STATS 451 Homework 6

Yuzhou Peng

2024-03-27

```
library(ggplot2)
library(gridExtra)
library(tidyr)
library(mcmc)
library(coda)
```

Problem 1

$$y_i|\theta_i \sim Bin(n_i, \theta_i)$$
$$logit(\theta_i) = log(\frac{\theta_i}{1-\theta_i}) = \alpha + \beta x_i$$
$$\text{The likelihood is given by } \mathbb{P}(y_i|\theta_i) \propto \prod_{i=1}^{k} \theta_i^{y_i}(1-\theta_i)^{n_i-y_i}$$
$$\text{or equivalently, } \mathbb{P}(y_i|\alpha,\beta) \propto \prod_{i=1}^{k} [\frac{e^{\alpha+\beta x_i}}{e^{\alpha+\beta x_i}+1}]^{y_i}[\frac{1}{e^{\alpha+\beta x_i}+1}]^{n_i-y_i}$$
$$\text{The prior is non-informative } \mathbb{P}(\alpha,\beta) \propto 1$$
$$\text{The posterior is given by } \mathbb{P}(\alpha,\beta|y_i) \propto \prod_{i=1}^{k} \mathbb{P}(y_i|\alpha,\beta) \propto [\frac{e^{\alpha+\beta x_i}}{e^{\alpha+\beta x_i}+1}]^{y_i}[\frac{1}{e^{\alpha+\beta x_i}+1}]^{n_i-y_i}$$

Problem 2

```r
# The example data
df1 <- data.frame(
  x = c(-0.86, -0.30, -0.05, 0.73),
  n = c(5, 5, 5, 5),
  y = c(0, 1, 3, 5)
)



log_prior <- function(param){
  return(0)
}

log_likelihood <- function(param, data){
  # param is the vector containing alpha and beta
  # data is the dataframe containing ni, xi and yi

  y <- data$y
  x <- data$x
  n <- data$n
  alpha <- param[1]
  beta <- param[2]

  p_data_given_param <- 0

  for (i in 1:length(n)) {
    p <- y[i] * (alpha + beta * x[i] - log(1 + exp(alpha + beta * x[i]))) + (n[i] - y[i]) * (1
- log(1 + exp(alpha + beta * x[i])))
    p_data_given_param = p_data_given_param + p
  }

  return(p_data_given_param)
}

rel_log_posterior <- function(param, data){
  rel_prob <- log_prior(param) + log_likelihood(param, data)
  return(rel_prob)
}


rel_log_posterior(param = c(0.8,7.7), data = df1)
```

```
## [1] 5.10399
```

