HW2

Kelly Li

2/22/2022

Bayesian Probit Regression

1)

Let's compute our target distribution:

$$\begin{aligned} p(\vec{\beta}|\vec{y}) &\propto p(\vec{y}|\vec{\beta})p(\vec{\beta}) \\ &= \prod_{i=1}^{n}(p_i^{y_i}(1-p_i)^{1-y_i})\exp[-\frac{1}{2}\vec{\beta}^T(100(X^TX)^{-1})^{-1}\vec{\beta}] \end{aligned}$$

Our MCMC generates symmetric proposals about the current β:

$$\hat{\beta}_{i+1} \sim N(\beta_i, 10^2(X^TX)^{-1})$$

With acceptance criteria:

$$u \sim U(0,1)$$

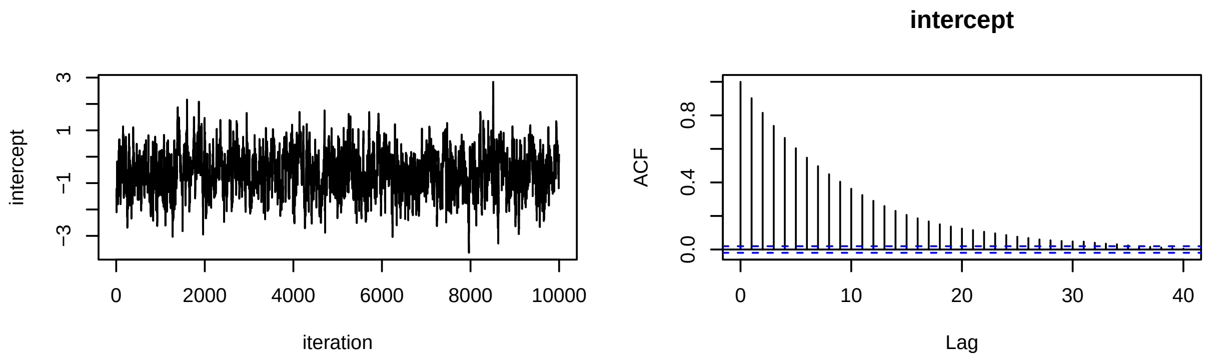
$$\alpha = \frac{p(\hat{\beta}_{i+1}|\vec{y})}{p(\beta_i|\vec{y})}$$

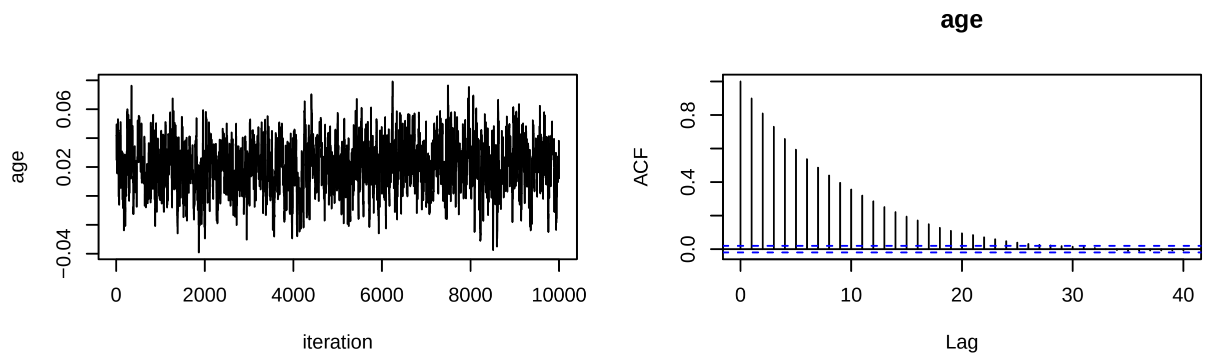
$$\beta_{i+1} = \begin{cases} \hat{\beta}_{i+1} & u \leq \alpha \\ \beta_i & u > \alpha \end{cases}$$

