STAT221 HW3

Jonathan Luu

Problem 1

Simulate n (say 300) observations for the logistic regression model where Y_i is the binary response, X the observed p-dimensional covariate vector (p=4), and β is a vector of p coefficients (unknown).

```
set.seed(128)
X <- matrix(rnorm(300*4),300,4)
beta <- matrix(rnorm(4, sd=10))
probability <- 1/(1+exp(-X%*%beta))
Y <- rbinom(n=300,size=1,prob=probability)
est.cov <- sqrt(diag(vcov(glm(Y~X, family=binomial))[2:5,2:5]))
beta
```

```
           [,1]
[1,]  1.3992910
[2,]  0.4281307
[3,]  6.6684619
[4,] -2.0624577
```

Part 1a

Write down the posterior distribution of β, up to a normalizing constant.

$$\ell(\beta) \propto \prod_{i=1}^{n} \left[\frac{exp(X_i\beta)}{1+exp(X_i\beta)}\right]^{Y_i} \left[\frac{1}{1+exp(X_i\beta)}\right]^{1-Y_i}$$

$$p(\beta|X) \propto \ell(\beta) * p(\beta)$$

$$\propto \prod_{i=1}^{n} \left[\frac{exp(X_i\beta)}{1+exp(X_i\beta)}\right]^{Y_i} \left[\frac{1}{1+exp(X_i\beta)}\right]^{1-Y_i} \prod_{j=1}^{p} \frac{1}{\sigma_j} exp\left(-\frac{\beta_j^2}{2\sigma_j^2}\right)$$

Part 1b

Write a Metropolis algorithm to simulate β from its posterior distribution.

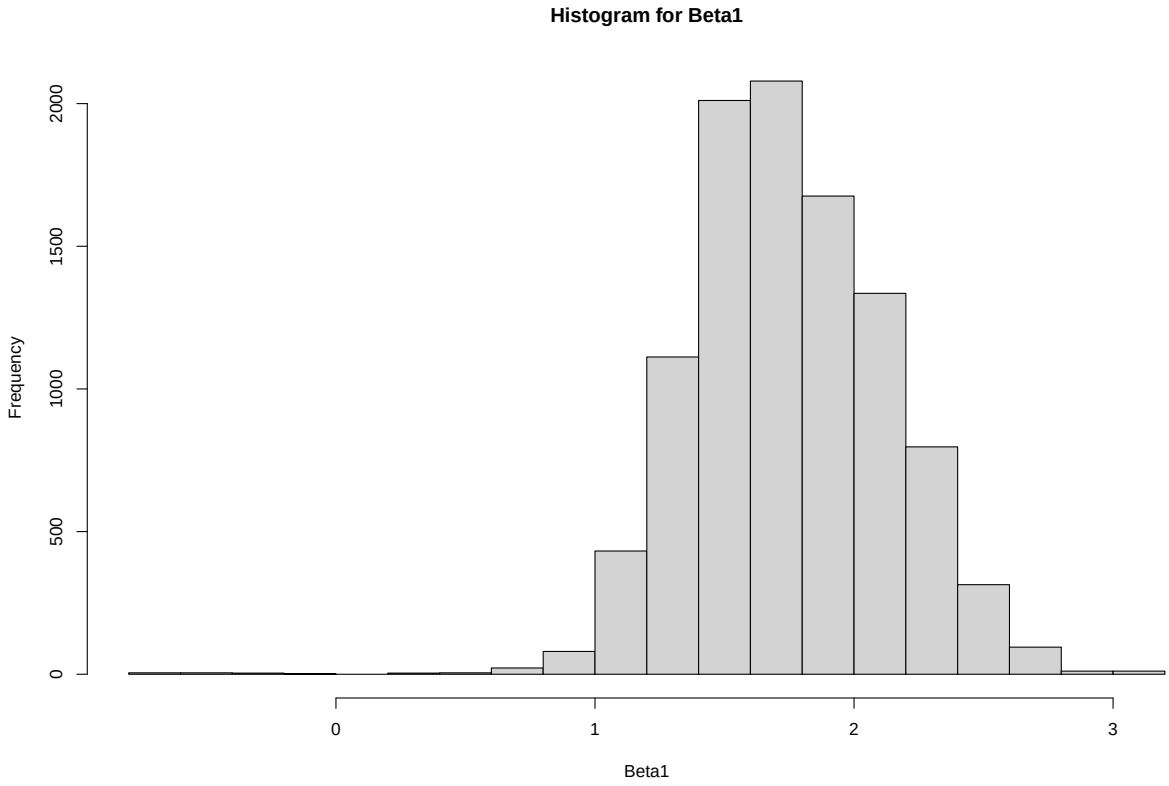
Using a multivariate normal distribution with estimated covariance as my proposal distribution, the algorithm is as follows:

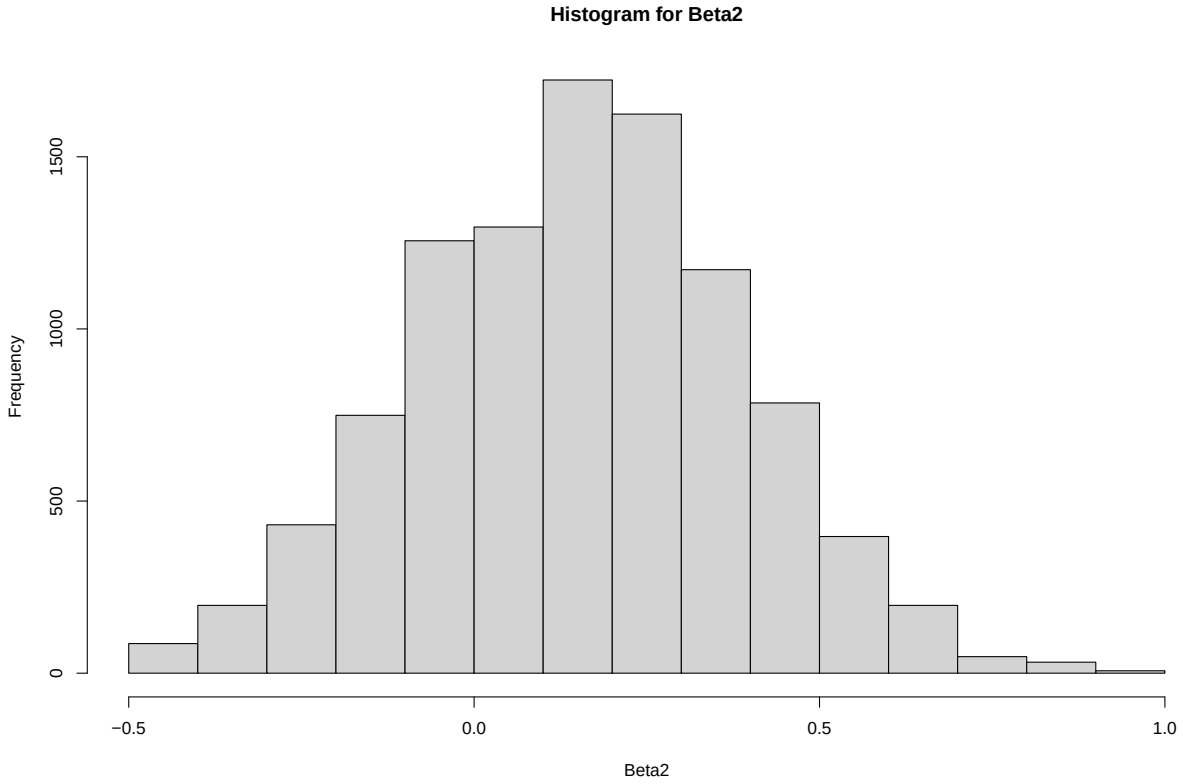
1. Generate a random value from the proposal distribution and add it to the previous x
2. Calculate the acceptance ratio using the posterior distribution above as the acceptance function
3. Generate a random U(0,1) RV and compare it to the acceptance ratio
4. If less than the acceptance ratio, accept the new value of y, else keep the old value of x