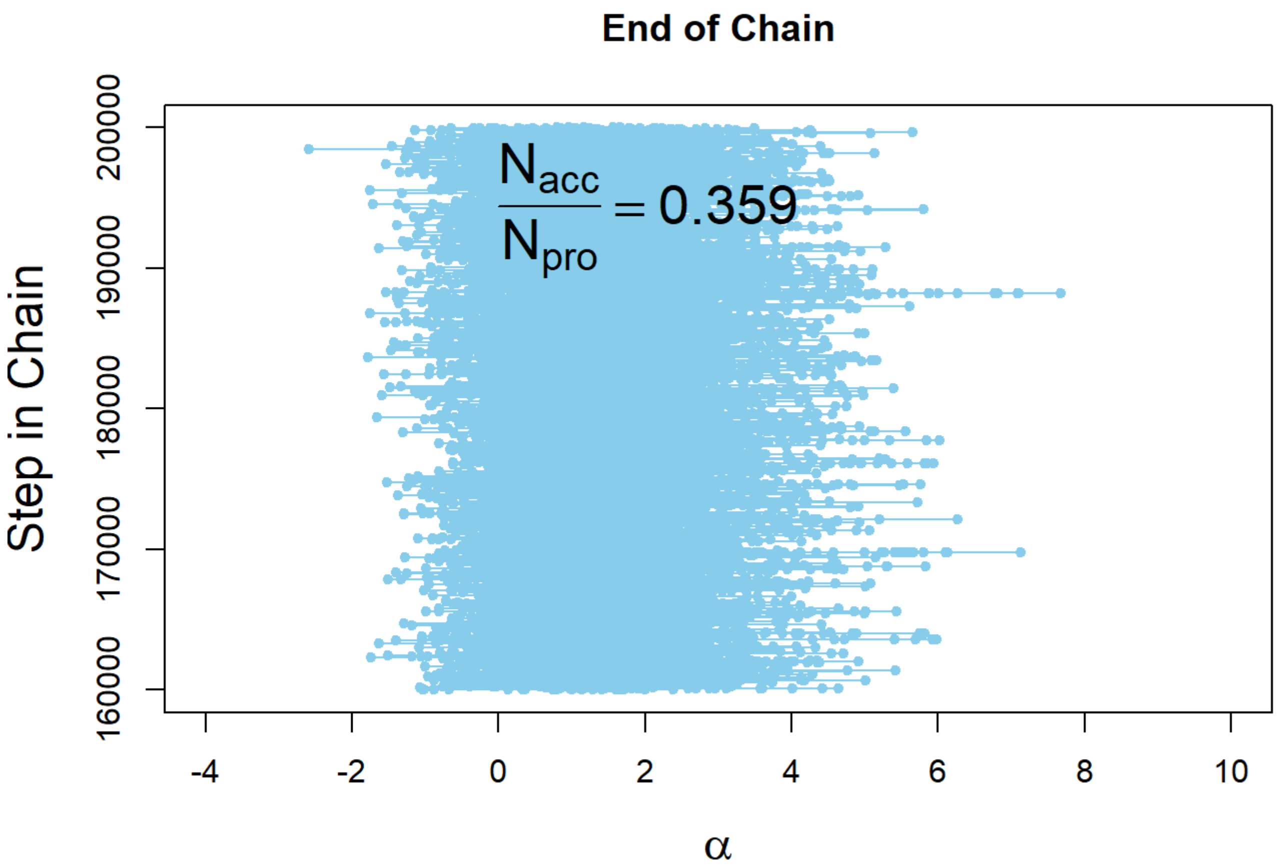
Metropolis algorithm

```
# Specify the length of the trajectory, i.e., the number of jumps to try:
trajLength = 200000 # arbitrary large number
# Initialize the vector that will store the results/samples of theta1 and theta2:
trajectory = array( 0 , c(2, trajLength))
# Specify where to start the trajectory:
trajectory[,1] = c(0.8, 7.7)
# arbitrary value # you can do it in a smarter way
# Specify the burn-in period:
burnIn = ceiling( 0.2 * trajLength )
# arbitrary number, less than trajLength
# Initialize accepted, rejected counters, just to monitor performance:
nAccepted = 0
nRejected = 0
```