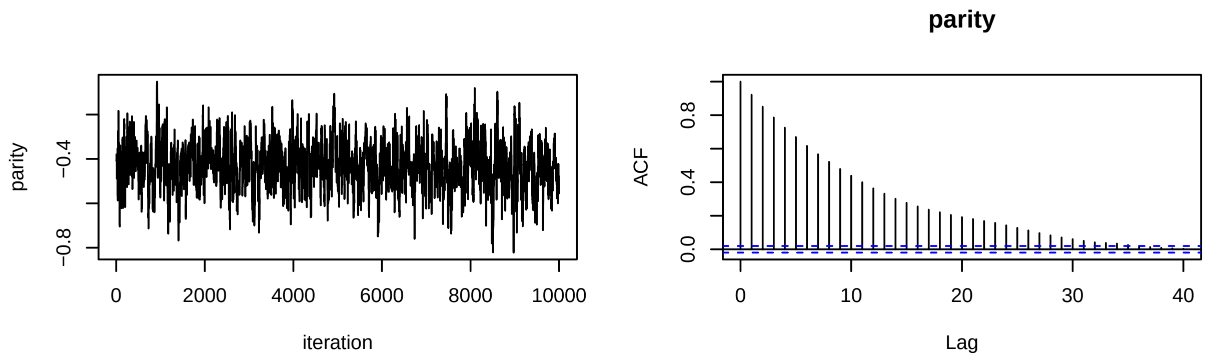
```r
data <- infert
y <- infert$case
x <- cbind(rep(1, length(y)), data$age, data$parity,
               as.integer(data$education), data$spontaneous, data$induced)

getLogLik <- function(y, x, xtx, beta){
  p <- pnorm(x%*%beta)
  loglik <- sum(y * log(p) + (1-y)*log(1-p)) - 1/(200) * t(beta) %*% xtx %*% beta
  return(loglik)
}

adaptive.mh <- function(n.sim, burn, x, y, beta0){
  #basic math and matrix setups
  n <- length(y)
  p <- ncol(x)
  xtx <- t(x) %*% x
  xtx.inv <- solve(xtx)
  #chain information
  n.total <- n.sim*(1.0 + burn)
  n.burn <- n.sim*burn

  #initialize chain
  beta.out <- matrix(NA, nrow = n.sim, ncol = p)
  beta <- beta0
  for(i in 1:n.total){
    beta_new <- rmvnorm(1, mean = beta,
                        sigma = xtx.inv) %>% t()
    loglik_new <- getLogLik(y, x, xtx, beta_new)
    loglik_old <- getLogLik(y, x, xtx, beta)
    logdiff <- loglik_new - loglik_old
    #accept-reject
    if(log(runif(1)) < logdiff){
      beta <- beta_new
    }
    if(i > n.burn){
      i1 <- i - n.burn
      beta.out[i1, ] <- beta
    }
  }
  colnames(beta.out) <- c("intercept", "age", "parity",
                          "education", "spontaneous", "induced")
  return(beta.out)
}

beta0 <- solve(t(x) %*% x) %*% t(x) %*% y

mh.out <- adaptive.mh(10000, 0.1, x, y, beta0)

mean <- colMeans(mh.out)
sd <- apply(mh.out, 2, sd)
cbind(mean, sd) %>% kable(linesep = "")
```

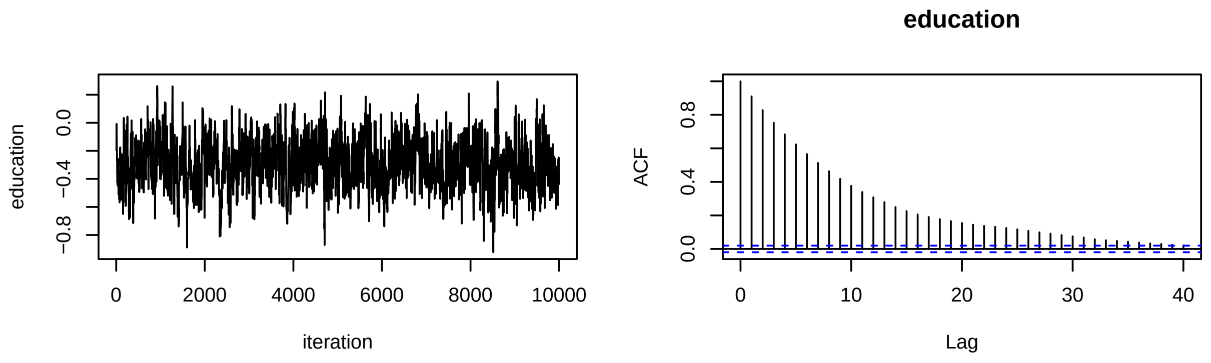
	mean	sd
intercept	-0.6029083	0.8039954
age	0.0205784	0.0172019
parity	-0.4315151	0.1056423
education	-0.2911854	0.1697412
spontaneous	1.1494598	0.1674838
induced	0.7048107	0.1599721

