Hierarchical Model

First we simulate a dataset with underlying hierarchical structure

```r
set.seed(111)
# define the number of individuals and features
n_id <- 300
p <- 20

# randomly define the number of measurements for different individuals
times <- sample(5:15, n_id, replace = TRUE)
# generate id sequence
id <- rep(1:n_id, times = times)
N <- length(id)

# generate covariance matrix
X <- matrix(rnorm(N * p), nrow = N, ncol = p)
colnames(X) <- paste0("x", 1:p)

# define the underlying true beta and id-specific standard deviation
beta_true <- c(1.9, -0.9, 0.4, 2.1, -1.1, rep(0, p - 5))
alpha_true <- 0 # explicitly define the intercept
sigma_b_true <- 1
Z_true <- c(rep(1, 5), rep(0, p - 5))

# randomly generate id-specific contribution
b_true <- rnorm(n_id, mean = 0, sd = sigma_b_true)

# calculate the Bernoulli probability and generate y
eta <- alpha_true + as.vector(X %*% beta_true) + b_true[id]
p_true <- 1 / (1 + exp(-eta))
y <- rbinom(n = N, size = 1, prob = p_true)

# store the values
dat <- data.frame(
  id = id,
  y = y,
  X
)

truth <- list(
  alpha_true = alpha_true,
  beta_true = beta_true,
  b_true = b_true,
  sigma_b_true = sigma_b_true,
  p_true = p_true,
  eta_true = eta
)
```

```r
head(dat)
```

```
##   id y         x1          x2          x3          x4          x5          x6
## 1  1 1  0.1061932 -1.92299290  1.38408920  0.04055086  0.05020737 -1.36105887
## 2  1 1  0.2881800 -1.12919963 -0.05056547 -0.56591704 -1.53511748 -0.05821540
## 3  1 0 -0.2971965  2.17758019  0.11644058 -0.87718488 -0.22110055  0.13126926
## 4  1 0 -0.1624143 -0.25748628  0.59161592 -1.82510506  0.52830258 -1.64079935
## 5  1 0 -1.6696433 -0.04552489 -0.71642117  0.87114289 -1.01188513  0.01129197
## 6  1 1 -0.5427853 -1.08201650 -1.49646181  0.83917092  0.31414187 -0.29452461
##           x7        x8          x9          x10        x11         x12
## 1 -0.7725859  1.976667  0.22638352  1.159370285  0.1091549 -0.33729330
## 2 -0.8523855 -1.080488 -3.09116592 -1.138109491 -0.2180373  0.02513033
## 3  0.6574077 -1.356557 -0.41342654  0.419517017 -1.2627505  0.16321666
## 4 -0.8866740  0.477777 -0.08993882  0.002445591 -1.6395363  2.82907100
## 5  0.8998769 -0.398591  0.63959704  0.209607988  1.0742615  0.74545492
## 6 -0.2520969 -1.051270  0.65347799 -0.439633640  1.0543734 -2.13565516
##            x13        x14        x15          x16        x17        x18
## 1  0.008878051  0.4533650  0.8162014  0.927213109  1.4807535 -0.3150755
## 2 -0.645998320  1.1468077 -0.1585042  1.937436705 -0.3366068  0.2142337
## 3 -0.825742629  1.2156627 -0.1489291  0.001393825 -0.1035279 -0.9601828
## 4 -1.061598489 -0.1634854  0.8953985  0.028725470 -0.6325180  1.6859621
## 5  0.939349314 -1.2536649 -1.6048691  0.581156868 -0.6752614 -0.4076423
## 6  0.480236948 -1.5751585 -0.7584556 -1.205792792 -1.4038525 -0.6414932
##           x19         x20
## 1  0.02567756  0.53259712
## 2 -0.85343145  0.43622709
## 3 -0.72367296 -0.42614565
## 4  0.50992050 -0.55813878
## 5  0.67784747  0.08948889
## 6 -0.03182698 -0.09182010
```

