HW5

1. (a) Sampling a random dataset from the model. Samples of α are generated by sampling from $t_4(0, 1)$ and multiplying the sample by 2

```
J <- 10
x <- runif(J)
alpha <- 2 * rt(1, 4)
beta <- rt(1, 4)
rtpois <- function(n, lambda)
  qpois(runif(n, dpois(0, lambda), 1), lambda)
n <- rtpois(J, 5)
inv_logit <- function(x)
  exp(x)/(1 + exp(x))
theta <- inv_logit(alpha + beta * x)
y <- rbinom(J, n, theta)
```

α is

```
alpha
```

```
[1] -1.902133
```

and β is

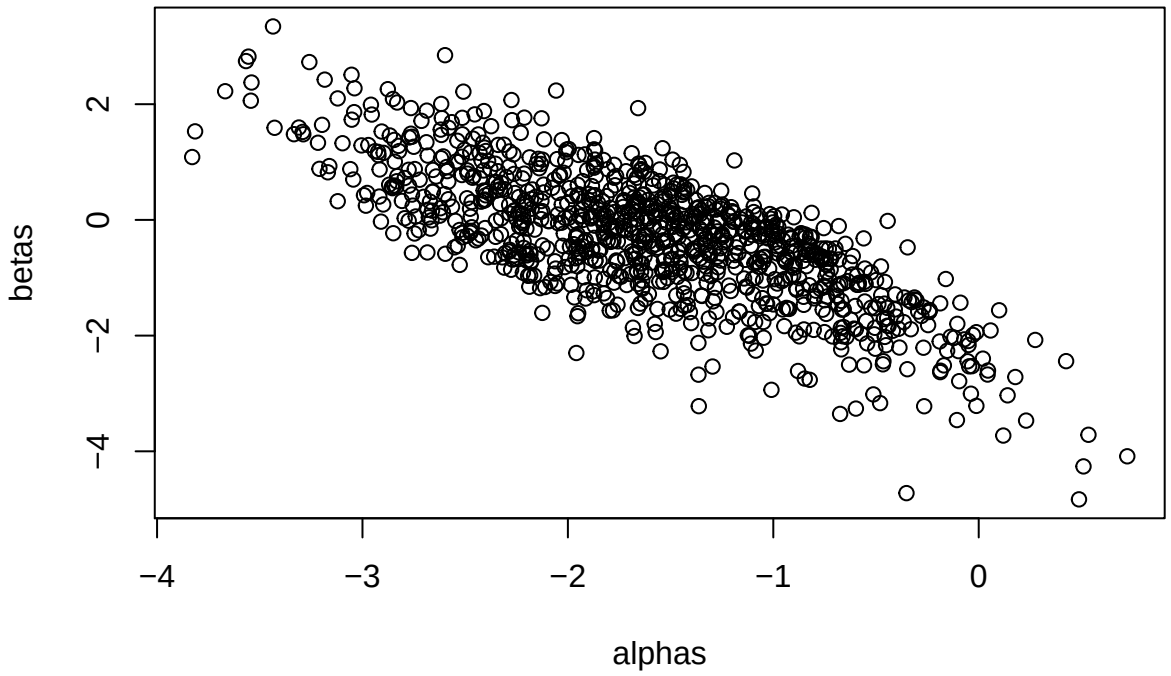
```
beta
```

```
[1] 0.1807983
```

(b) Sampling from the posterior with rejection sampling using the prior for the g distribution. M is chosen adaptively by starting at 1/100000 and then increasing if the acceptance probabilities are too high (i.e. greater than 1).

```
rejection_sampling <- function(samps, y, x, n)
{
  s <- 1
  alphas <- numeric(length = 1000)
  betas <- numeric(length = 1000)
  m <- 100000
  save <- FALSE
  max_lik <- 0
  while(save != TRUE)
  {
    while(s <= samps)
    {
      alpha <- 2 * rt(1, 4)
      beta <- rt(1, 4)
      theta <- inv_logit(alpha + beta * x)
      lik <- prod(dbinom(y, n, theta))
      accept_prob <- m * lik
```

```
      if(lik > max_lik)
      {
        max_lik <- lik
      }
      if(runif(1) < accept_prob)
      {
        alphas[s] <- alpha
        betas[s] <- beta
        s <- s + 1
      }
    }
    if(m * max_lik > 1)
    {
      m <- 1/ max_lik
    }
    else
    {
      save <- TRUE
    }
  }
  list(alphas, betas)
}
samps <- rejection_sampling(1000, y, x, n)
alphas <- unlist(samps[1])
betas <- unlist(samps[2])
plot(alphas, betas)
```