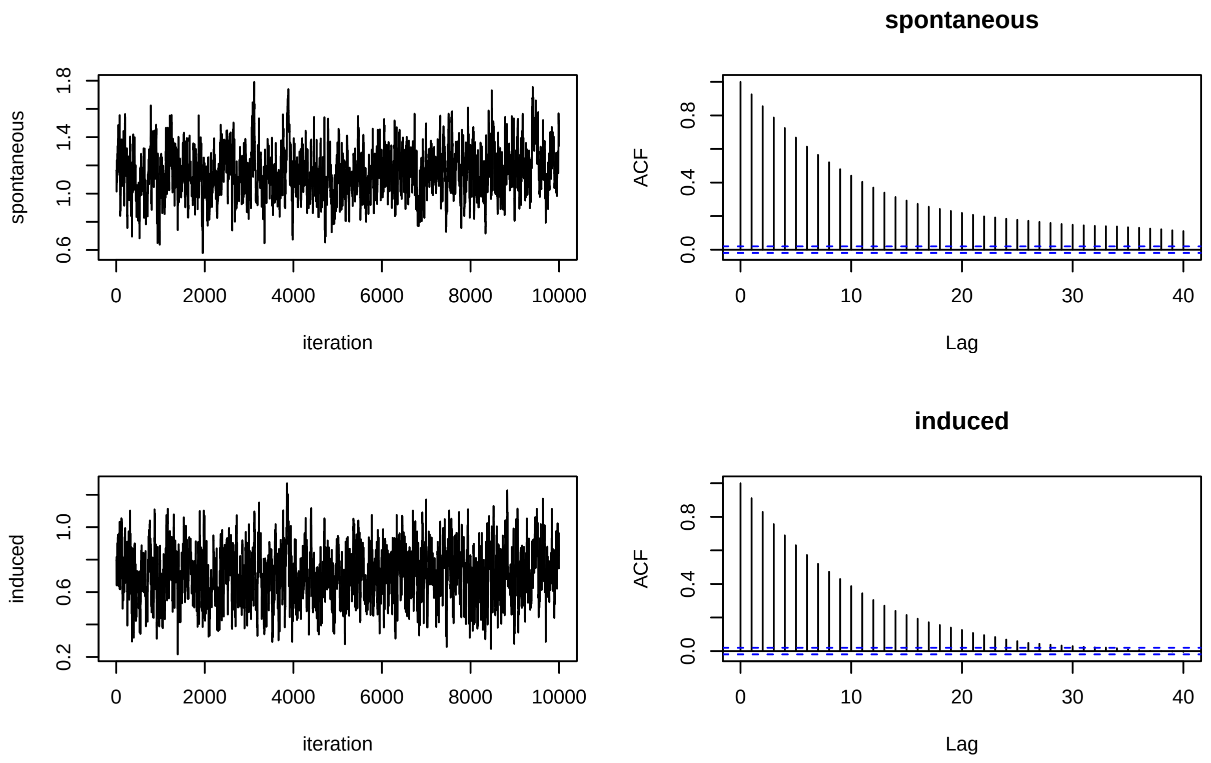
```
par(mfrow = c(4,2))
for(j in 1:6){
  matplot(c(1:10000), mh.out[,j], type = "l",
          xlab = "iteration", ylab = colnames(mh.out)[j])
  acf(mh.out[, j], main = colnames(mh.out)[j])
}
```


intercept
3
1
−1
−3
0
2000
4000
6000
8000
10000
iteration
ACF
0.0
0.4
0.8
0
10
20
30
40
Lag


age
0.06
0.02
−0.04
0
2000
4000
6000
8000
10000
iteration
ACF
0.0
0.4
0.8
0
10
20
30
40
Lag


parity
−0.4
−0.8
0
2000
4000
6000
8000
10000
iteration
ACF
0.0
0.4
0.8
0
10
20
30
40
Lag


education
0.0
−0.4
−0.8
0
2000
4000
6000
8000
10000
iteration
ACF
0.0
0.4
0.8
0
10
20
30
40
Lag


spontaneous
spontaneous
0.6
1.0
1.4
1.8
0
2000
4000
6000
8000
10000
iteration
ACF
0.0
0.4
0.8
0
10
20
30
40
Lag
induced
induced
0.2
0.6
1.0
0
2000
4000
6000
8000
10000
iteration
ACF
0.0
0.4
0.8
0
10
20
30
40
Lag

2)

Let $z_i \sim^{iid} N(\vec{x_i}^T\vec{\beta}, 1)$. Now:

$P(y_i = 1|\vec{x_i}, \vec{\beta}) = P(\mathbf{1}(z_i > 0) = 1) = P(z_i = 0)$

$$\begin{aligned}
P(\vec{\beta}|\vec{z}, \vec{y}) &\propto p(\vec{y}, \vec{z}|\vec{\beta})p(\vec{\beta}) \\
&\propto p(z|\vec{\beta})p(\vec{\beta}) \\
&= N(X\vec{\beta}, I)N(0, 100(X^TX)^{-1}) \\
&\propto \exp[-\frac{1}{2}(\vec{z} - X\vec{\beta})^T(\vec{z} - X\vec{\beta})]\exp[-\frac{1}{2}\vec{\beta}^T(100(X^TX)^{-1})^{-1}\vec{\beta}] \\
&\propto \exp[-\frac{1}{2}(\vec{\beta}(\frac{100}{101}(X^TX)^{-1})^{-1}\vec{\beta} - 2\vec{\beta}^TX^T\vec{z})]
\end{aligned}$$

By definition, $\vec{\beta}|\vec{z}, \vec{y} \sim N(\frac{100}{101}(X^TX)^{-1}X^T\vec{z}, \frac{100}{101}(X^TX)^{-1})$. Also:

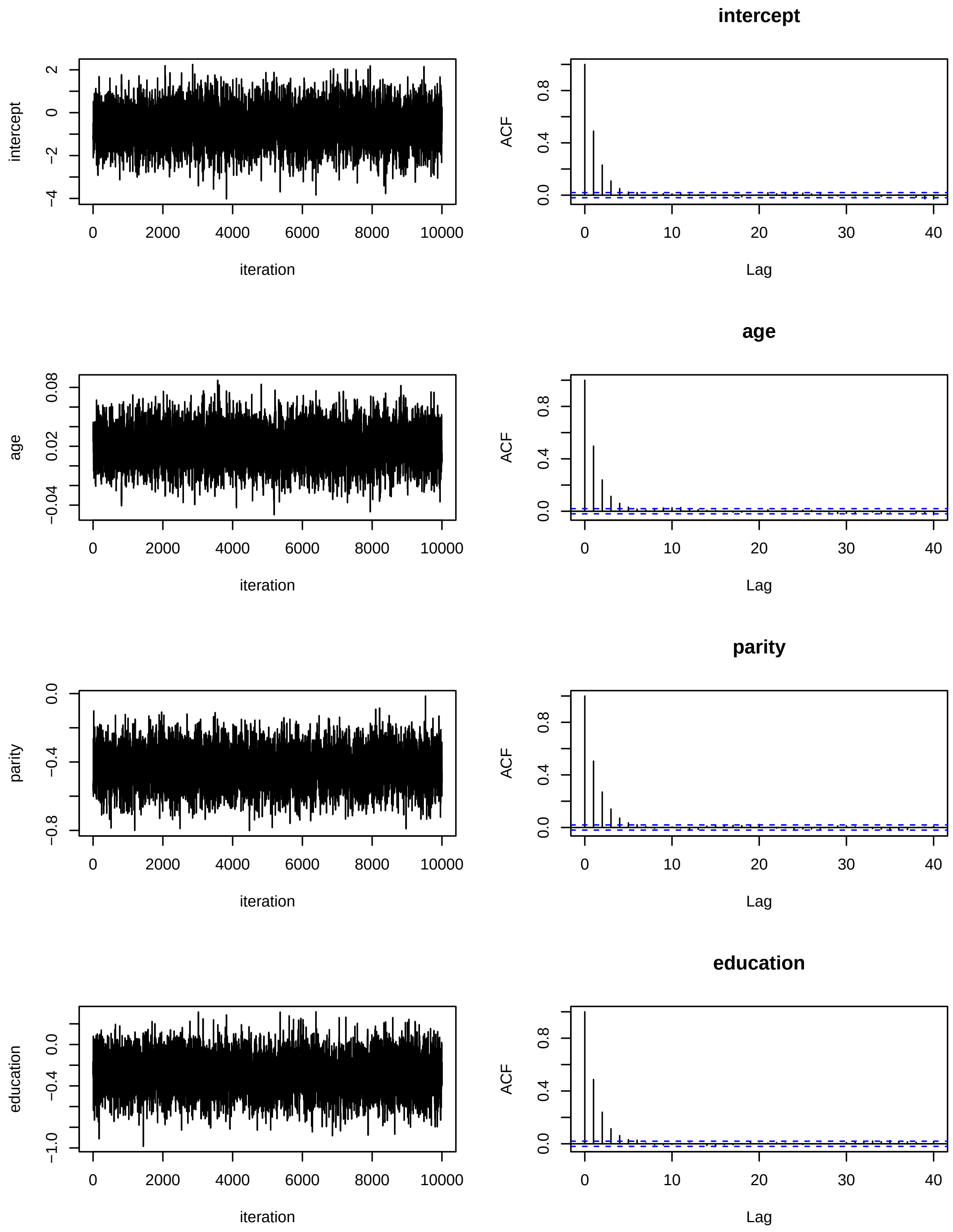
$$\begin{aligned}
z_i|\vec{\beta}, y_i &\propto p(y_i|\vec{\beta}, z_i)p(z_i|\vec{\beta}) \\
&= \begin{cases} N(\vec{x_i}^T\vec{\beta}, 1) * \mathbf{1}_{[0,\infty)}(z_i) & y_i = 1 \\ N(\vec{x_i}^T\vec{\beta}, 1) * \mathbf{1}_{(-\infty,0]}(z_i) & y_i = 0 \end{cases}
\end{aligned}$$