Then we perform MH

```r
# log likelihood
loglik_hier <- function(beta, z, b, X, y, id) {
  eta <- as.vector(X %*% (beta * z) + b[id])
  # simplified expression
  sum(y * eta - log1p(exp(eta)))
}

# log posterior
logpost_beta <- function(beta, z, b, X, y, id, sigma0 = 100) {
  loglik <- loglik_hier(beta, z, b, X, y, id)
  logprior <- -0.5 * sum(beta^2 / sigma0^2)
  loglik + logprior
}

# a single update for beta
mh_update_beta <- function(beta_curr, z_curr, b_curr,
                           X, y, id,
                           sigma0 = 100,
                            tau_beta = 0.05) {
  p <- length(beta_curr)
  beta_prop <- rnorm(p, mean = beta_curr, sd = tau_beta)

  logpi_curr <- logpost_beta(beta_curr, z_curr, b_curr, X, y, id, sigma0)
  logpi_prop <- logpost_beta(beta_prop, z_curr, b_curr, X, y, id, sigma0)
  # acceptance probability
  log_alpha <- logpi_prop - logpi_curr

  if (log(runif(1)) < log_alpha) {
    return(beta_prop)   # accept
  } else {
    return(beta_curr)   # reject
  }
}

# log posterior for b
logpost_b <- function(b, beta, z, X, y, id, sigma2_b) {
  loglik  <- loglik_hier(beta, z, b, X, y, id)
  # independent prior for each b_k
  logprior <- sum(dnorm(b, mean = 0, sd = sqrt(sigma2_b), log = TRUE))
  loglik + logprior
}

# a single update for b
mh_update_b <- function(b_curr, beta_curr, z_curr,
                        X, y, id, n_id,
                        sigma2_b,
                        tau_b = 0.1) {
  b_new <- b_curr
  # store the times of acceptance
  acc_vec <- integer(n_id)

  for (k in sample.int(n_id)) {
    bk_curr <- b_new[k]
    bk_prop <- rnorm(1, mean = bk_curr, sd = tau_b)
    b_prop <- b_new
    b_prop[k] <- bk_prop

    logpi_curr <- logpost_b(b_new, beta_curr, z_curr, X, y, id, sigma2_b)
    logpi_prop <- logpost_b(b_prop, beta_curr, z_curr, X, y, id, sigma2_b)

    # log acceptance prob
    log_alpha <- logpi_prop - logpi_curr

    if (log(runif(1)) < log_alpha) {
      b_new[k]  <- bk_prop
      acc_vec[k] <- acc_vec[k] + 1
    }
  }

  list(b = b_new, acc = acc_vec)
}
```

```r
# a single update for sigma(conjugate)
update_sigma2_b <- function(b_curr, a0 = 0.001, b0 = 0.001) {
  n_id  <- length(b_curr)
  a_post <- a0 + n_id / 2
  b_post <- b0 + 0.5 * sum(b_curr^2)
  1 / rgamma(1, shape = a_post, rate = b_post)
}