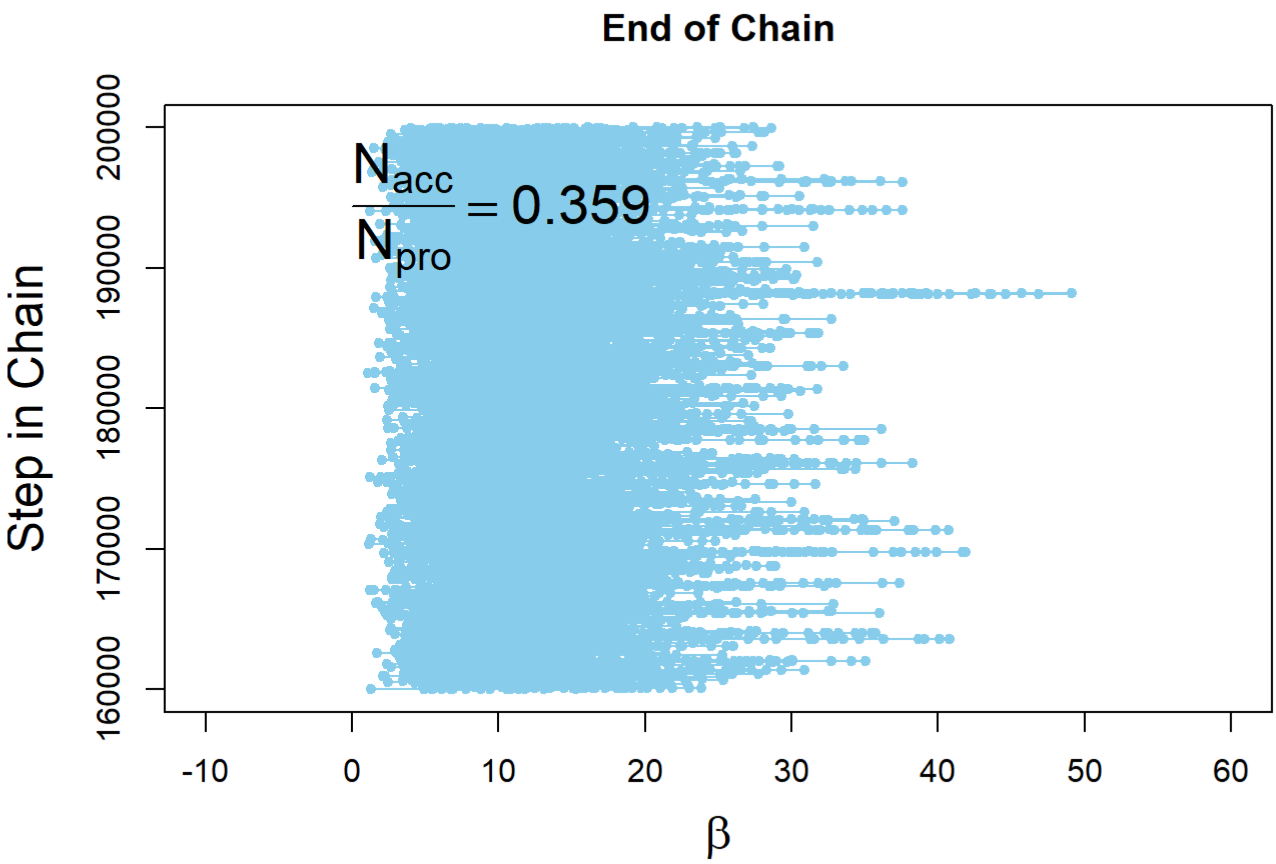
```
set.seed(451)

# change proposal SD to see how the chain runs and how the result changes
#proposalSD = rep(c(0.02,0.2,2.0)[2], 2)
proposalSD <- c(2, 4)

for ( t in 1:(trajLength-1) )  {

    currentPosition <- trajectory[, t]
    # Use the proposal distribution to generate a proposed jump.
    proposedJump <- rnorm( 2, mean=0 , sd=proposalSD )
  proposedPosition <- currentPosition + proposedJump
  log_probAccept <- min(0, rel_log_posterior(proposedPosition, df1) - rel_log_posterior(current
Position, df1))
  probAccept <- exp(log_probAccept)
    # Generate a random uniform value from the interval [0,1] to
    # decide whether or not to accept the proposed jump.
    if ( runif(1) < probAccept )  {
        # accept the proposed jump
        trajectory[, t+1 ] <- currentPosition + proposedJump
        # increment the accepted counter, just to monitor performance
        if ( t > burnIn )  { nAccepted = nAccepted + 1  }
     } else  {
        # reject the proposed jump, stay at current position
        trajectory[ ,t+1 ] = currentPosition
        # increment the rejected counter, just to monitor performance
        if ( t > burnIn )  { nRejected = nRejected + 1  }
     }
 }




# Extract the post-burnIn portion of the trajectory.
acceptedTraj = trajectory[ , (burnIn+1) : dim(trajectory)[2] ]

# End of Metropolis algorithm.
```

