
    Z = Z_save[(burn_in + 1):n_iter, , drop = FALSE]
  )
}
```

```
library(coda)

t_pg <- system.time(
  res_pg <- run_mcmc_pg(X, y, n_iter = 15000, burn_in  = 5000)
)
```

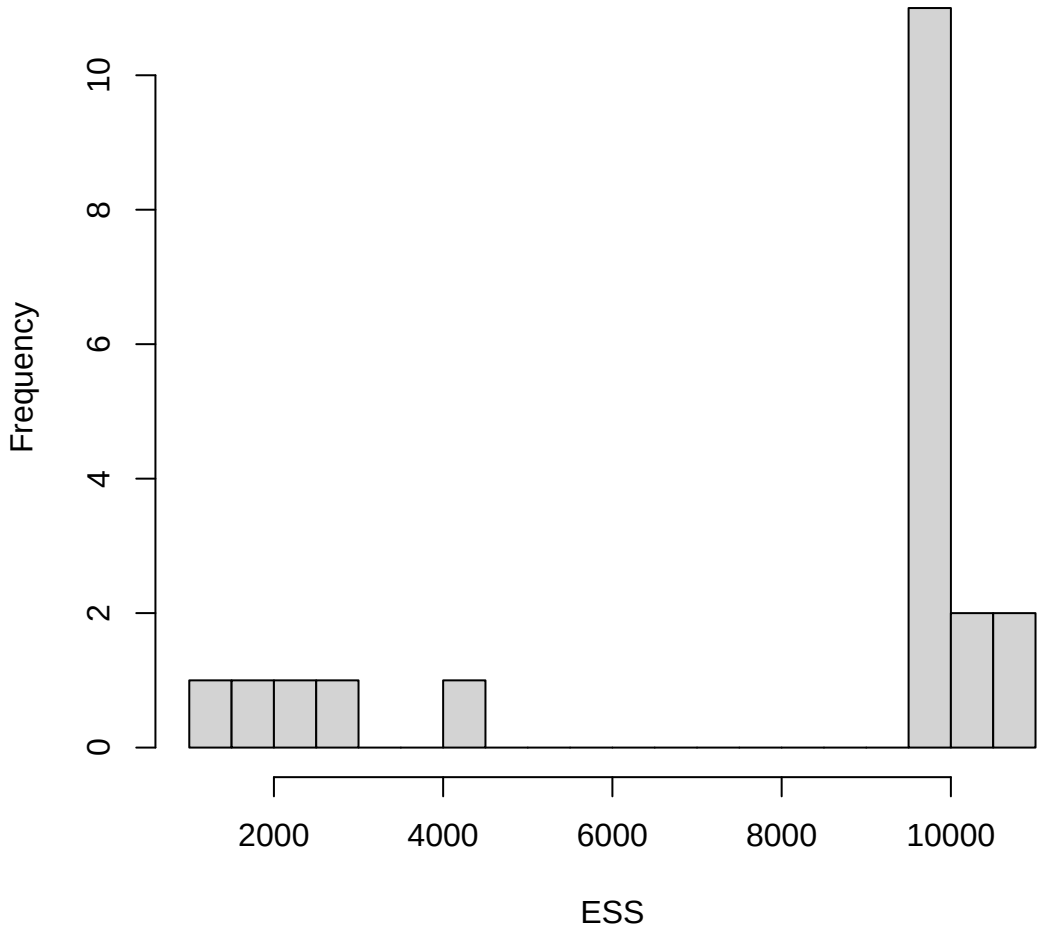
```
## iter = 10000
```

```
t_pg["elapsed"]
```

```
## elapsed
##  41.446
```

```
# extract the result
beta_pg <- res_pg$B
Z_pg <- res_pg$Z
beta_eff_pg <- beta_pg * Z_pg
```

```
ess_beta_pg <- effectiveSize(as.mcmc(beta_pg))
hist(ess_beta_pg, breaks = 20, main = "ESS for beta by PG", xlab = "ESS")
```




```
# compute 95% credible interval coverage for beta
CI_beta_pg <- apply(beta_eff_pg, 2, quantile, c(0.025, 0.975))
in_CI_beta_pg <- (beta_true >= CI_beta_pg[1, ]) & (beta_true <= CI_beta_pg[2, ])
mean(in_CI_beta_pg)  # coverage of beta
```

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