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

  # log-likelihood as a function of z
  loglik_given_z <- function(z_vec) {
    loglik_hier(beta = beta_curr, z = z_vec, b = b_curr,
                X = X, y = y, id = id)
  }

  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_j from Bern
    z_curr[j] <- rbinom(1, size = 1, prob = p1)
  }

  return(z_curr)
}

run_mcmc_hier <- function(X, y, id, sigma0 = 100,
                          n_iter = 5000, burn_in = 1000,
                          tau_beta = 0.05, tau_b = 0.10,
                          a0 = 0.001, b0 = 0.001) {
  set.seed(111)

  N <- nrow(X)
  p <- ncol(X)
  n_id <- max(id)

  # initialization
  beta_curr <- rep(0, p)
  z_curr <- rep(1, p)
  sigma2_b <- 1
  b_curr <- rnorm(n_id, 0, sqrt(sigma2_b))

  # matrices to store samples
  B_save <- matrix(NA, nrow = n_iter, ncol = p)
  Z_save <- matrix(NA, nrow = n_iter, ncol = p)
  sigma2_b_save <- numeric(n_iter)

  # to store the number of accept
  acc_beta <- 0
  acc_b    <- integer(n_id)

  for (iter in 1:n_iter) {

    # update beta
    beta_new <- mh_update_beta(beta_curr, z_curr, b_curr,
                               X, y, id,
                               sigma0 = sigma0,
                               tau_beta = tau_beta)
    if (!all(beta_new == beta_curr)) acc_beta <- acc_beta + 1
    beta_curr <- beta_new

    # update b
    tmp_b <- mh_update_b(b_curr, beta_curr, z_curr,
                         X, y, id, n_id,
                         sigma2_b = sigma2_b,
                         tau_b = tau_b)
    b_curr <- tmp_b$b
    acc_b <- acc_b + tmp_b$acc

    # update sigma_b^2
    sigma2_b <- update_sigma2_b(b_curr, a0 = a0, b0 = b0)

    # update z
    z_curr <- flip_z_hier(z_curr, beta_curr, b_curr, X, y, id)

    B_save[iter, ] <- beta_curr
    Z_save[iter, ] <- z_curr
    sigma2_b_save[iter] <- sigma2_b

    if (iter %% 1000 == 0) {
      cat("iter =", iter, "/", n_iter, "\n")
    }
  }

  keep <- (burn_in + 1):n_iter

  list(
    B = B_save[keep, , drop = FALSE],
    Z = Z_save[keep, , drop = FALSE],
    sigma2_b = sigma2_b_save[keep],
    acc_rate_beta = acc_beta / n_iter,
    acc_rate_b = acc_b / n_iter
  )
}
```

```r
library(coda)
```

```r
t_mh <- system.time(
  res_hier <- run_mcmc_hier(X, y, id, n_iter = 15000, burn_in = 5000)
)
```