Visualization

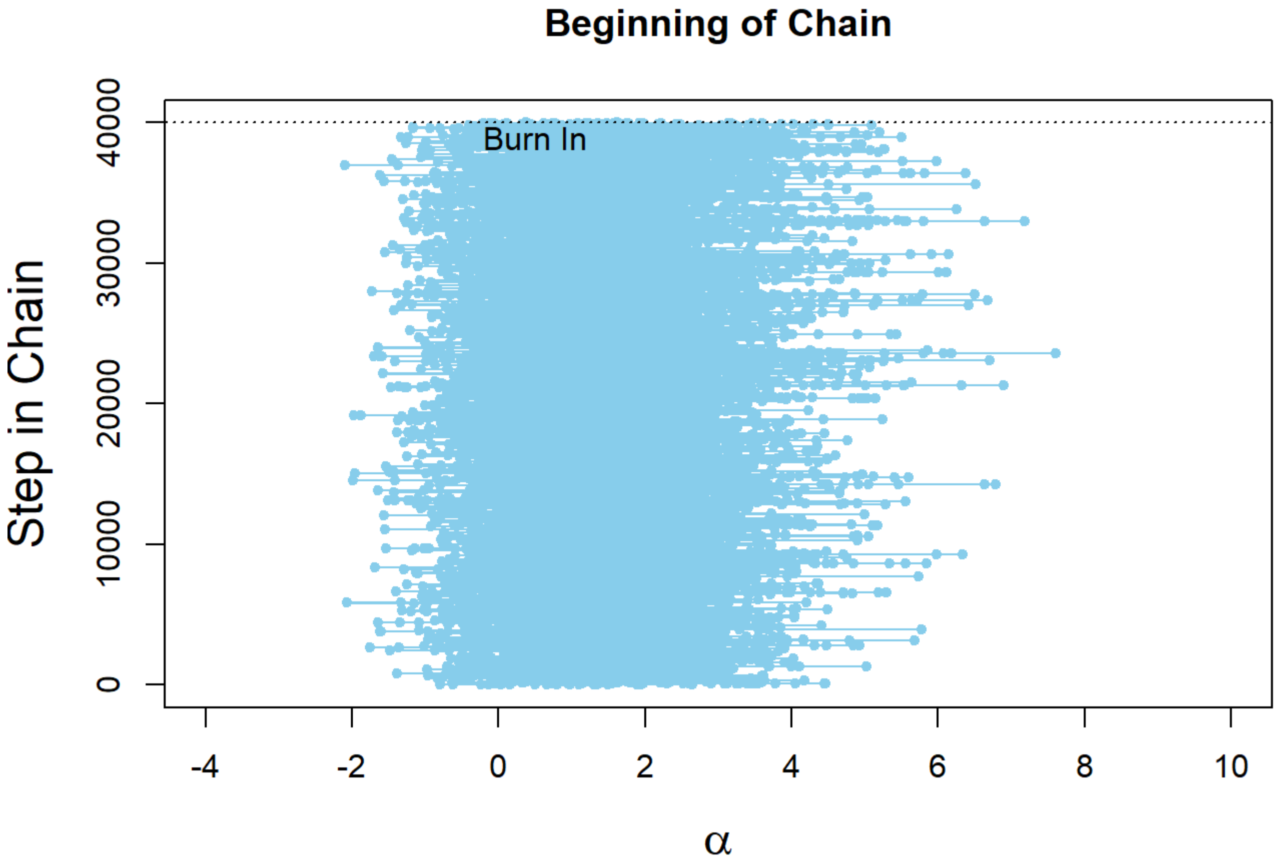
```
#------------------------------------------------------------------------
# Display the chain.

#openGraph(width=4,height=8)
#layout( matrix(1:2,nrow=2) )
#par(mar=c(3,4,2,1),mgp=c(2,0.7,0))

# Trajectory, a.k.a. trace plot, end of chain:
idxToPlot = (trajLength-burnIn):trajLength

plot( trajectory[1,idxToPlot] , idxToPlot , main="End of Chain" ,
        xlab=bquote(alpha) , xlim=c(-4,10) , ylab="Step in Chain" ,
        type="o" , pch=20 , col="skyblue" , cex.lab=1.5 )
  # Display proposal SD and acceptance ratio in the plot.
text( 0.0 , trajLength , adj=c(0.0,1.1) , cex=1.75 ,
        labels = bquote( frac(N[acc],N[pro]) ==
                           .(signif( nAccepted/(nAccepted+nRejected) , 3 ))))
```