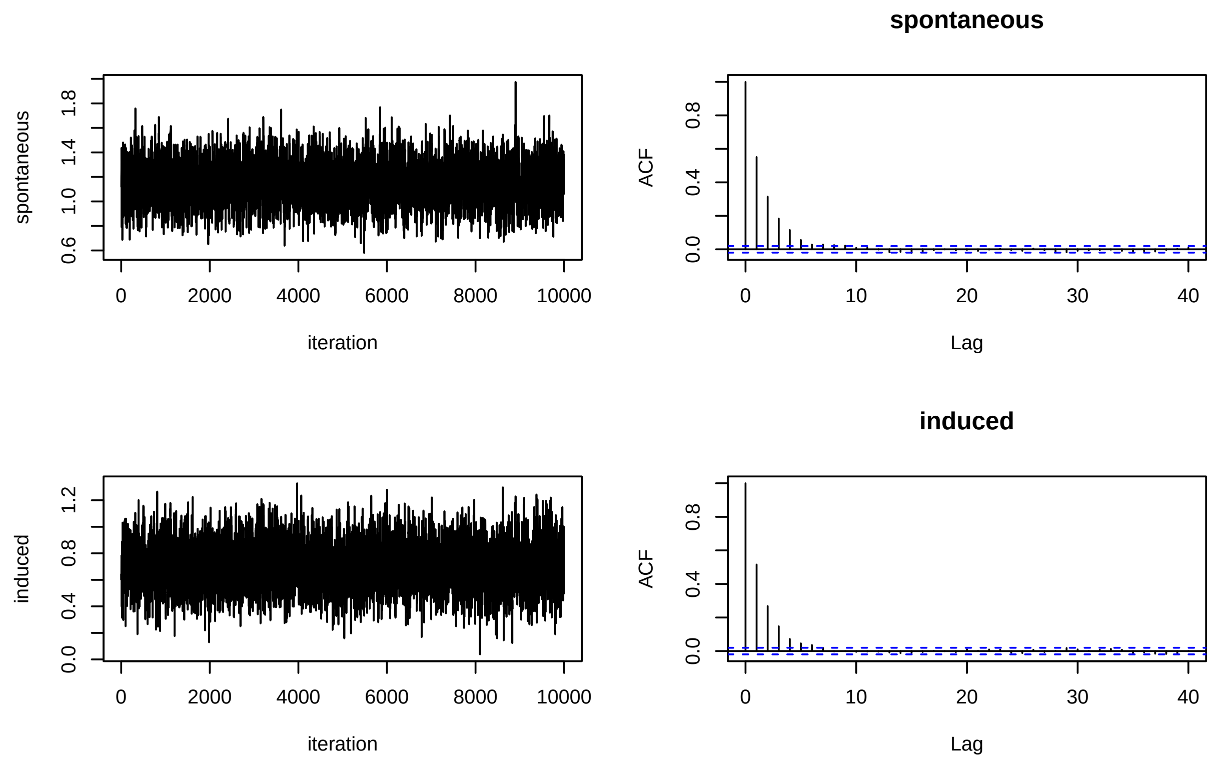
Then, we can implement DA-MCMC by Gibbs sampling $p(\vec{\beta}, \vec{z}, \vec{y})$ and $p(\vec{z}|\vec{\beta}, \vec{y})$:

```r
da.mcmc <- function(n.sim, burn, x, y, beta0){
  #matrix stuff
  n <- length(y)
  p <- ncol(x)
  xtx <- t(x) %*% x
  xtx.inv <- solve(xtx)
  var <- 100/101 * xtx.inv

  #chain information
  n.total <- n.sim*(1.0 + burn)
  n.burn <- n.sim*burn

  #initialize chain
  z <- rmvnorm(1, x%*%beta0, diag(1, nrow = n))
  beta <- beta0
  beta.out <- matrix(NA, nrow = n.sim, ncol = p)

  #beta
  for(i in 1:n.total){
    beta <- rmvnorm(1, var %*% t(x) %*% t(z), var) %>% t()
  #z
    for(j in 1:n){
      if(y[j] == 1){
      z[j] <- rtruncnorm(1, a = 0, b = Inf, mean = x[j,] %*% beta, sd = 1)
      } else{
      z[j] <- rtruncnorm(1, a = -Inf, b = 0, mean = x[j,] %*% beta, sd = 1)
      }
    }
  #store variables after burn
    if(i > n.burn){
      i1 <- i - n.burn
      beta.out[i1, ] <- beta
    }
  }

  colnames(beta.out) <- c("intercept", "age", "parity",
                          "education", "spontaneous", "induced")
  return(beta.out)
}

da.mcmc.out <- da.mcmc(10000, 0.1, x, y, beta0)

mean <- colMeans(da.mcmc.out)
sd <- apply(da.mcmc.out, 2, sd)
cbind(mean, sd) %>% kable(linesep = "")
```

	mean	sd
intercept	-0.6091844	0.8319951
age	0.0203811	0.0183088
parity	-0.4365080	0.1033902
education	-0.2840048	0.1711240
spontaneous	1.1528047	0.1626955
induced	0.7073339	0.1664085

```
par(mfrow = c(4,2))
for(j in 1:6){
  matplot(c(1:10000), da.mcmc.out[,j], type = "l",
          xlab = "iteration", ylab = colnames(da.mcmc.out)[j])
  acf(da.mcmc.out[, j], main = colnames(da.mcmc.out)[j])
}
```


intercept
age
parity
education
iteration
Lag
ACF


spontaneous
spontaneous
0.6
1.0
1.4
1.8
0
2000
4000
6000
8000
10000
iteration
ACF
0.0
0.4
0.8
0
10
20
30
40
Lag
induced
induced
0.0
0.4
0.8
1.2
0
2000
4000
6000
8000
10000
iteration
ACF
0.0
0.4
0.8
0
10
20
30
40
Lag