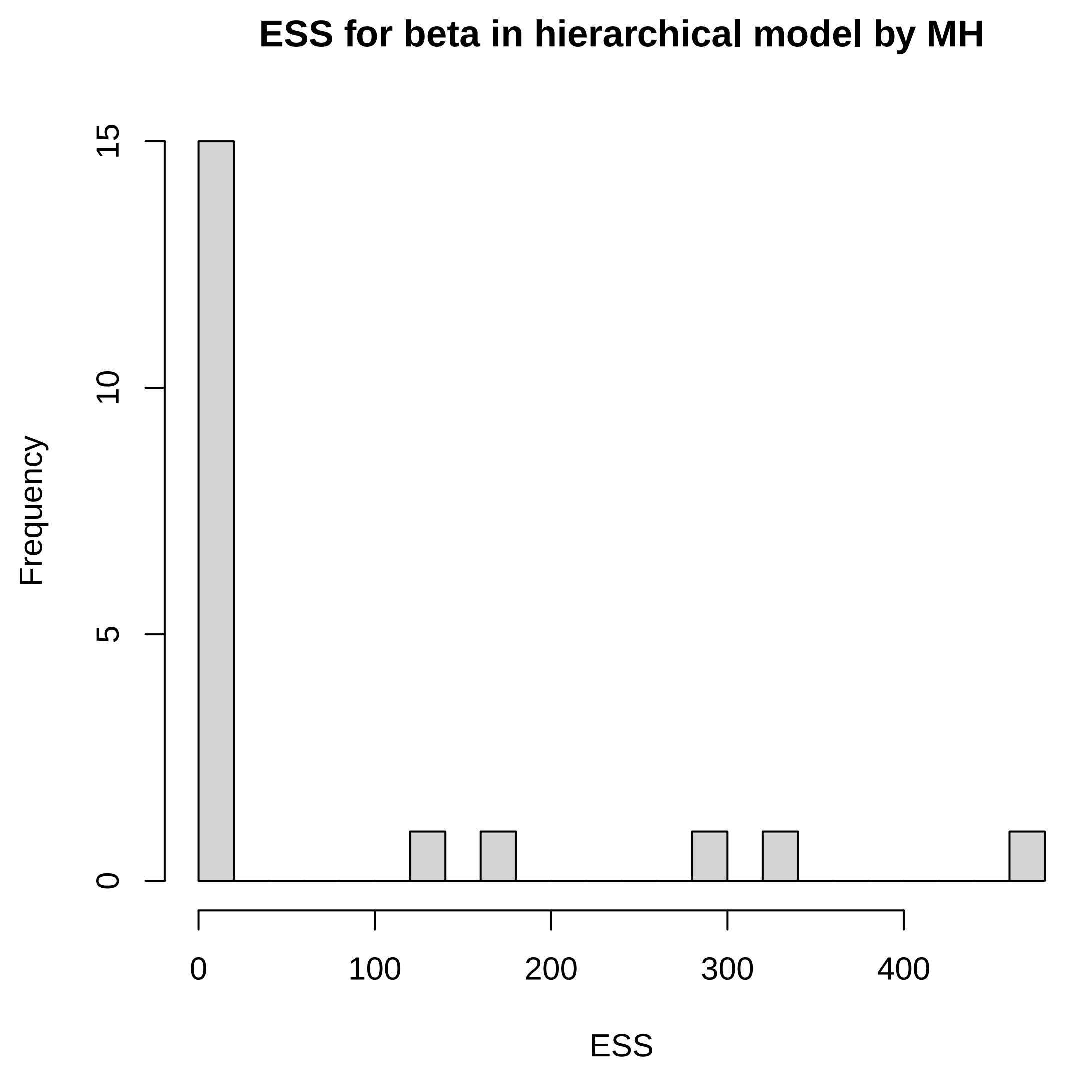
```
## iter = 1000 / 15000
## iter = 2000 / 15000
## iter = 3000 / 15000
## iter = 4000 / 15000
## iter = 5000 / 15000
## iter = 6000 / 15000
## iter = 7000 / 15000
## iter = 8000 / 15000
## iter = 9000 / 15000
## iter = 10000 / 15000
## iter = 11000 / 15000
## iter = 12000 / 15000
## iter = 13000 / 15000
## iter = 14000 / 15000
## iter = 15000 / 15000
```