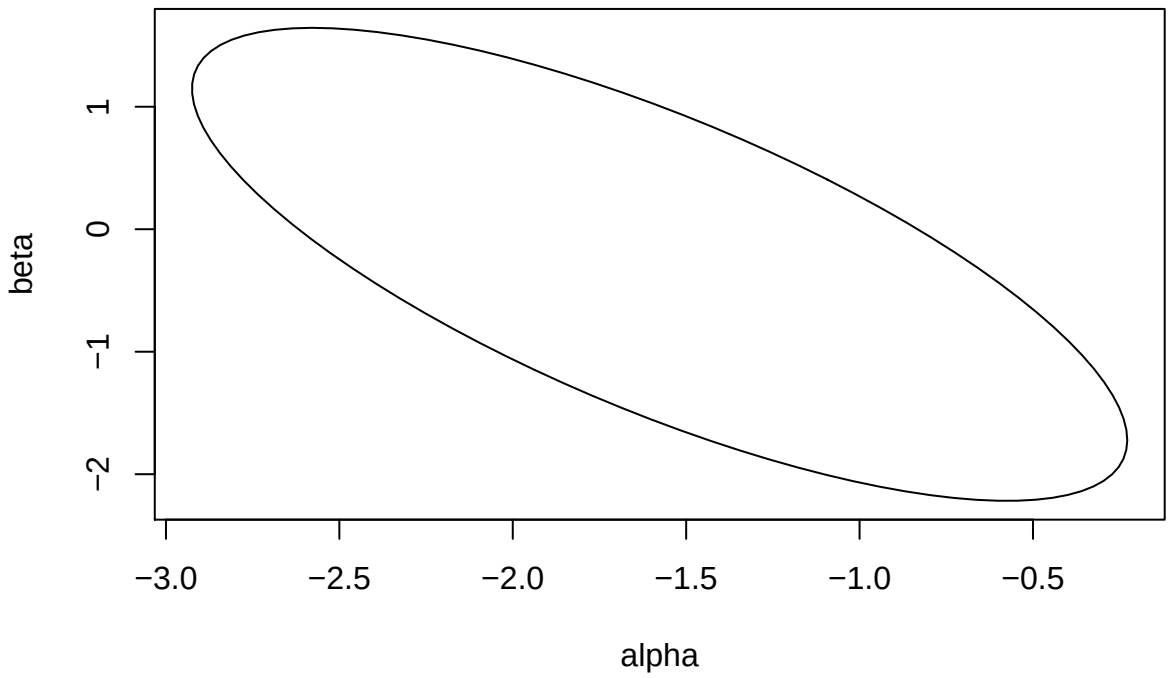


(c) Computing the mode and hessian numerically using nlm. nlm finds the mode by minimizing negative log likelihood

```
log_post <- function(alpha, beta)
{
  theta <- inv_logit(alpha + beta * x)
  (sum(dbinom(y, n, theta, log = TRUE)) + dt(beta, 4, log = TRUE) + dt(alpha/2, 4, log = TRUE))
}

vect_post <- Vectorize(log_post)

nl_post <- function(params)
{
  alpha <- params[1]
  beta <- params[2]
  - log_post(alpha, beta)
}

post_mode <- nlm(nl_post, c(0, 0), hessian = TRUE)

library(ellipse)
```

```
## Warning: package 'ellipse' was built under R version 3.1.2
```

```
cov <- solve(post_mode$hessian)
plot(ellipse(cov, centre = post_mode$estimate), type = 'l', xlab="alpha", ylab="beta",)
points(post_mode$estimate)
```