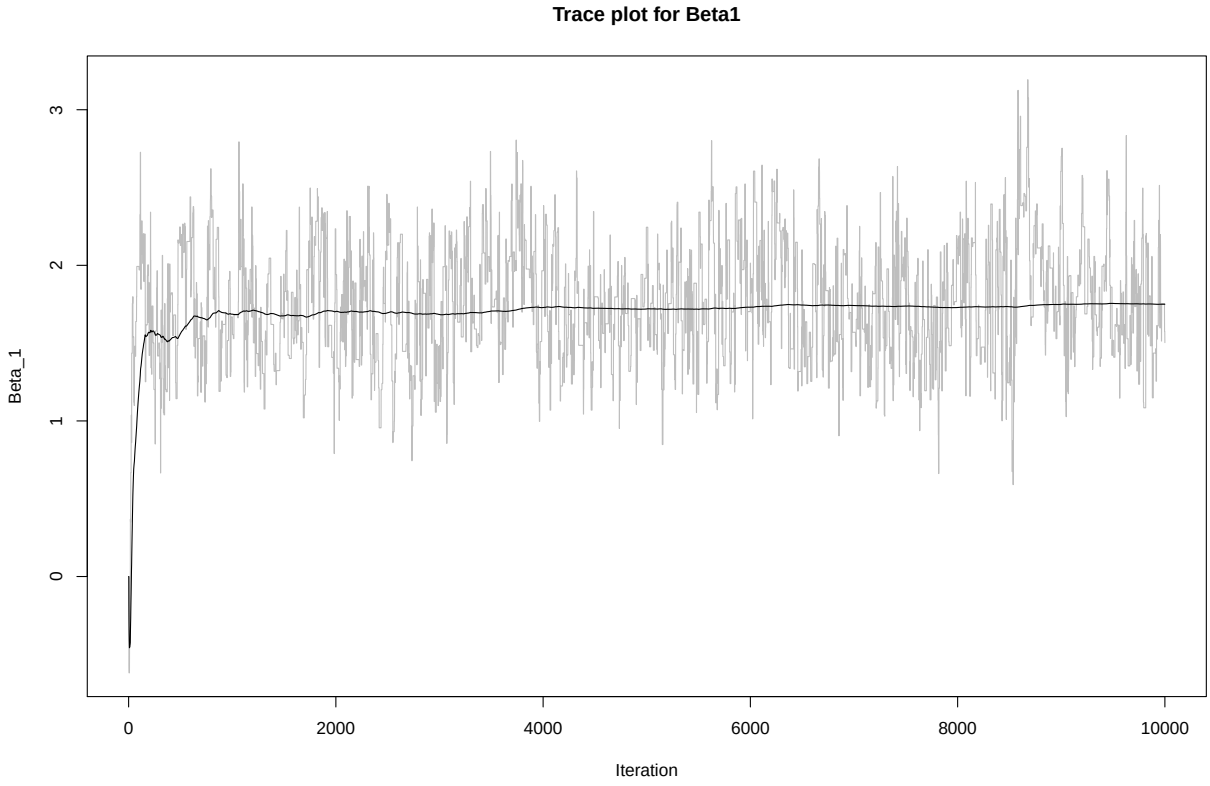
```
post.f <- function(beta, sigma2=10){
  l1<-(exp(X%*%beta)/(1+exp(X%*%beta)))^Y
  l2<-(1-exp(X%*%beta)/(1+exp(X%*%beta)))^(1-Y)
  l3<-prod(l1*l2)

  prior <- prod(exp(-(beta^2)/(2*sigma2))/sqrt(sigma2))
  return(l3*prior)
}

metropolis <- function(N=10000){
  x <- matrix(0, nrow=N, ncol=4)

  # Run the simulation
  for(i in 2:N) {
    # Generate a value from the proposal distribution
    e <- rnorm(4, sd=est.cov)

    # Generate y = x + e
    y <- x[i-1,] + e

    # Sample from the target distribution
    a.ratio <- min(log(post.f(y)) - log(post.f(x[i-1,])), 0)

    # Simulate a U(0,1) rv to decide acceptance
    u <- runif(1)

    if(log(u) <= a.ratio)
       x[i,] <- y
    else
       x[i,] <- x[i-1,]
  }

  return(x)
}
```