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

```
## elapsed
## 631.877
```

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

```r
ess_beta_mh <- effectiveSize(as.mcmc(beta_mh))
hist(ess_beta_mh, breaks=20, main = "ESS for beta in hierarchical model 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 95% credible interval coverage for sigma
CI_sigma_mh <- quantile(res_hier$sigma2_b, c(0.025, 0.975))
in_CI_sigma_mh <- (sigma_b_true^2 >= CI_sigma_mh[1]) & (sigma_b_true^2 <= CI_sigma_mh[2])
in_CI_sigma_mh
```

```
## 2.5%
## TRUE
```

```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)

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

TPR
```

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

```r
FPR
```

```
## [1] 0.06666667
```

Now we perform PG

```r
library(BayesLogit)
library(MASS)

# update (omega, beta, b) by PG, given z and sigma2_b
pg_update_beta_b <- function(beta_curr, b_curr, z_curr,
                             X, Z_mat, y, sigma0 = 100, sigma2_b) {
  n    <- nrow(X)
  p    <- ncol(X)
  n_id <- ncol(Z_mat)

  # construct the prior precision matrix
  prec_beta <- rep(1 / sigma0^2, p)
  prec_b    <- rep(1 / sigma2_b, n_id)
  Sigma0_inv <- diag(c(prec_beta, prec_b))

  # sample latent variable
  beta_eff <- beta_curr * z_curr
  eta      <- as.vector(X %*% beta_eff + Z_mat %*% b_curr)
  omega    <- BayesLogit::rpg(num = n, h = 1, z = eta)

  # compute the corresponding values for sampling coefs
  kappa <- y - 0.5
  X_z     <- sweep(X, 2, z_curr, "*")
  X_tilde <- cbind(X_z, Z_mat)
  WX    <- X_tilde * omega
  XtWX  <- t(X_tilde) %*% WX
  V_om  <- solve(XtWX + Sigma0_inv)
  m_om  <- V_om %*% (t(X_tilde) %*% kappa)
  # sample from MVN
  theta_new <- as.vector(
    MASS::mvrnorm(1, mu = as.vector(m_om), Sigma = V_om)
  )
  beta_new <- theta_new[1:p]
  b_new    <- theta_new[(p + 1):(p + n_id)]

  list(beta = beta_new, b = b_new)
}


run_mcmc_pg_hier <- function(X, y, id,