(d) Importance sampling according to given model

```
library(mvtnorm)
```

```
## Warning: package 'mvtnorm' was built under R version 3.1.3
```

```
imp_samps <- rmvt(1000, sigma = cov, df = 4, delta = post_mode$estimate)
g_theta <- dmvt(imp_samps, sigma = cov, df = 4, delta = post_mode$estimate)
q_theta <- vect_post(imp_samps[, 1], imp_samps[, 2])
imp_weights <- exp(q_theta - g_theta)
exp_alpha <- sum(imp_samps[, 1] * imp_weights)/sum(imp_weights)
exp_beta <- sum(imp_samps[, 2] * imp_weights)/sum(imp_weights)
```

$E(\alpha|y)$

```
exp_alpha
```

```
[1] -1.601107
```

$E(\beta|y)$

```
exp_beta
```

```
[1] -0.3595182
```

2. General metropolis algorithm. Takes an X matrix of any size and a y vector with the same length as the number of rows in X. Transition function is independent normal with mean 0 and variance 1 around current beta vector.

```
log_post <- function(beta, y, X)
{
  log_lik <- sum(dpois(y, exp(X %*% beta), log = TRUE))
  log_prior <- sum(dcauchy(beta, scale = 2.5))
  log_lik + log_prior
}
metropolis <- function(init_beta, n_samps, y, X)
{
  samps <- matrix(nrow = n_samps, ncol = length(init_beta))
  beta <- init_beta
  post <- log_post(beta, y, X)
  for(i in 1:n_samps)
  {
    beta_star <- beta + rnorm(length(beta))
    post_star <- log_post(beta_star, y, X)
    r <- min(exp(post_star - post), 1)
    if(rbinom(1, 1, r))
    {
      beta <- beta_star
      post <- post_star
    }
    samps[i, ] <- beta
  }
  samps
}
```

(b) Simulating fake data according to the model:

```
x1 <- runif(50)
x2 <- runif(50)
x3 <- runif(50)
X <- as.matrix(cbind(x1, x2, x3))
true_beta <- rcauchy(3, scale = 2.5)
y <- rpois(50, exp(X %*% true_beta))
```

Getting 3 chains of 1000 metropolis samples

```
init_beta <- rcauchy(3, scale = 2.5)
met_samps1 <- metropolis(init_beta, 1000, y, X)
init_beta <- rcauchy(3, scale = 2.5)
met_samps2 <- metropolis(init_beta, 1000, y, X)
init_beta <- rcauchy(3, scale = 2.5)
met_samps3 <- metropolis(init_beta, 1000, y, X)
```

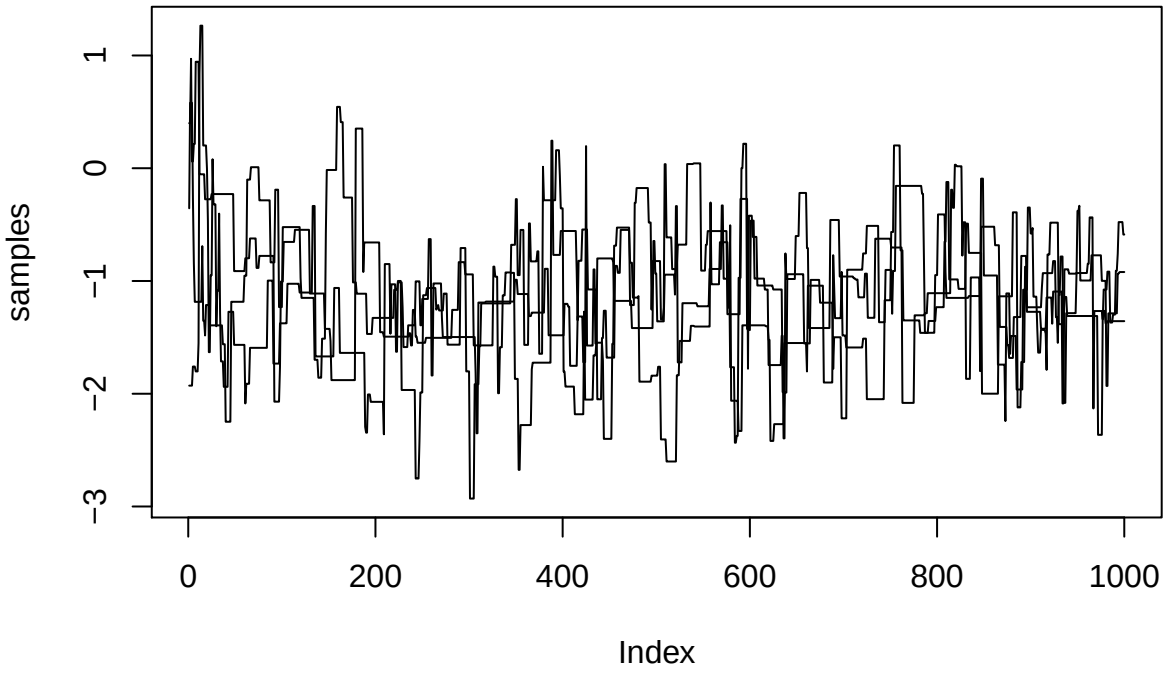
We can now estimate the betas with the posterior mean:

```
good_samples <- rbind(met_samps1[501:1000,], met_samps2[501:1000,], met_samps3[501:1000,])
colMeans(good_samples)
```