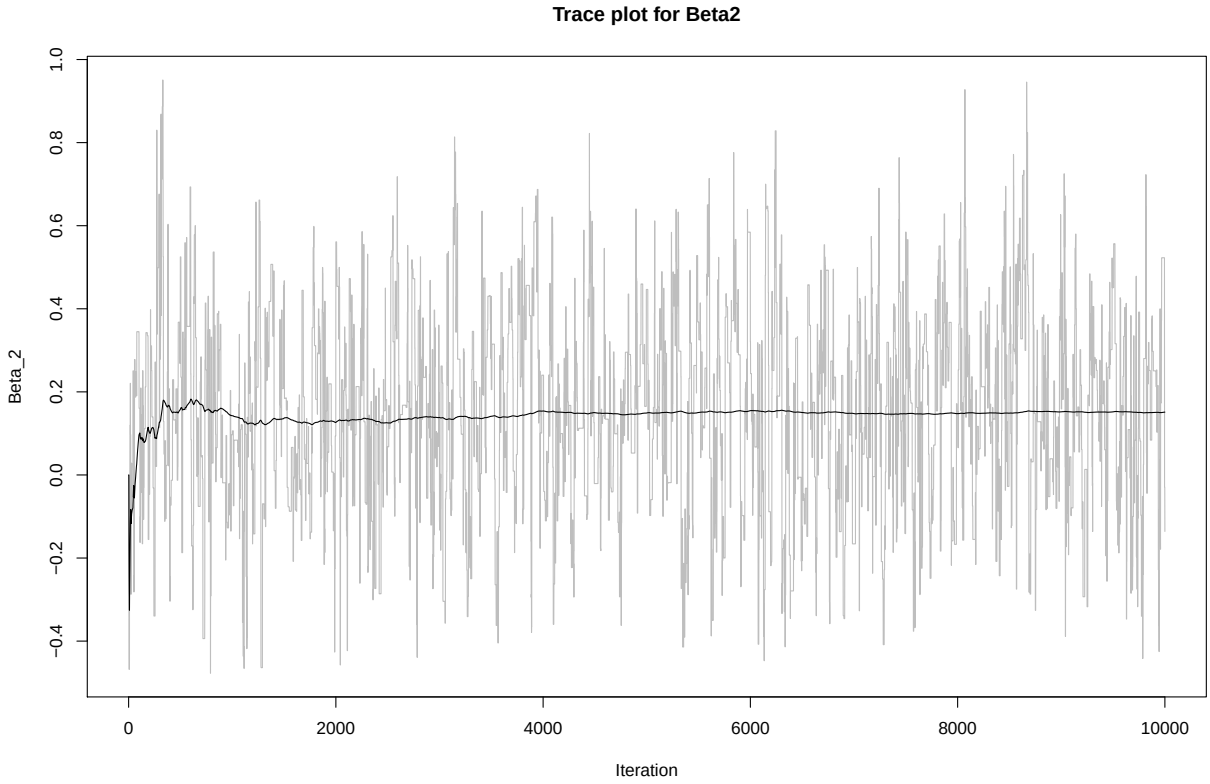
Part 1c

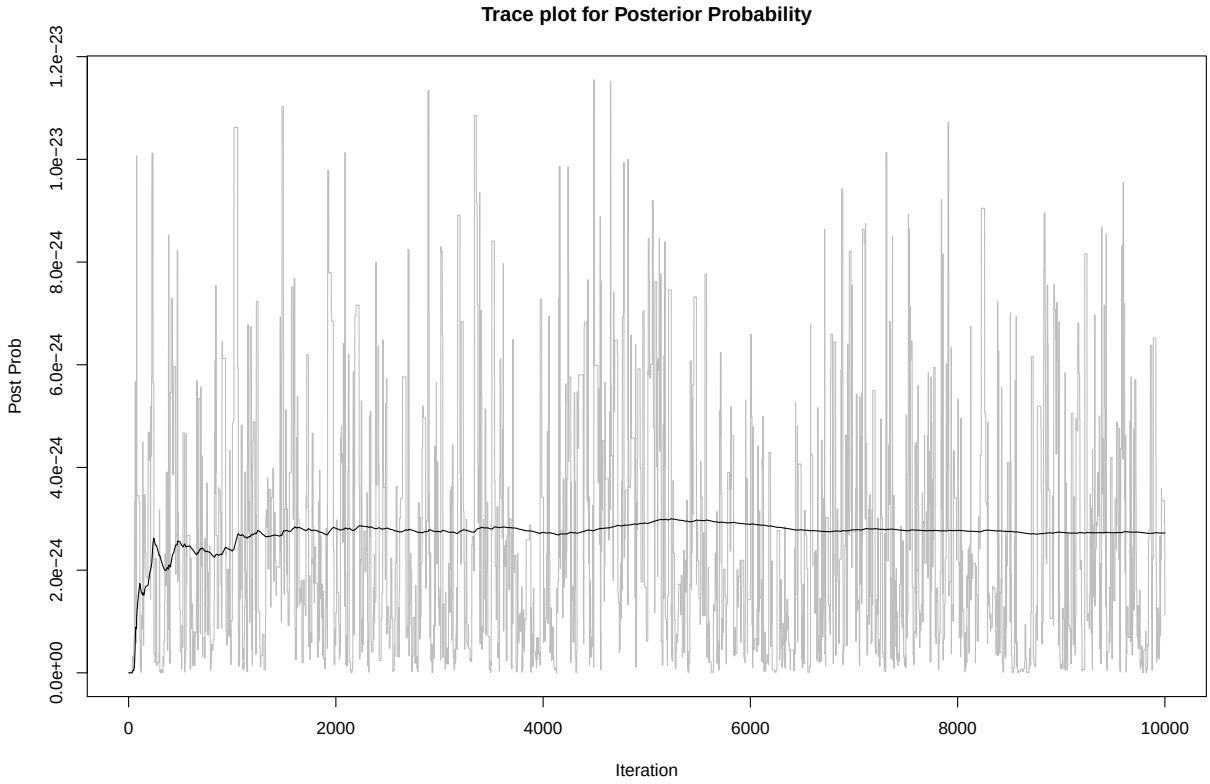
Draw the trace plot for the log-posterior probability, and draw trace plots for β_1 and β_2, the coefficients of the first two features. What would be an appropriate burn-in sample size? Estimate the effective sample sizes based on the three statistics that you have drawn trace plots for.




Histogram for Beta2
Frequency
0
500
1000
1500
−0.5
0.0
0.5
1.0
Beta2


Trace plot for Beta1
Beta_1
0
1
2
3
0
2000
4000
6000
8000
10000
Iteration





For beta_1, looking at the trace plot it seems to settle after around 800 iterations. For beta_2, it seems to

settle after around 800 iterations as well. Lastly, the posterior probability takes somewhat longer than the betas to settle at around 2200 iterations. The mixing is not that great when using a multivariate normal distribution as the proposal distribution, so choosing another proposal might lead to better mixing.

```
# Effective sample size
10000/(1+2*sum(acf(results[,1], plot = FALSE, lag.max=75)[[1]]))
```

```
[1] 276.2615
```

```
10000/(1+2*sum(acf(results[,2], plot = FALSE, lag.max=30)[[1]]))
```

```
[1] 434.4052
```

```
10000/(1+2*sum(acf(post.results, plot = FALSE, lag.max=35)[[1]]))
```

```
[1] 397.8539
```

Using the ESS formula $\frac{n}{1+2\sum_{k=1}^{\infty} ACF(k)}$ and stopping k as ACF(k) < 0.05, we get the above ESS for beta1, beta2, and the log posterior probability above.

Part 1d

Think of a way to further improve the sampling efficiency.

One potential way would be to parallelize the operation. There are some proposed methods out there that utilize existing MCMC methods to generalize and parallelize MH which would make sampling much more efficient. Another proposed method better utilizes the information generated in each iteration by using multiple evaluations of the posterior density, or a "multiple-try" algorithm. However, although the number of steps may decrease from this method, there is more computational time spent within each step. Another way to improve sampling efficiency would be to run a GLM on the data to get some initial "guesses" of beta, rather than starting at 0. This could potentially lower the number of iterations for the coefficients to settle since they start closer to the ideal value.