                             sigma0  = 100,
                             n_iter  = 5000,
                             burn_in = 1000,
                             a0 = 0.001, b0 = 0.001) {
  set.seed(111)

  n <- nrow(X)
  p <- ncol(X)
  n_id <- max(id)

  # design matrix for random intercepts
  Z_mat <- model.matrix(~ factor(id) - 1)

  # initialization
  beta_curr <- rep(0, p)
  z_curr <- rep(1, p)
  sigma2_b <- 1
  b_curr <- rnorm(n_id, 0, sqrt(sigma2_b))

  # store samples
  B_save <- matrix(NA, nrow = n_iter, ncol = p)
  Z_save <- matrix(NA, nrow = n_iter, ncol = p)
  sigma2_b_save <- numeric(n_iter)

  for (iter in 1:n_iter) {

    # (omega, beta, b) block via PG
    theta_new <- pg_update_beta_b(beta_curr, b_curr, z_curr,
                                  X, Z_mat, y,
                                  sigma0   = sigma0,
                                  sigma2_b = sigma2_b)
    beta_curr <- theta_new$beta
    b_curr <- theta_new$b

    # update sigma_b^2
    sigma2_b <- update_sigma2_b(b_curr, a0 = a0, b0 = b0)

    # update z
    z_curr <- flip_z_hier(z_curr, beta_curr, b_curr, X, y, id)

    # save
    B_save[iter, ] <- beta_curr
    Z_save[iter, ] <- z_curr
    sigma2_b_save[iter] <- sigma2_b

    if (iter %% 1000 == 0) {
      cat("iter =", iter, "/", n_iter, "\n")
    }
  }

  keep <- (burn_in + 1):n_iter

  list(
    B = B_save[keep, , drop = FALSE],
    Z = Z_save[keep, , drop = FALSE],
    sigma2_b = sigma2_b_save[keep]
  )
}
```

```
t_pg <- system.time(
  res_pg_hier <- run_mcmc_pg_hier(X, y, id, n_iter  = 15000, burn_in = 5000)
)
```

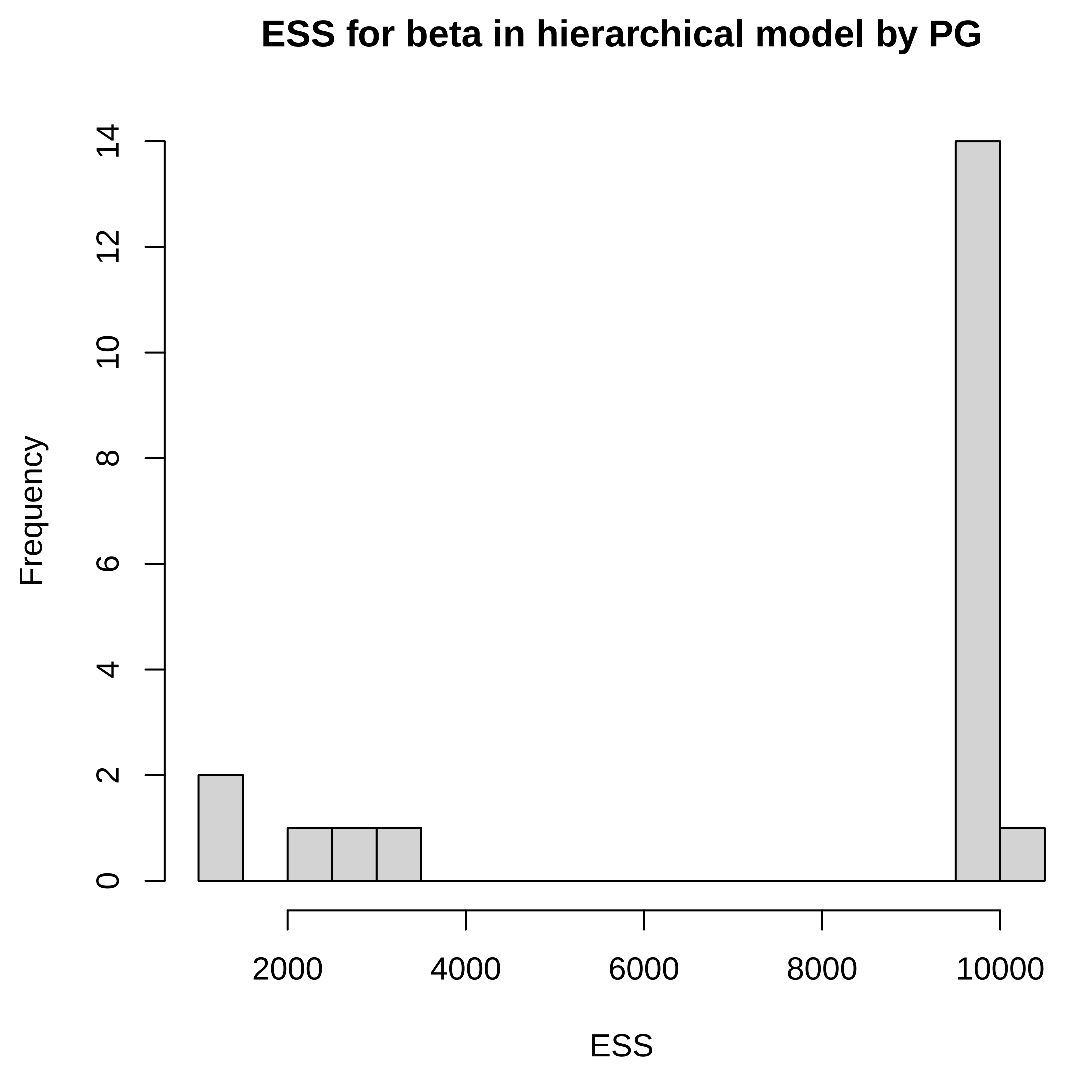
```
## iter = 1000 / 15000
## iter = 2000 / 15000
## iter = 3000 / 15000
## iter = 4000 / 15000
## iter = 5000 / 15000
## iter = 6000 / 15000
## iter = 7000 / 15000
## iter = 8000 / 15000
## iter = 9000 / 15000
## iter = 10000 / 15000
## iter = 11000 / 15000
## iter = 12000 / 15000
## iter = 13000 / 15000
## iter = 14000 / 15000
## iter = 15000 / 15000
```