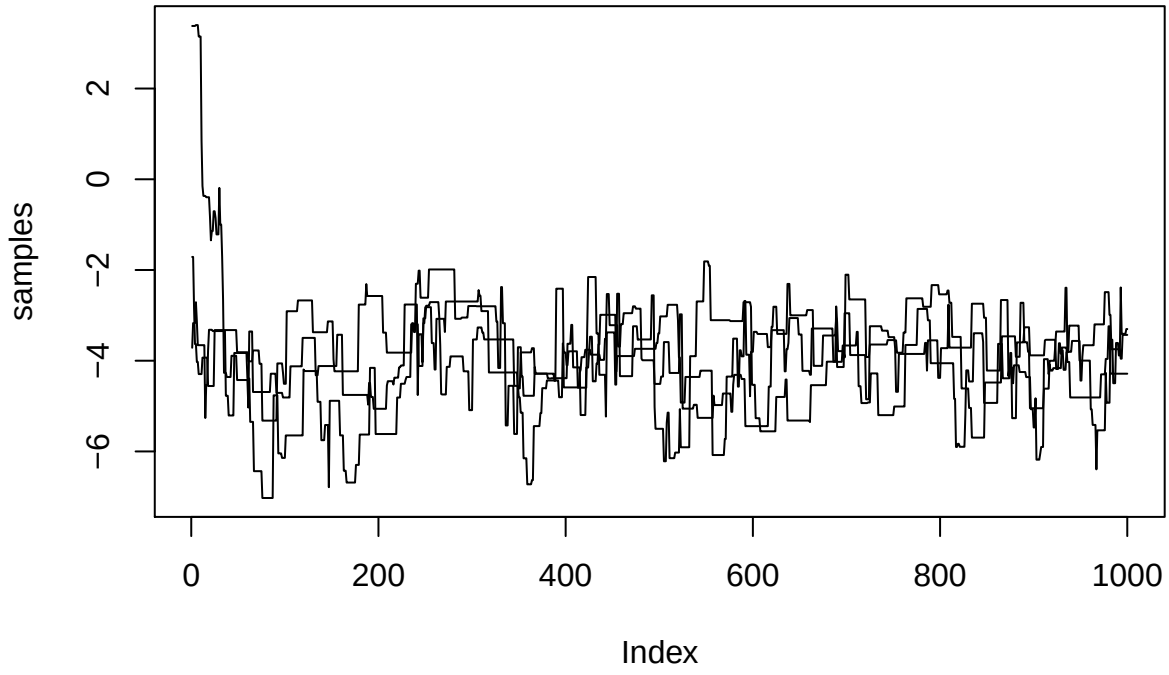
```
## [1] -1.141934 -4.017796  1.677419
```