Question 2

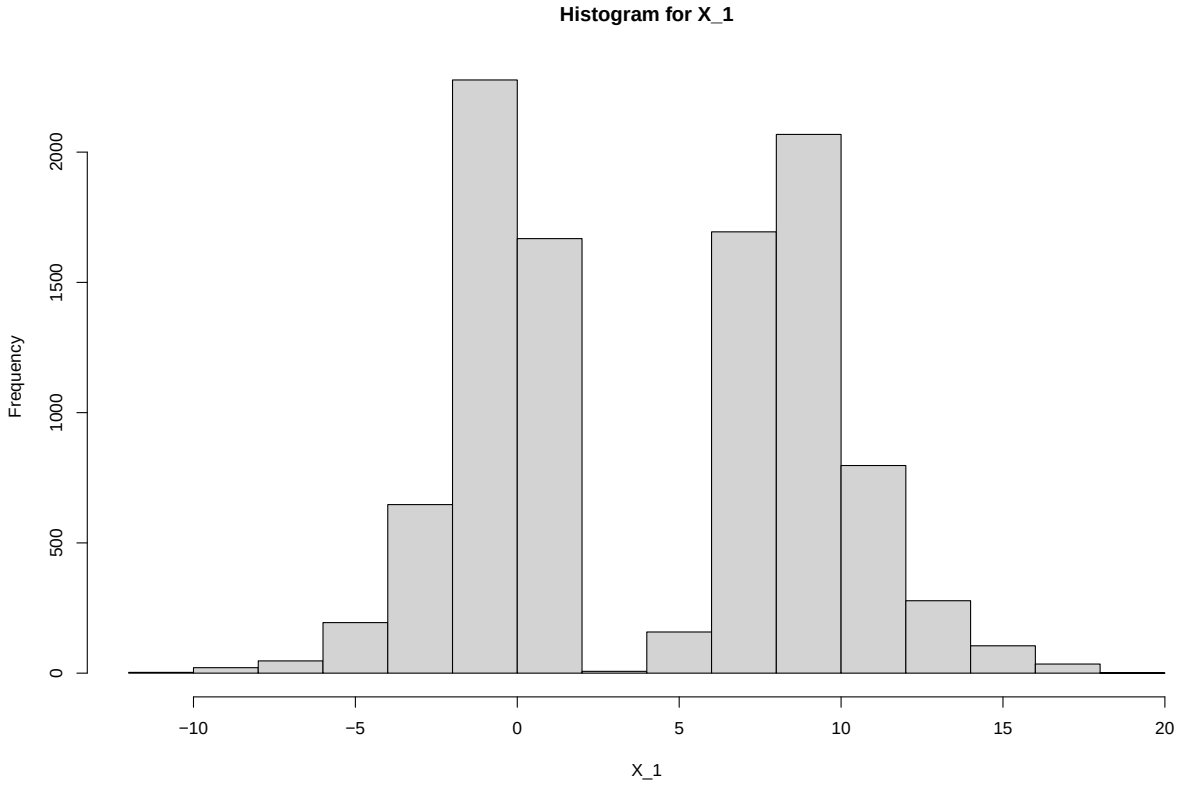
Consider a 3-dimensional bimodal distribution.

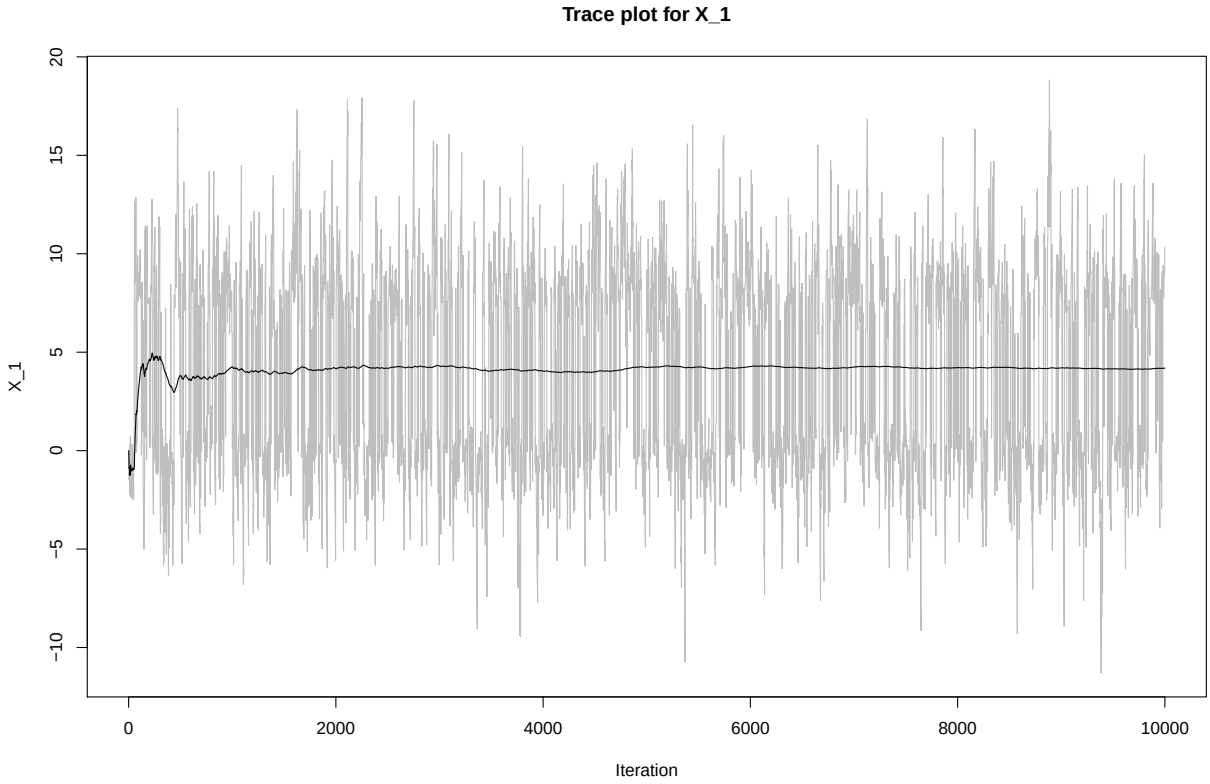
Part 2a+2b

Design Metropolis algorithm to sample from it. I will be using the multiple try metropolis algorithm with a random grid between 0 and 8.

```
mu1 <- matrix(c(0,0,0))
mu2 <- matrix(c(8,8,8))
sigma1 <- diag(3)
sigma2 <- diag(0.2, 3) + 0.8

target.2a <- function(x){
  exp(-0.5*t(x-mu1)%*%solve(sigma1)%*%(x-mu1))+
    exp(-0.5*t(x-mu2)%*%solve(sigma2)%*%(x-mu2))
}

MTM <- function(N=10000, m=8){
  x <- matrix(0, nrow=N, ncol=3)
  rg <- matrix(rep(1:m,each=3), ncol=m)

  # Run the simulation
  for(i in 2:N) {
    # Generate y = x + e
    y <- x[i-1,] + rnorm(1)*rg

    # Calculate weights
    weights <- apply(y,2,target.2a)
    weights <- weights/sum(weights)

    # Select y using these weights
    j <- sample(1:m, 1, prob=weights)
    y_j <- y[,j]

    # Draw x_star
    x_star <- y_j + rnorm(1)*rg
    x_star <- cbind(x_star, x[i-1,])

    # Sample from the target distribution
    a.ratio <- min(log(sum(weights)) - log(sum(apply(x_star,2,target.2a))), 0)

    # Simulate a U(0,1) rv to decide acceptance
    u <- runif(1)

    if(log(u) <= a.ratio)
       x[i,] <- y_j
    else
       x[i,] <- x[i-1,]
  }
```

```
  return(x)
}
```