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

```
##  elapsed
## 2222.032
```

```
# extract the result
beta_pg <- res_pg_hier$B
Z_pg <- res_pg_hier$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 in hierarchical model by PG", xlab = "ESS")
```



```r
# 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
```

```r
# compute 95% credible interval coverage for sigma
CI_sigma_pg <- quantile(res_pg_hier$sigma2_b, c(0.025, 0.975))
in_CI_sigma_pg <- (sigma_b_true^2 >= CI_sigma_pg[1]) & (sigma_b_true^2 <= CI_sigma_pg[2])
in_CI_sigma_pg
```

```
## 2.5%
## TRUE
```

```
# 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)  # true positive rate
FPR <- FP / (FP + TN)  # false positive rate

TPR
```

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

```
FPR
```

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

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

  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]
  }

  for (k in 1:n_id) {
    b[k] ~ dnorm(0.0, tau_b)
  }

  tau_b ~ dgamma(0.001, 0.001)
  sigma_b <- 1 / sqrt(tau_b)
}
"
```

```
library(rjags)
```

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

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

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

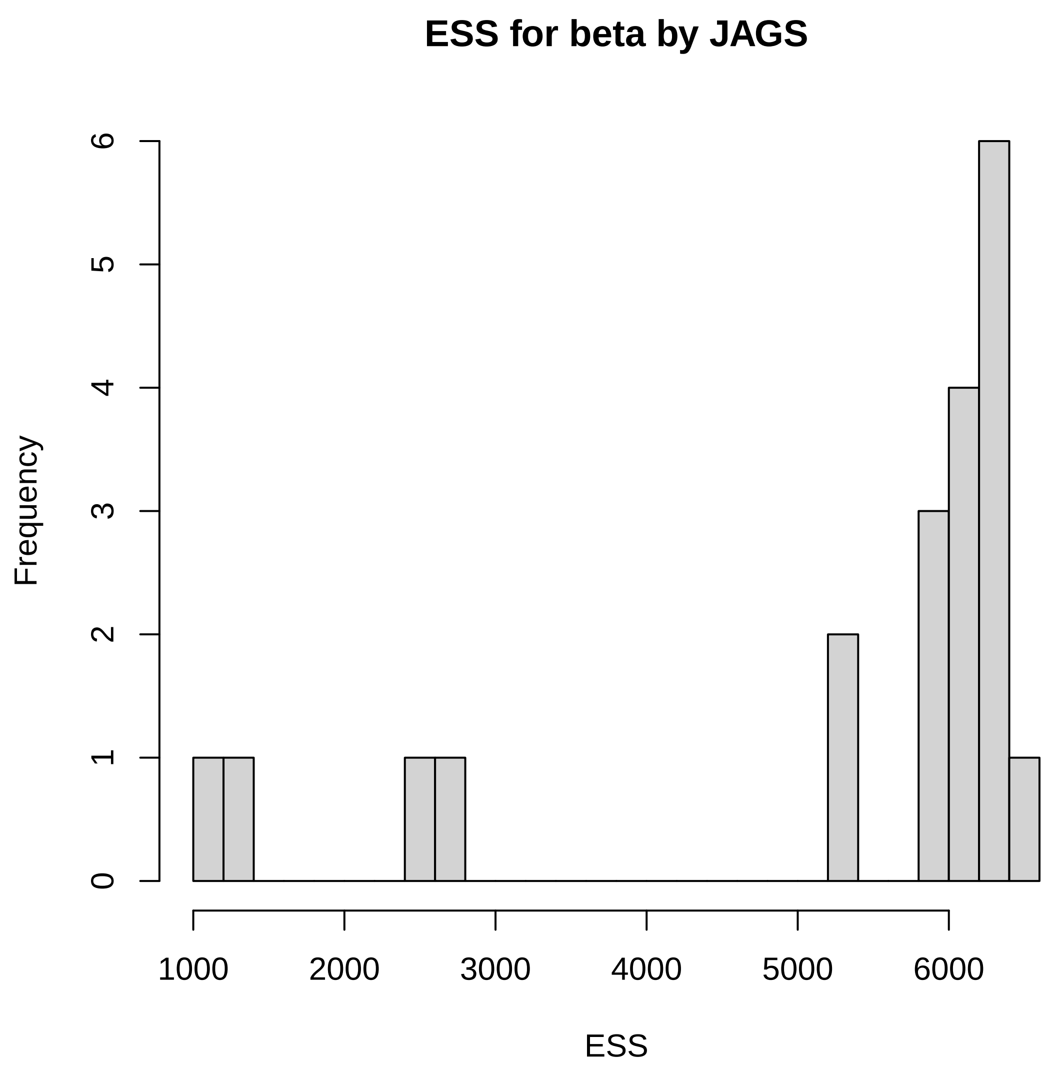
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 3032
##    Unobserved stochastic nodes: 341
##    Total graph size: 88300
##
## Initializing model
```

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

```
## elapsed
## 590.334
```

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

```r
# 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")
```



```
# 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
```

```
# compute 95% credible interval coverage for sigma
CI_sigma_jags <- quantile(s[,41], c(0.025, 0.975))
in_CI_sigma_jags <- (sigma_b_true >= CI_sigma_jags[1]) & (sigma_b_true <= CI_sigma_jags[2])
in_CI_sigma_jags
```

```
##  2.5%
## FALSE
```

```
# 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
```

```
FPR
```

```
## [1] 0.06666667
```