3)

Let's introduce another parameter $\alpha^2 \sim$ Inverse Gamma(a, b). Let $W_i = \alpha Z_i$. We now sample from the following distributions using Gibbs:

$$\vec{\beta}|\vec{w}, \alpha, \vec{y} \sim N(\frac{101}{100}(X^TX)^{-1}\frac{X^t\vec{w}}{\alpha}, \frac{101}{100}(X^TX)^{-1}))$$

$$w_i|\vec{y}\vec{\beta}\alpha = \begin{cases} N(\alpha\vec{x_i}^T\vec{\beta}, \alpha^2) * \mathbf{1}_{[0,\infty)}(w_i) & y_i = 1 \\ N(\alpha\vec{x_i}^T\vec{\beta}, \alpha^2) * \mathbf{1}_{(-\infty,0]}(w_i) & y_i = 0 \end{cases}$$

$$p(\alpha^2|\vec{w}, \vec{\beta}, \vec{y}) \propto \exp[-\frac{1}{2}(\frac{\vec{w}^t\vec{w}}{\alpha^2} - \frac{2\vec{\beta}^TX^T\vec{w}}{\alpha})]{\alpha^2}^{-a-1}\exp[-\frac{b}{\alpha^2}]$$

$$\log p(\alpha^2|\vec{w}, \vec{\beta}, \vec{y}) \propto -\frac{1}{2}\frac{\vec{w}^T\vec{w}}{\alpha^2} + \frac{\vec{\beta}^TX^T\vec{w}}{\alpha} - (a+1)\log(\alpha^2) - \frac{b}{\alpha^2}$$

$$\log p(\alpha^2) \propto -(a+1)\log(\alpha^2) - \frac{b}{\alpha^2}$$

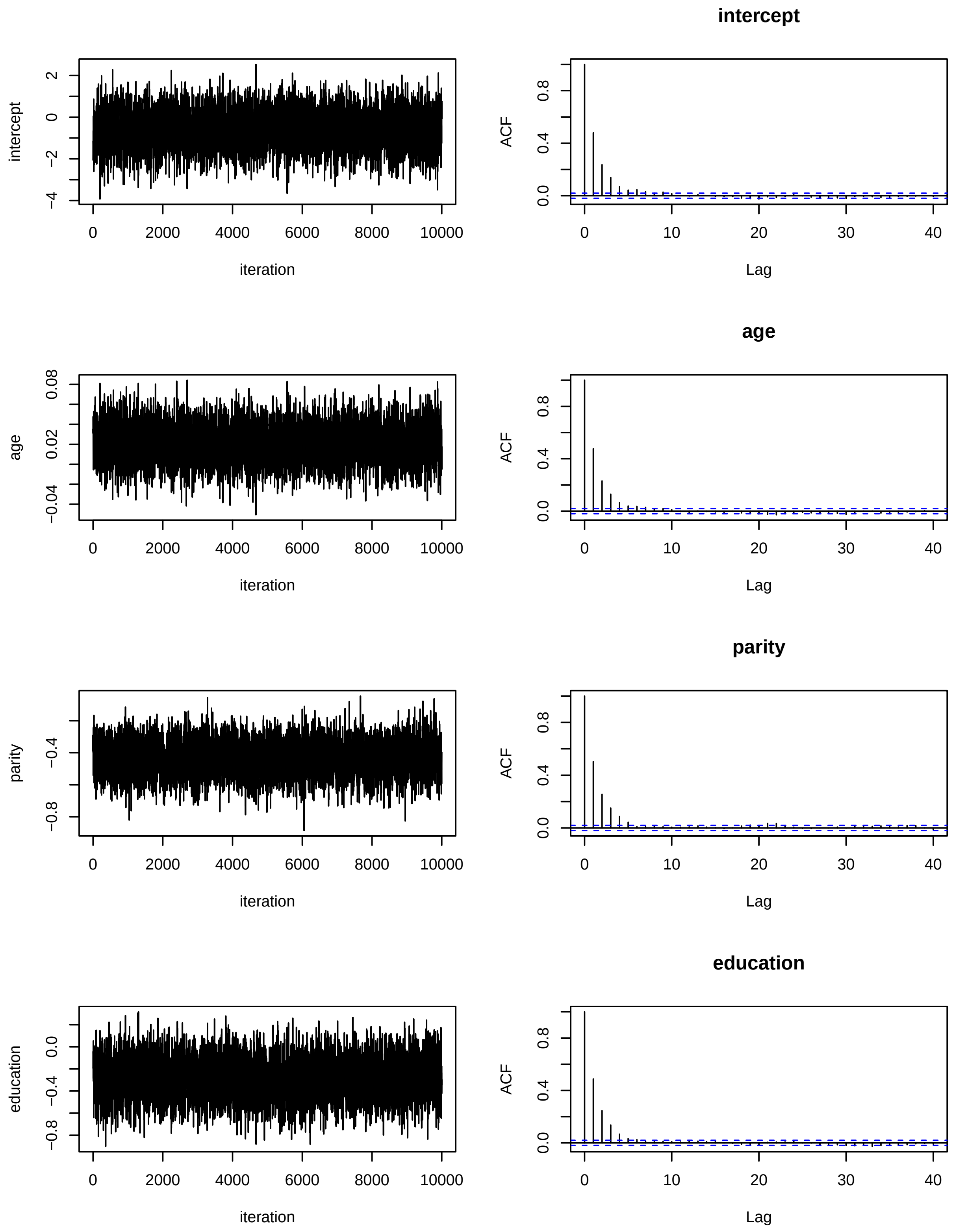
To sample using Metropolis-Hastings for α^2, we sample from the centered distribution with some tuning parameter c:

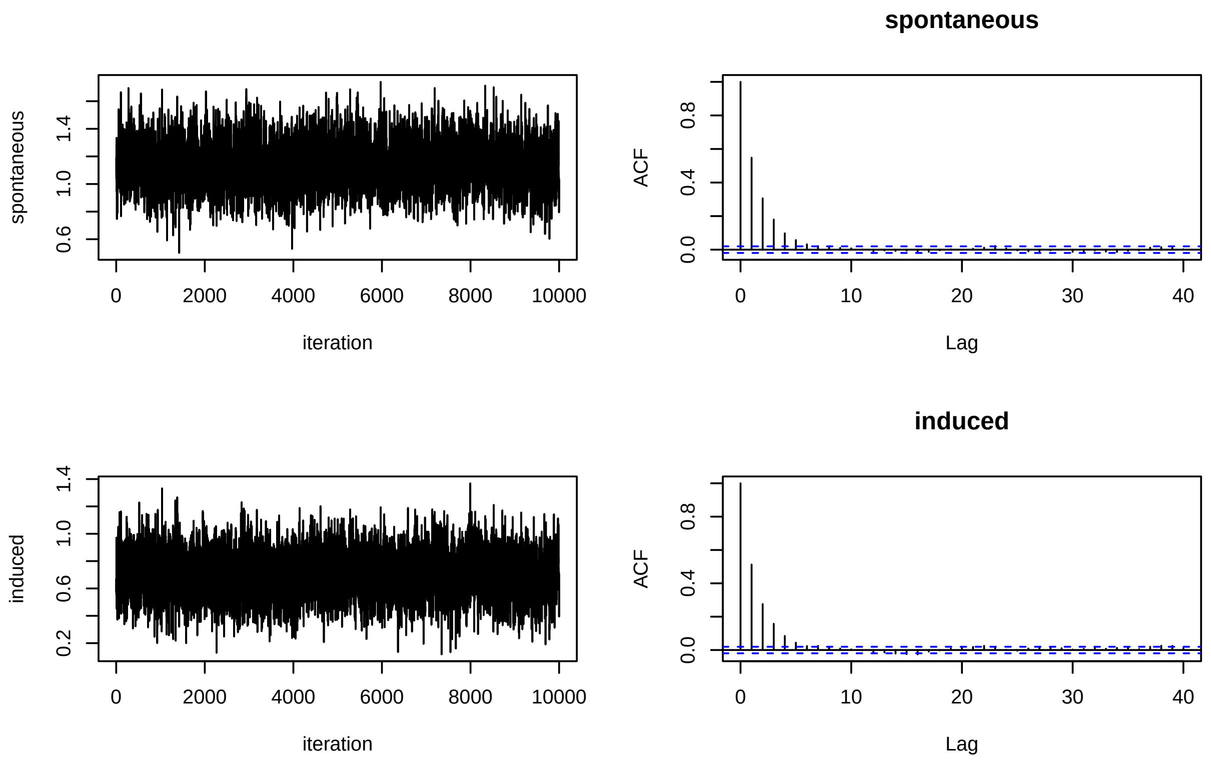
$$\alpha_{i+1} \sim \text{Inverse Gamma}(c, \alpha_i^2(c+1))$$

```r
getLogLikIG <- function(alpha2, a, b){
  loglik <- (-a + 1) * log(alpha2) - b/alpha2
  return(loglik)
}
getLogLikAlpha <- function(x, alpha2, w, beta, a, b){
  loglik = -1/2 * sum(w^2)/alpha2 + t(beta) %*% t(x) %*% w/sqrt(alpha2) - (a + 1)*log(alpha2) - b/alpha
  return(loglik)
}

px.da.mcmc <- function(n.sim, burn, x, y, a, b, beta0, c){
  #matrix stuff
  n <- length(y)
  p <- ncol(x)
  xtx <- t(x) %*% x
  xtx.inv <- solve(xtx)
  var <- 100/101 * xtx.inv

  #chain information
  n.total <- n.sim*(1.0 + burn)
  n.burn <- n.sim*burn

  #initialize chain
  beta <- beta0
  alpha2 <- 1/rgamma(1, shape = a, rate = b)
  w <- sqrt(alpha2) * rmvnorm(1, x%*%beta, diag(1, nrow = n)) %>% t()
  beta.out <- matrix(NA, nrow = n.sim, ncol = p)

  for(i in 1:n.total){
    #beta
    beta <- rmvnorm(1, (var%*% t(x) %*% w)/sqrt(alpha2), var) %>% t()
    #alpha2
    alpha2.prop <- 1/rgamma(1, shape = c, rate = alpha2 * (c+1))
    p.alpha2.prop <- getLogLikAlpha(x, alpha2, w, beta, a, b) +
      getLogLikIG(alpha2, c, alpha2.prop * (c+1)) -
      getLogLikAlpha(x, alpha2.prop, w, beta, a, b) -
      getLogLikIG(alpha2.prop, c, alpha2*(c+1))
    if(log(runif(1)) <= p.alpha2.prop){
      alpha2 <- alpha2.prop
    } else{
      alpha2 <- alpha2
    }
    #w
    for(j in 1:n){
      if(y[j] == 1){
      w[j] <- rtruncnorm(1, a = 0, b = Inf, mean = x[j,] %*% beta, sd = 1)
      } else{
      w[j] <- rtruncnorm(1, a = -Inf, b = 0, mean = x[j,] %*% beta, sd = 1)
      }
    }
    w <- sqrt(alpha2) * w
    #store variables after burn
    if(i > n.burn){
      i1 <- i - n.burn
      beta.out[i1, ] <- beta
    }
  }
  colnames(beta.out) <- c("intercept", "age", "parity",
                          "education", "spontaneous", "induced")
  return(beta.out)
}

px.da.mcmc.out <- px.da.mcmc(10000, 0.1, x, y, 300, 1, beta0, 150)

mean <- colMeans(px.da.mcmc.out)
sd <- apply(px.da.mcmc.out, 2, sd)
cbind(mean, sd) %>% kable(linesep = "")
```