And plotting chains to see if they properly mixed

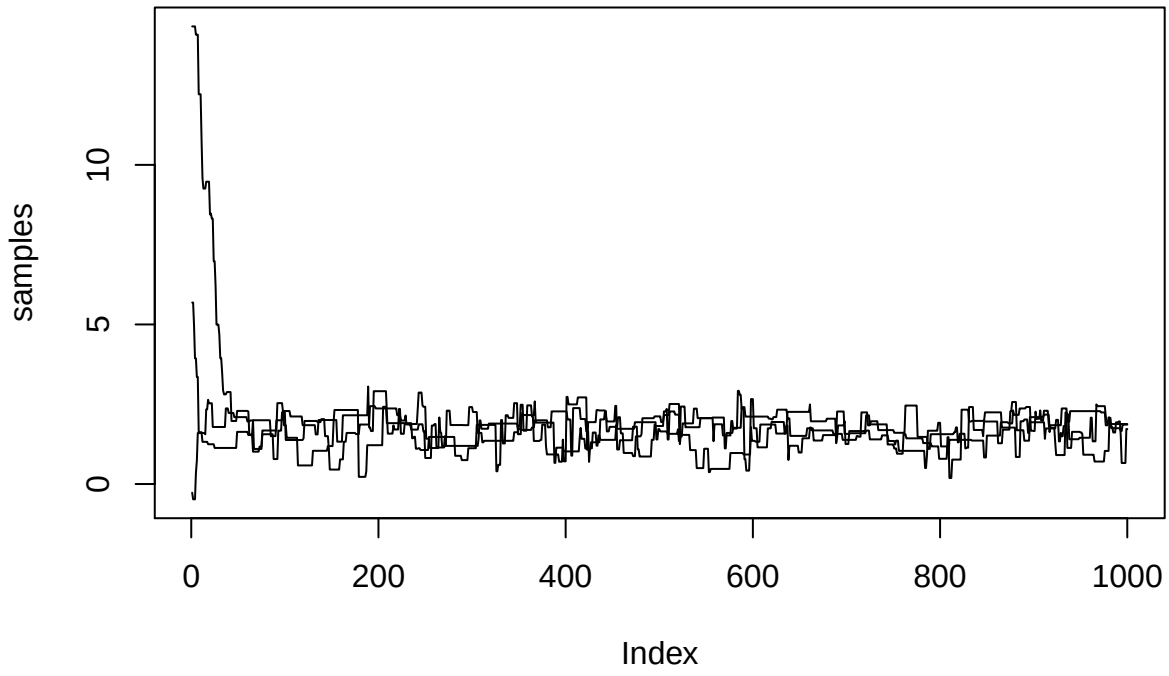
```
yrange <- range(c(met_samps1[, 1], met_samps2[, 1], met_samps3[, 1]))
plot(met_samps1[, 1], type="l", ylim=yrange, ylab="samples")
lines(met_samps2[, 1], type="l", ylim=yrange)
lines(met_samps3[, 1], type="l", ylim=yrange)
```



```
yrange <- range(c(met_samps1[, 2], met_samps2[, 2], met_samps3[, 2]))
plot(met_samps1[, 2], type="l", ylim=yrange, ylab="samples")
lines(met_samps2[, 2], type="l", ylim=yrange)
lines(met_samps3[, 2], type="l", ylim=yrange)
```




```
yrange <- range(c(met_samps1[, 3], met_samps2[, 3], met_samps3[, 3]))
plot(met_samps1[, 3], type="l", ylim=yrange, ylab="samples")
lines(met_samps2[, 3], type="l", ylim=yrange)
lines(met_samps3[, 3], type="l", ylim=yrange)
```



```
data {
  int n;
  int p;
  matrix[n, p] X;
  int y[n];
}
parameters {
  vector[p] beta;
}
model {
  y ~ poisson(exp(X * beta));
}
```

Stan model with estimates

```
library(rstan)
n = 50
p = 3
fit <- stan("metropolis.stan")
```

Estimates for beta parameters (sorry! I couldn't figure out how to print it better)

```
print(fit, pars = c("beta[1]", "beta[2]", "beta[3]"))
```

```
## Inference for Stan model: metropolis.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##          mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## beta[1] -1.28    0.02 0.62 -2.51 -1.69 -1.28 -0.85 -0.07  1239    1
## beta[2] -4.11    0.03 0.96 -6.14 -4.73 -4.05 -3.43 -2.41  1442    1
## beta[3]  1.75    0.01 0.50  0.72  1.40  1.76  2.10  2.67  1152    1
##
## Samples were drawn using NUTS(diag_e) at Fri Oct 16 09:18:20 2015.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```