Histogram for X_1
Frequency
0
500
1000
1500
2000
−10
−5
0
5
10
15
20
X_1

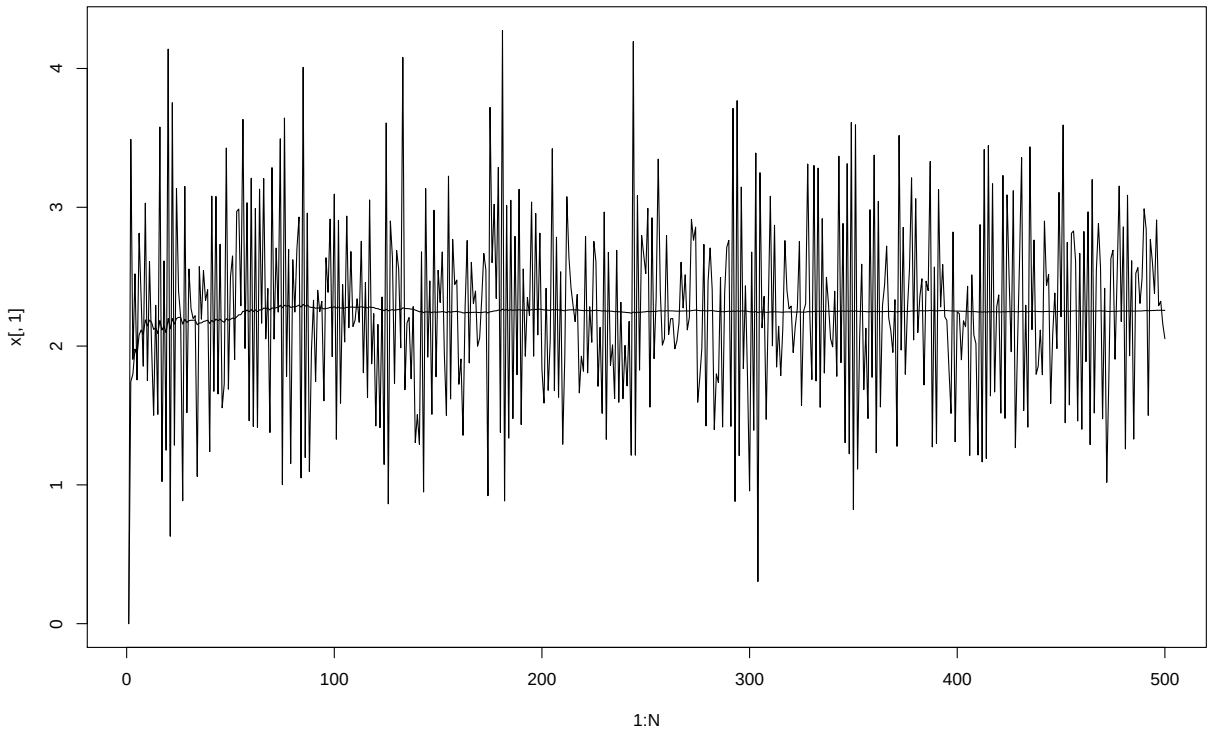

Trace plot for X_1
X_1
Iteration
20
15
10
5
0
−5
−10
0
2000
4000
6000
8000
10000

Part 2c

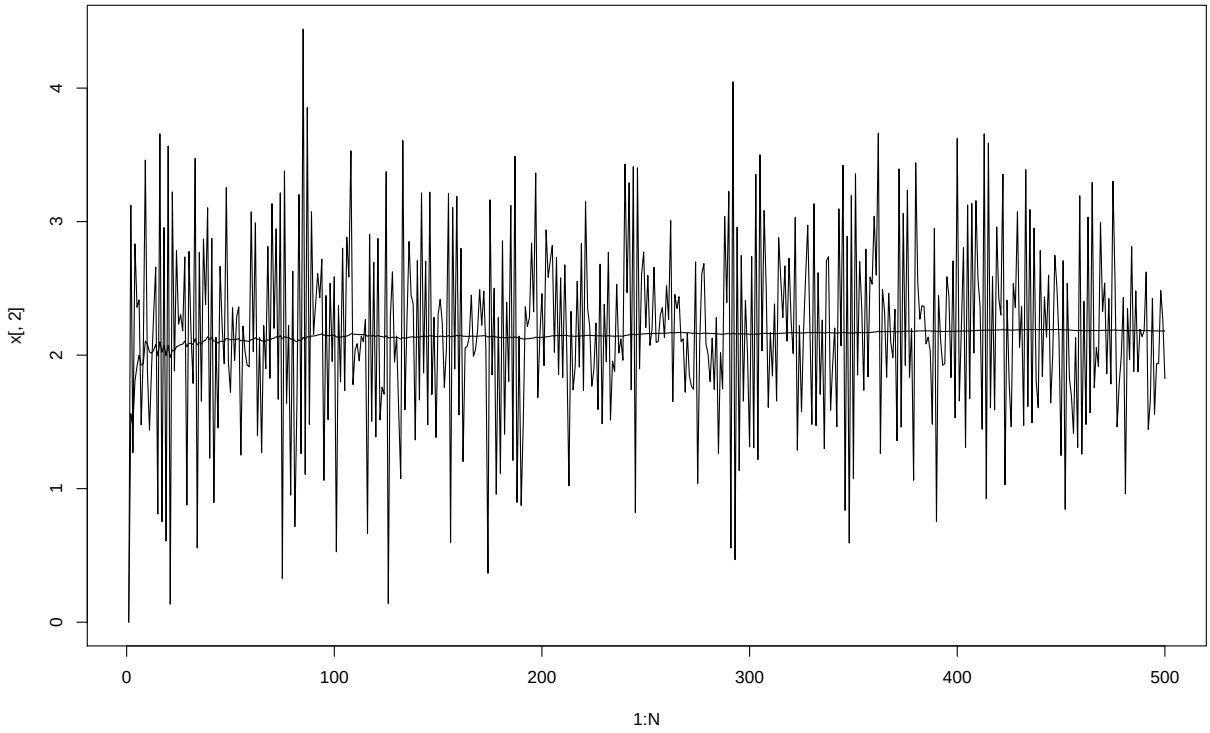
Try out the HMC sampling for this example.

```
# HMC code
log_target = function(x){
    -0.5*t(x-mu1)%*%solve(sigma1)%*%(x-mu1) + -0.5*t(x-mu2)%*%solve(sigma2)%*%(x-mu2)
}

log_grad = function(x){
    -0.5*t(x-mu1)%*%(solve(sigma1) + t(solve(sigma1))) - 0.5*t(x-mu2)%*%(solve(sigma2) + t(solve(sigma2
}

# run leapfrog integrator from xp0=(x0, p0) (with k components) for L steps of size dt
leapfrog = function(xp0, log_grad, t, dt, k) {
    L = floor(t / dt)
    x = 1:k
    p = k + 1:k

    xp = matrix(nrow=L+1, ncol=2*k)
    xp[1,] = xp0

    xp[1,p] = xp[1,p] + 0.5*dt * log_grad(xp[1,x])
    for (i in 2:L) {
        xp[i,x] = xp[i-1,x] + dt * xp[i-1,p]
        xp[i,p] = xp[i-1,p] + dt * log_grad(xp[i,x])
    }
    xp[L+1,x] = xp[L,x] + dt * xp[L,p]
    xp[L+1,p] = xp[L,p] + 0.5*dt * log_grad(xp[L+1,x])

    xp
}

# Run HMC on log_target (with gradient log_grad) for N samples,
# starting from x0 with leapfrog params t and dt
hmc = function(N, log_target, log_grad, x0, t=8, dt=0.001) {
    k = length(x0)
    x = matrix(nrow=N, ncol=k)
    x[1,] = x0

    for (i in 2:N) {
        p0 = rnorm(k)
        xp0 = c(x[i-1,], p0)
        xp = leapfrog(xp0, log_grad, t, dt, k)
        # extract proposal and momentum flip
        proposal = tail(xp, 1)
        proposal[k+1:k] = -proposal[k+1:k]

        hamil_curr = sum(p0^2)/2 - log_target(x[i-1,])
        hamil_proposal = sum(proposal[k+1:k]^2)/2 - log_target(proposal[1:k])
        alpha = exp(hamil_curr - hamil_proposal)
        if (runif(1) <= alpha)
            x[i,] = proposal[1:k]
        else
```