```
# compute evaluation of selcetion
pip <- colMeans(Z_pg)

selected <- as.integer(pip > 0.5)

TP <- sum(selected == 1 & Z_true == 1)
FP <- sum(selected == 1 & Z_true == 0)
TN <- sum(selected == 0 & Z_true == 0)
FN <- sum(selected == 0 & Z_true == 1)
```

```
TPR <- TP / (TP + FN)
FPR <- FP / (FP + TN)

TPR
```

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

```
FPR
```

```
## [1] 0
```

Now we move on to compute JAGS

```
model <- "
model {
  # likelihood
  for (i in 1:N) {
    y[i] ~ dbern(p[i])
    p[i] <- 1 / (1 + exp(-eta[i]))
    eta[i] <- inprod(X[i,], beta_eff[])
  }

  # priors and feature selection
  for (j in 1:P) {
    beta[j] ~ dnorm(0.0, 1.0E-4)
    z[j] ~ dbern(0.5)
    beta_eff[j] <- z[j] * beta[j]
  }

}
"
```

```
library(rjags)
```

```
## Linked to JAGS 4.3.2
```

```
## Loaded modules: basemod,bugs
```

```
t_jags <- system.time({
  jm <- jags.model(textConnection(model), data = list(N = nrow(X), P = ncol(X), y=y, X=X))
  update(jm, 5000) # burnin
  cs <- coda.samples(jm, c('beta', 'z'), 10000)
  s <- as.data.frame(cs[[1]])
})
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 2000
```

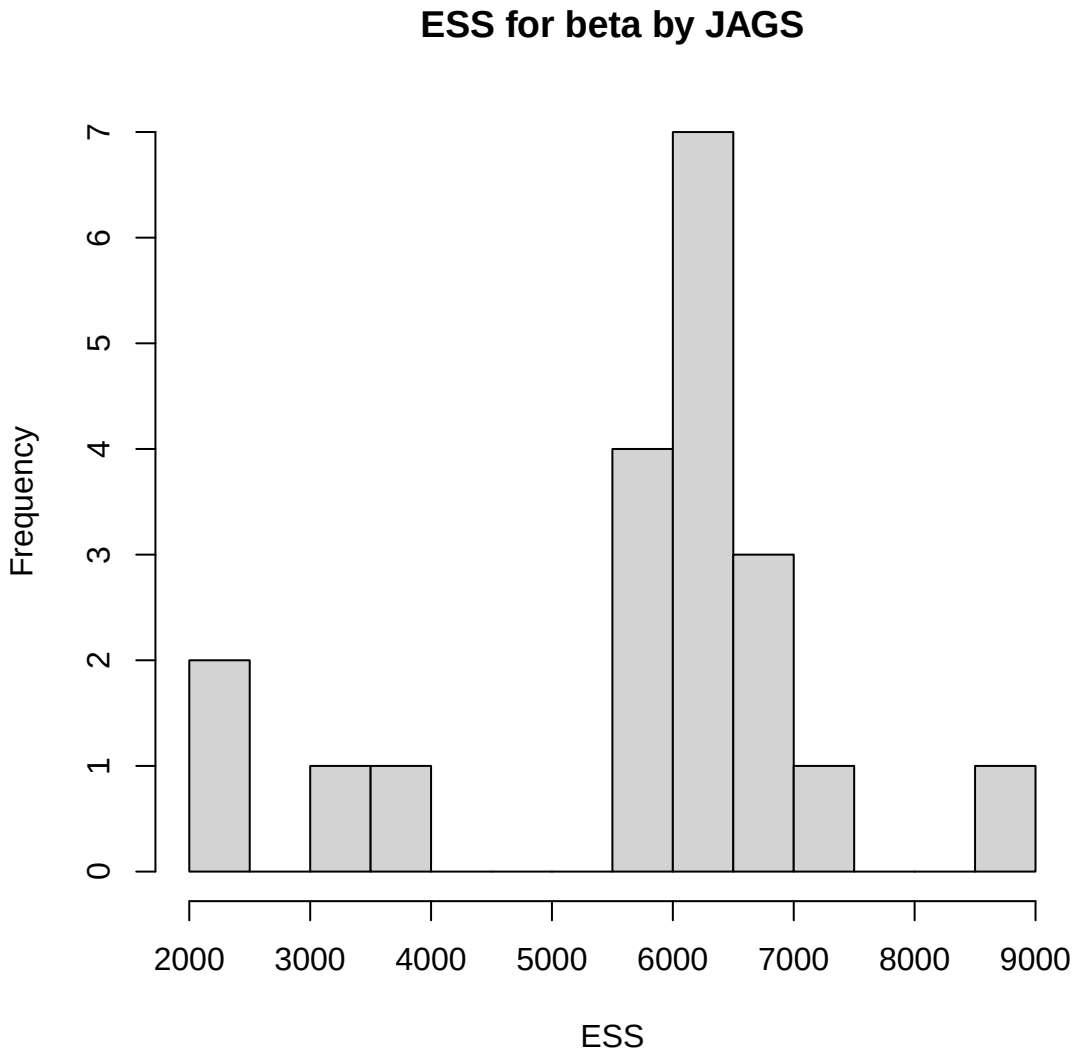
```
##    Unobserved stochastic nodes: 40
##    Total graph size: 54067
##
## Initializing model
```