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

  # 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]
  }

  # random effects
  for (k in 1:n_id) {
    b[k] ~ dnorm(0.0, tau_b)
  }

  # hyperprior
  tau_b ~ dgamma(0.001, 0.001)
  sigma_b <- 1 / sqrt(tau_b)
}
"
```

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

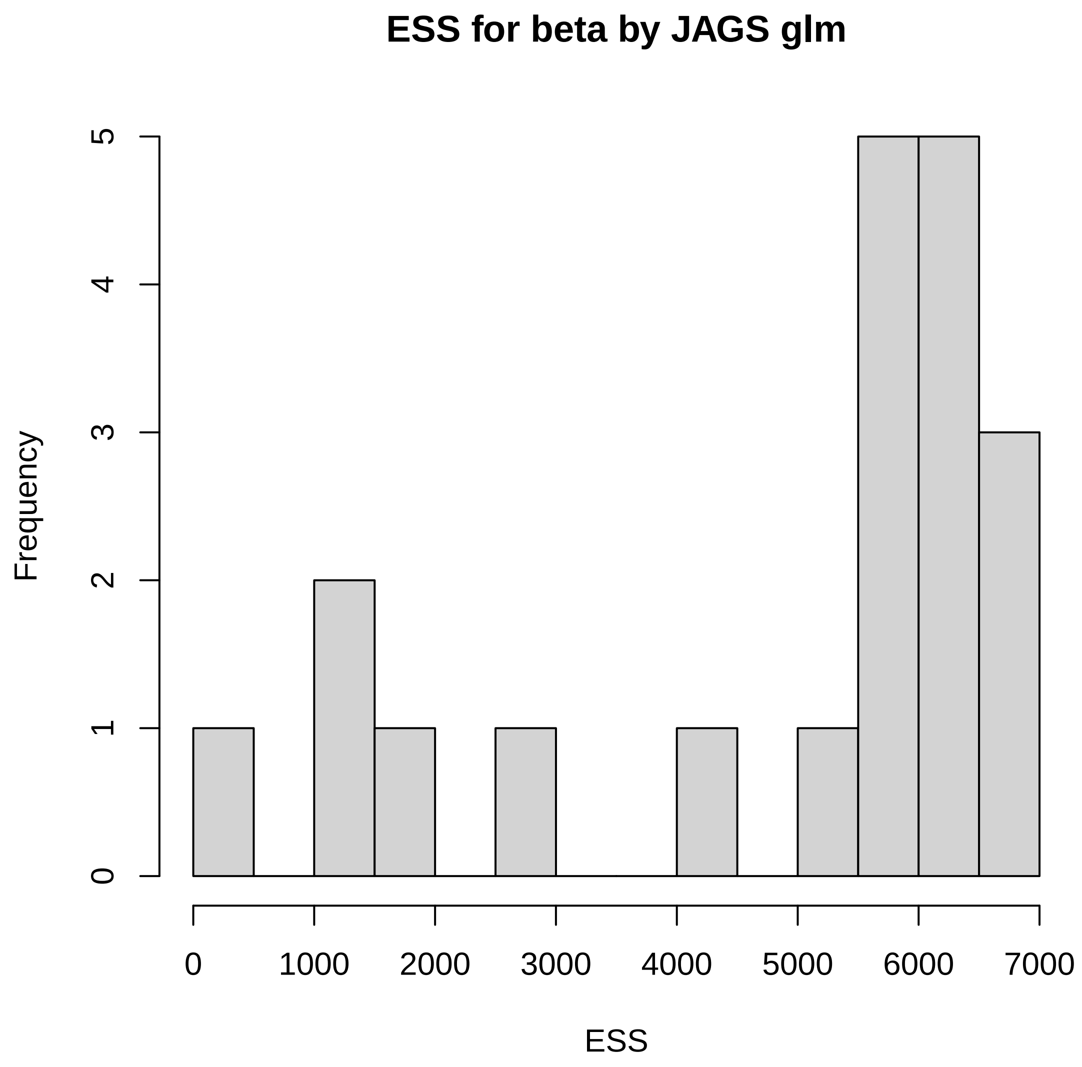
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 3032
##    Unobserved stochastic nodes: 341
##    Total graph size: 79204
##
## Initializing model
```

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

```
## elapsed
## 491.259
```

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

```
# 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")
```



```r
# 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
```

```r
# compute 95% credible interval coverage for sigma
CI_sigma_jags_glm <- quantile(s[,41], c(0.025, 0.975))
in_CI_sigma_jags_glm <- (sigma_b_true >= CI_sigma_jags_glm[1]) & (sigma_b_true <= CI_sigma_jags_glm[2])
in_CI_sigma_jags_glm
```

```
##  2.5%
## FALSE
```

```r
# 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)

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

TPR
```

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

```r
FPR
```

```
## [1] 0.06666667
```