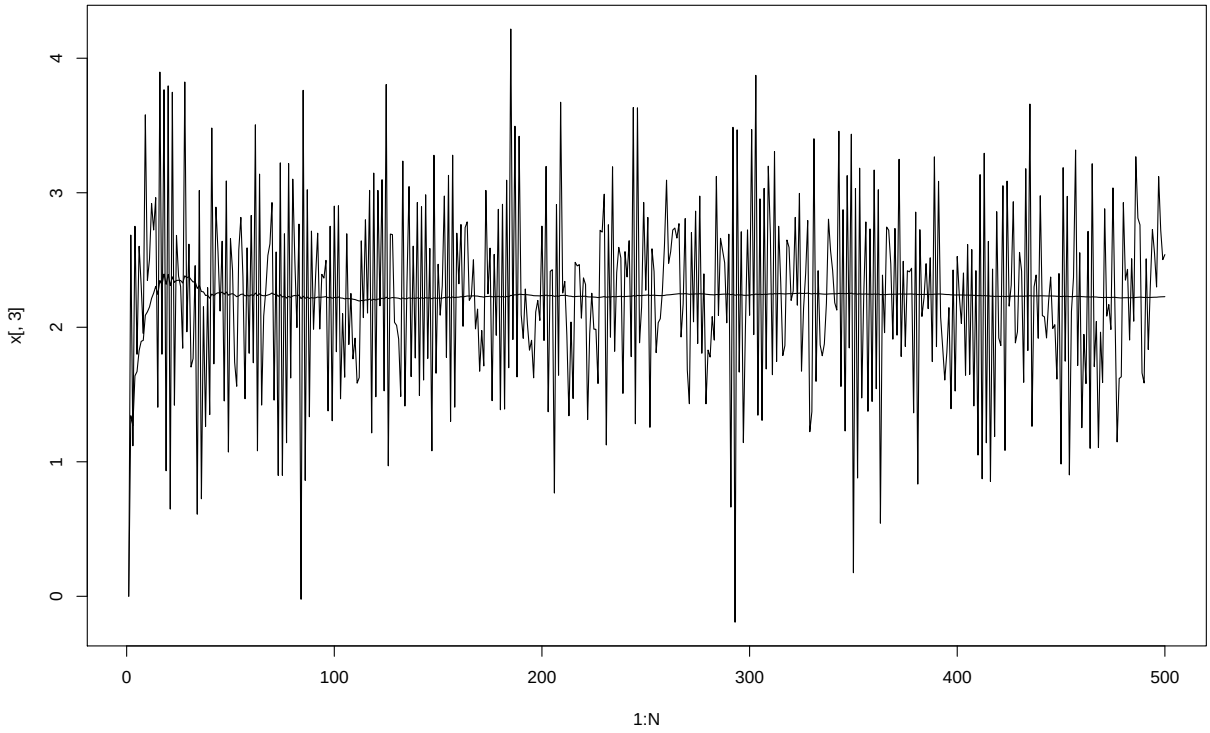
```
            x[i,] = x[i-1,]
    }

    x
}

N = 500
x = hmc(N, log_target, log_grad, c(0, 0, 0), t=2, dt=0.01)
plot(1:N, x[,1], type='l')
lines(1:500, cumsum(x[,1]) / (1:500))
```