	mean	sd
intercept	-0.6119322	0.8375915
age	0.0205423	0.0180875
parity	-0.4346481	0.1016932
education	-0.2819179	0.1743392
spontaneous	1.1462038	0.1638907
induced	0.7032631	0.1622376

```r
par(mfrow = c(4,2))
for(j in 1:6){
  matplot(c(1:10000), px.da.mcmc.out[,j], type = "l",
          xlab = "iteration", ylab = colnames(px.da.mcmc.out)[j])
  acf(px.da.mcmc.out[, j], main = colnames(px.da.mcmc.out)[j])
}
```


intercept
age
parity
education
iteration
Lag
ACF
0
2000
4000
6000
8000
10000
10
20
30
40
0.0
0.4
0.8
2
−2
−4
0.08
0.02
−0.04
−0.4
−0.8


spontaneous
spontaneous
ACF
iteration
Lag
induced
induced
ACF
iteration
Lag
0.6
1.0
1.4
0.2
0.6
1.0
1.4
0.0
0.4
0.8
0
2000
4000
6000
8000
10000
0
10
20
30
40

4)

```r
adaptive.mh.ram <- peakRAM(adaptive.mh(10000, 0.1, x, y, beta0))
da.mcmc.ram <- peakRAM(da.mcmc(10000, 0.1, x, y, beta0))
px.da.mcmc.ram <- peakRAM(px.da.mcmc(10000, 0.1, x, y, 300, 1, beta0, 150))

benchmark <- rbind(adaptive.mh.ram,
                   da.mcmc.ram,
                   px.da.mcmc.ram)

colnames(benchmark) <- c("Function", "Elapsed time (sec)",
                         "Total RAM used (MiB)", "Peak RAM used (MiB)")
benchmark$Function <- c("AMH", "DA-MCMC", "PX-DA-MCMC")

benchmark %>% kable(linesep = "")
```

Function	Elapsed time (sec)	Total RAM used (MiB)	Peak RAM used (MiB)
AMH	3.55	0.6	32.5
DA-MCMC	22.64	0.5	37.1
PX-DA-MCMC	23.36	0.5	37.1

All three algorithms converged to similar coefficients with 10,000 simulations and 1,000 burn-in iterations. However, AMH had significantly worse mixture than the other two algorithms, based off the ACF plots. Regarding coding complexity, AMH was the simplest to implement, with the lowest efficiency, followed by DA-MCMC and then PX-DA-MCMC. On run-time and RAM usage, AMH had the lowest of both, with the other two algorithms very similar in both time and storage, but higher.In our case, all three algorithms still converged and DA-MCMC / PX-DA-MCMC did not have strong differences in mixture, DA-MCMC may be the “best of both worlds” in being relatively straightforward to implement, while giving us an idea of the entire posterior distribution for our parameters of interest.