```
t_jags["elapsed"]
```

```
## elapsed
## 345.981
```

```
beta_jags <- s[, 1:20]
Z_jags <- s[, 21:40]
beta_eff_jags <- beta_jags * Z_jags
```

```
# compute the effective sample size
ess_beta_jags <- effectiveSize(as.mcmc(beta_jags))
hist(ess_beta_jags, breaks=20, main = "ESS for beta by JAGS", xlab = "ESS")
```



```r
# compute 95% credible interval of beta
CI_beta_jags <- apply(beta_eff_jags, 2, quantile, c(0.025, 0.975))
in_CI_beta_jags <- (beta_true >= CI_beta_jags[1, ]) & (beta_true <= CI_beta_jags[2, ])
mean(in_CI_beta_jags)  # coverage of beta
```

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

```r
# compute evaluation of selcetion
pip <- colMeans(Z_jags)

selected <- as.integer(pip > 0.5)

TP <- sum(selected == 1 & Z_true == 1)
FP <- sum(selected == 1 & Z_true == 0)
TN <- sum(selected == 0 & Z_true == 0)
FN <- sum(selected == 0 & Z_true == 1)

TPR <- TP / (TP + FN)  # true positive rate
FPR <- FP / (FP + TN)  # false positive rate

TPR
```

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

```r
FPR
```

```
## [1] 0
```

apply glm module in JAGS

```r
model_glm <- "
model {
  # likelihood (GLM form)
  for (i in 1:N) {
    y[i] ~ dbern(p[i])
    logit(p[i]) <- inprod(X[i,], beta_eff[])
  }

  # priors and feature selection
  for (j in 1:P) {
    beta[j] ~ dnorm(0.0, 1.0E-4)
    z[j] ~ dbern(0.5)
    beta_eff[j] <- z[j] * beta[j]
  }
}
"
```