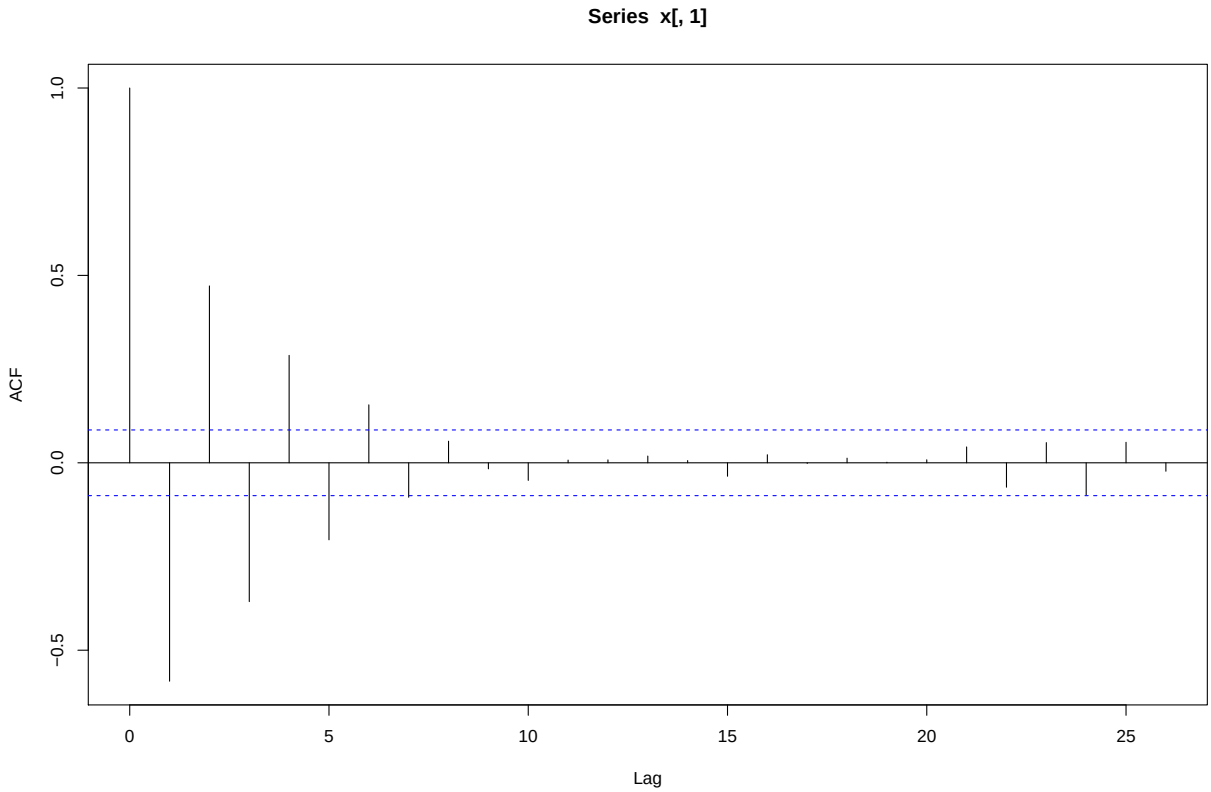

```
plot(1:N, x[,2], type='l')
lines(1:500, cumsum(x[,2]) / (1:500))
```



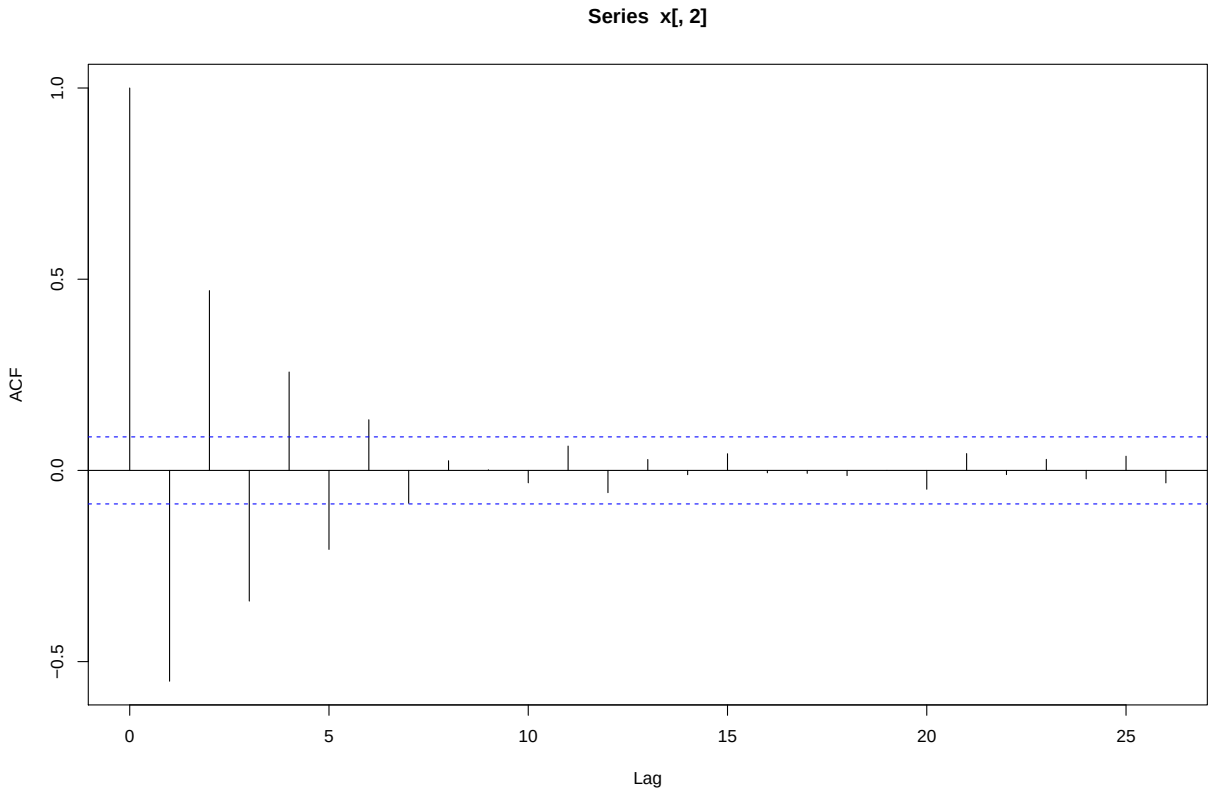

```
plot(1:N, x[,3], type='l')
lines(1:500, cumsum(x[,3]) / (1:500))
```



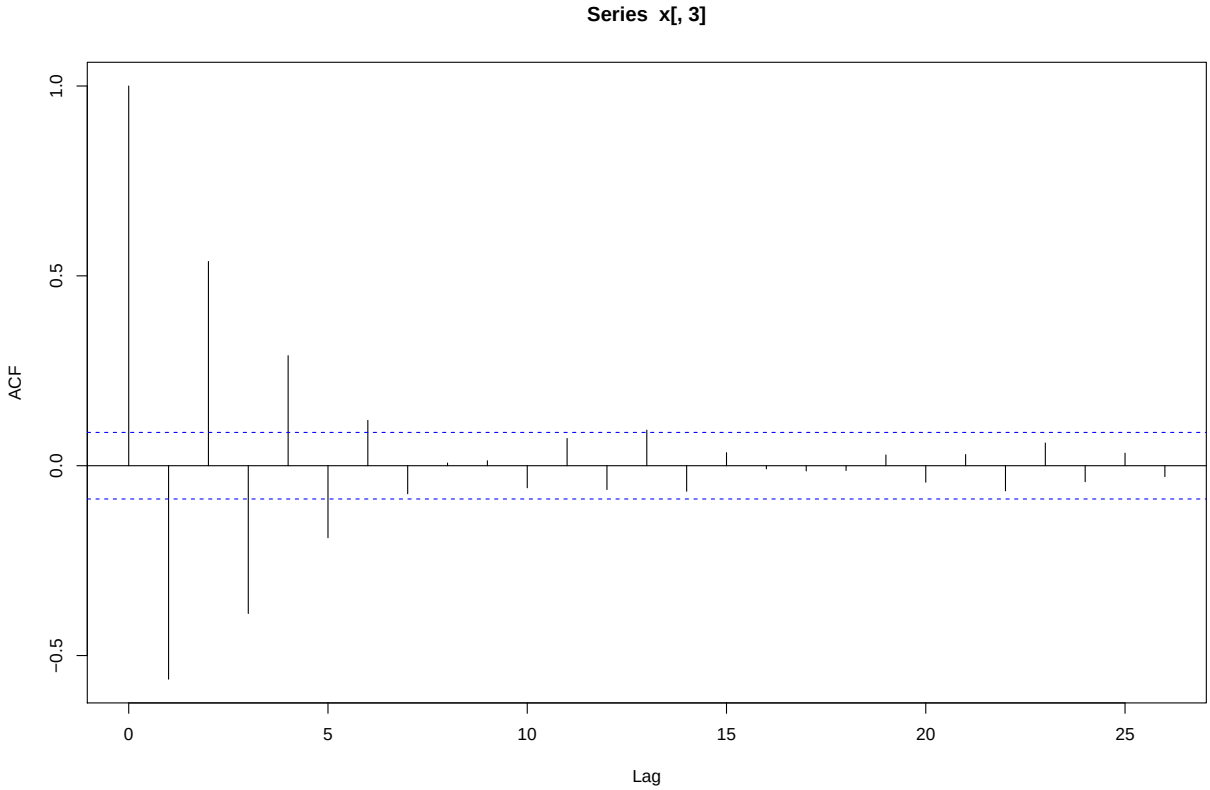

```
acf(x[,1])
```




```
acf(x[,2])
```




```
acf(x[,3])
```