```r
library(rjags)
load.module("glm")
```

```
## module glm loaded
```

```r
t_jags_glm <- system.time({
  jm_glm <- jags.model(
    textConnection(model_glm),
    data = list(N = nrow(X), P = ncol(X), y = y, X = X)
  )
  update(jm_glm, 5000)
  cs_glm <- coda.samples(jm_glm, c("beta", "z"), 10000)
  s_glm <- as.data.frame(cs_glm[[1]])
})
```

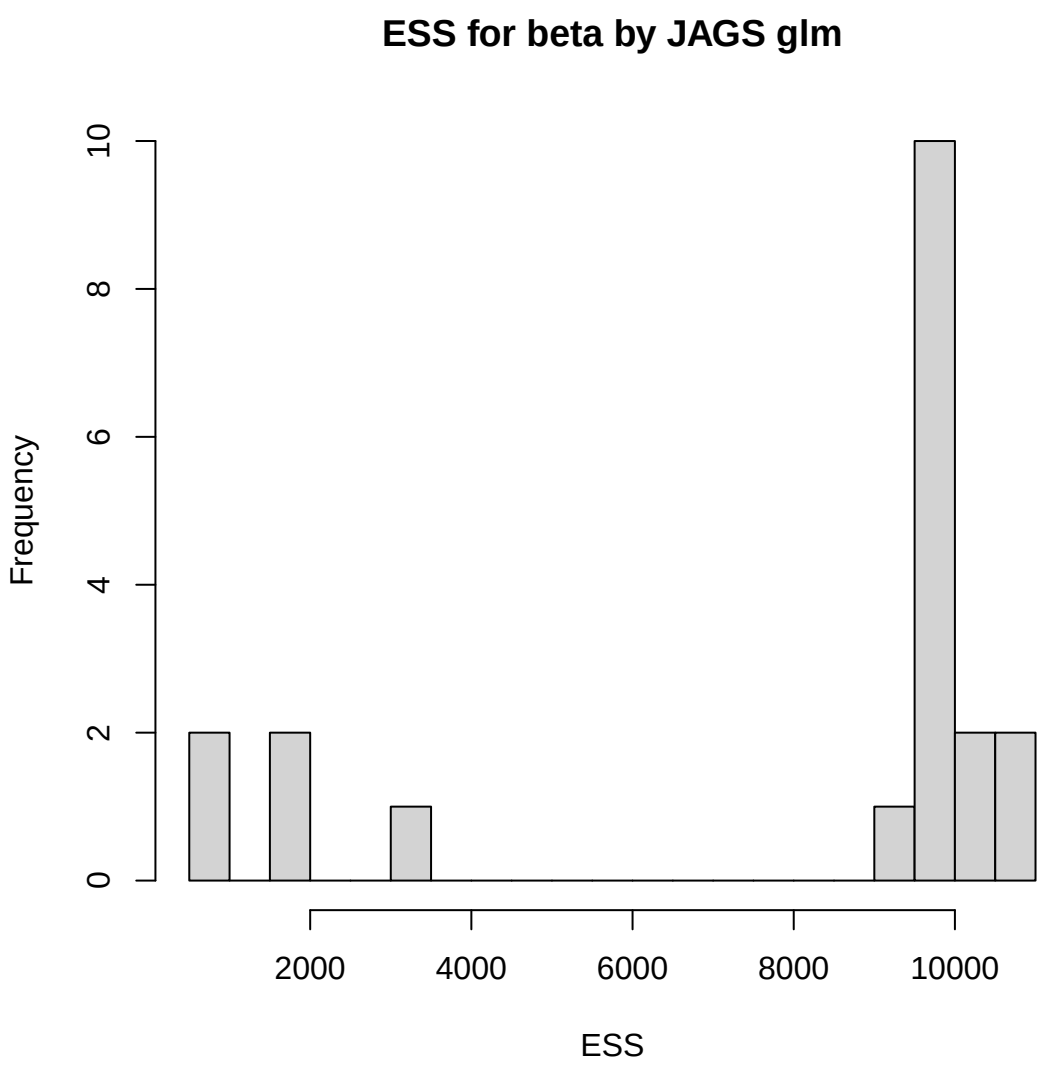
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 2000
##    Unobserved stochastic nodes: 40
##    Total graph size: 48066
##
## Initializing model
```

```r
t_jags_glm["elapsed"]
```

```
## elapsed
## 166.972
```

```r
beta_jags_glm <- s_glm[, 1:20]
Z_jags_glm <- s_glm[, 21:40]
beta_eff_jags_glm <- beta_jags_glm * Z_jags_glm
```

```r
# compute the effective sample size
ess_beta_jags_glm <- effectiveSize(as.mcmc(beta_jags_glm))
hist(ess_beta_jags_glm, breaks=20, main = "ESS for beta by JAGS glm", xlab = "ESS")
```




```
# compute 95% credible interval of beta
CI_beta_jags_glm <- apply(beta_eff_jags_glm, 2, quantile, c(0.025, 0.975))
in_CI_beta_jags_glm <- (beta_true >= CI_beta_jags_glm[1, ]) & (beta_true <= CI_beta_jags_glm[2, ])
mean(in_CI_beta_jags_glm)  # coverage of beta
```

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

```
# compute evaluation of selcetion
pip <- colMeans(Z_jags_glm)

selected <- as.integer(pip > 0.5)

TP <- sum(selected == 1 & Z_true == 1)
FP <- sum(selected == 1 & Z_true == 0)
TN <- sum(selected == 0 & Z_true == 0)
FN <- sum(selected == 0 & Z_true == 1)
```

```r
TPR <- TP / (TP + FN)  # true positive rate
FPR <- FP / (FP + TN)  # false positive rate

TPR
```

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

```r
FPR
```

```
## [1] 0
```