Series x[, 3]
ACF
1.0
0.5
0.0
−0.5
0
5
10
15
20
25
Lag

Problem 3

```
# Simulate probit regression model
set.seed(128)
N=1000
X <- matrix(c(rep(1,N),rnorm(N)),N,2)
beta <- matrix(c(0,3))
Z <- X%*%beta + rnorm(N)
Y <- matrix(ifelse(Z > 0, 1, 0))
data3 <- cbind(Y, X)
beta
```

```
     [,1]
[1,]    0
[2,]    3
```

Part 3a

Prove the two are equivalent.

$$\begin{aligned} P(Y_i = 1|X_i) &= P(X^T\beta + \epsilon > 0) \\ &= P(e > -X^T\beta) \\ &= P(e < X^T\beta) \\ &= \Phi(X^T\beta) \end{aligned}$$

where step 3 is from the symmetry of the normal distribution.

Part 3b

Write an EM algorithm to estimate β.

Step 1: Select a starting value $\beta^{(0)}$

Step 2: E Step: Compute $Z^{(t+1)}$ by using the following conditional probabilities:

$$Z_i^{(t+1)} = \begin{cases} x_i^T \beta^{(t)} - \frac{\phi(x_i^T \beta^{(t)})}{\Phi(-x_i^T \beta^{(t)})} & \text{if } y_i = 0 \\ x_i^T \beta^{(t)} + \frac{\phi(x_i^T \beta^{(t)})}{1-\Phi(-x_i^T \beta^{(t)})} & \text{if } y_i = 1 \end{cases}$$

Step 3: M Step: Compute $\beta^{(t+1)}$ using the least squares MLE: $(X^T X)^{-1} X^T Z^{(t+1)}$.

Step 4: Repeat until convergence.

```
initial_beta <- c(1,1)

probit_y1 <- function(betak, xi, sig=1){
  xi%*%betak + dnorm(-xi%*%betak,sd=sig)/(1-pnorm(-xi%*%betak,sd=sig))
}

probit_y0 <- function(betak, xi, sig=1){
  xi%*%betak - dnorm(-xi%*%betak,sd=sig)/(pnorm(-xi%*%betak,sd=sig))
}

runEM <- function(initial_beta, max.iter=1000, data=data3){
  beta_k <- initial_beta

  for (i in 1:max.iter){
    # E step
    Z <- matrix(apply(data, 1, function(inp){
      ifelse(inp[1]==1, probit_y1(beta_k, inp[2:3]), probit_y0(beta_k, inp[2:3]))
    }))

    # M step
    beta_k <- solve(t(X)%*%X)%*%t(X)%*%Z
  }
  return(beta_k)
}

runEM(initial_beta)
```

```
           [,1]
[1,] 0.02657059
[2,] 2.98907924
```

Part 3c

The algorithm is as follows:

Step 1: Select a starting value $\beta^{(0)}$

Step 2: E Step: Compute $Z^{(t+1)}$ by using the following conditional probabilities:

$$Z_i^{(t+1)} = \begin{cases} x_i^T\beta^{(t)} - \frac{\phi(x_i^T\beta^{(t)})}{\Phi(-x_i^T\beta^{(t)})} & \text{if } y_i = 0 \\ x_i^T\beta^{(t)} + \frac{\phi(x_i^T\beta^{(t)})}{1-\Phi(-x_i^T\beta^{(t)})} & \text{if } y_i = 1 \end{cases}$$

Step 3: Calculate $E(Z_i^2|Y,\beta^{(t)}) = (Z_i^{(t+1)})^2 + 1 - X\beta^{(t)}(Z_i^{(t+1)} - X\beta^{(t)})$.

Step 4: Calculate the cross product matrix of the data

Step 5: Sweep[1:2]C, where C is the cross product matrix of the data.

Step 6: The last column of sweep gives $\beta^{(t+1)}$ as well as the SSE. Taking the SSE and dividing it by the df and taking the square root gives σ.

Step 7: Repeat until convergence.

```
# Simulate probit regression model with sigma
set.seed(128)
N=1000
X <- matrix(c(rep(1,N),rnorm(N)),N,2)
sigma <- 3
beta <- matrix(c(0,3))
theta <- beta*sigma
Z <- X%*%theta + rnorm(N, sd=sigma)
Y <- matrix(ifelse(Z > 0, 1, 0))
data3 <- cbind(Y, X)

runEM2 <- function(initial_beta, max.iter=500, data=data3){
  beta_k <- matrix(initial_beta)
  sigma_k <- 1
  df=N-2

  for (i in 1:max.iter){
    # E step
    Z <- matrix(apply(data, 1, function(inp){
      ifelse(inp[1]==1, probit_y1(beta_k, inp[2:3],sigma_k), probit_y0(beta_k, inp[2:3],sigma_k))
    }))

    # Compute conditional Z^2 expectation
    temp <- cbind(X, Z)
    Z2 <- matrix(apply(temp,1,function(inp){
      inp[3]^2 + 1 - inp[1:2]%*%beta_k%*%(inp[3]-inp[1:2]%*%beta_k)
    }))

    # M step
    cp <- crossprod(temp,temp)
    cp[3,3] <- sum(Z2)

    beta_k[1] <- SWP(cp,1:2)[1,3]
```

```
    beta_k[2] <- SWP(cp,1:2)[2,3]
    sigma_k <- sqrt(SWP(cp,1:2)[3,3]/df)
  }
  return(list(beta=beta_k,sigma=sigma_k))
}
test<-runEM2(initial_beta)
test$beta/test$sigma
```

```
           [,1]
[1,] 0.02657034
[2,] 2.98904271
```