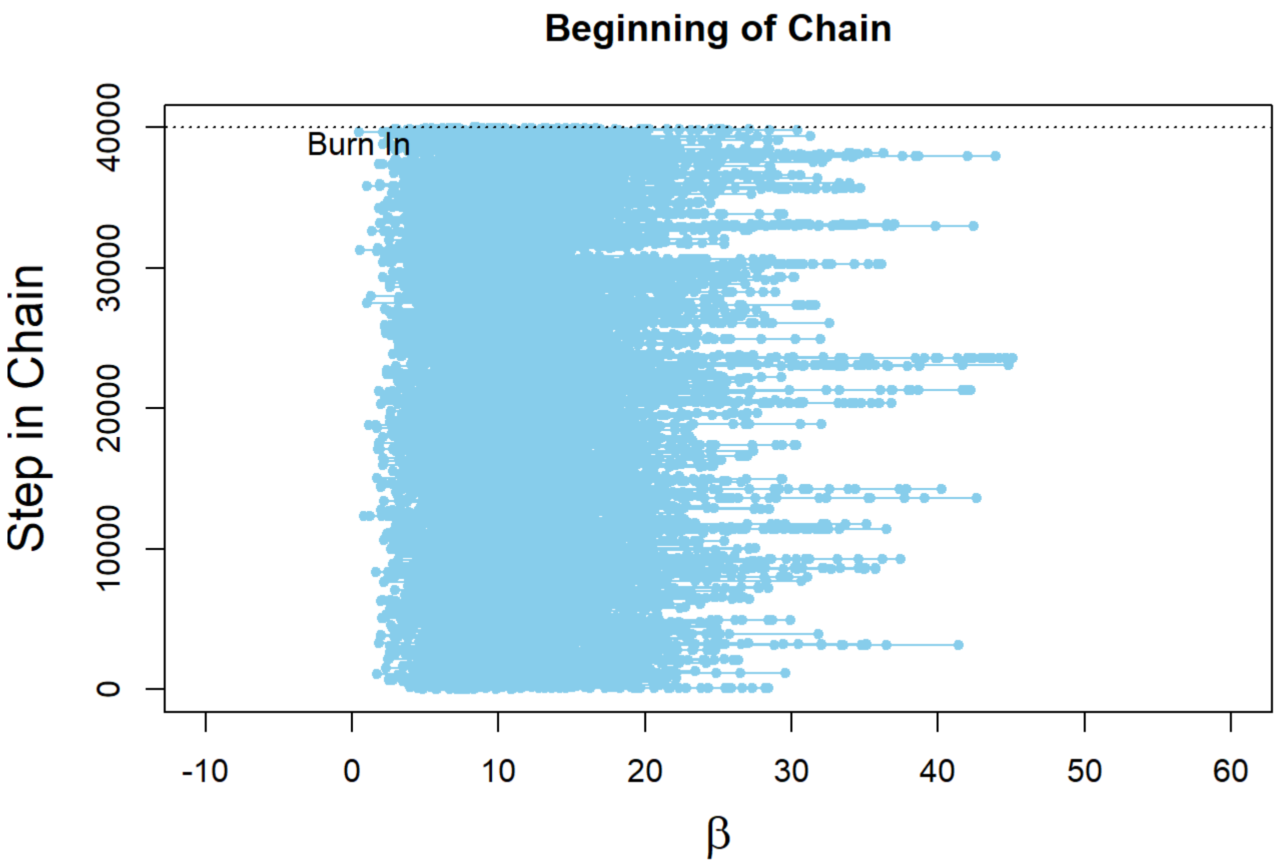

```
plot( trajectory[2,idxToPlot] , idxToPlot , main="End of Chain" ,
        xlab=bquote(beta) , xlim=c(-10,60) , ylab="Step in Chain" ,
        type="o" , pch=20 , col="skyblue" , cex.lab=1.5 )
  # Display proposal SD and acceptance ratio in the plot.
text( 0.0 , trajLength , adj=c(0.0,1.1) , cex=1.75 ,
        labels = bquote( frac(N[acc],N[pro]) ==
                           .(signif( nAccepted/(nAccepted+nRejected) , 3 ))))
```



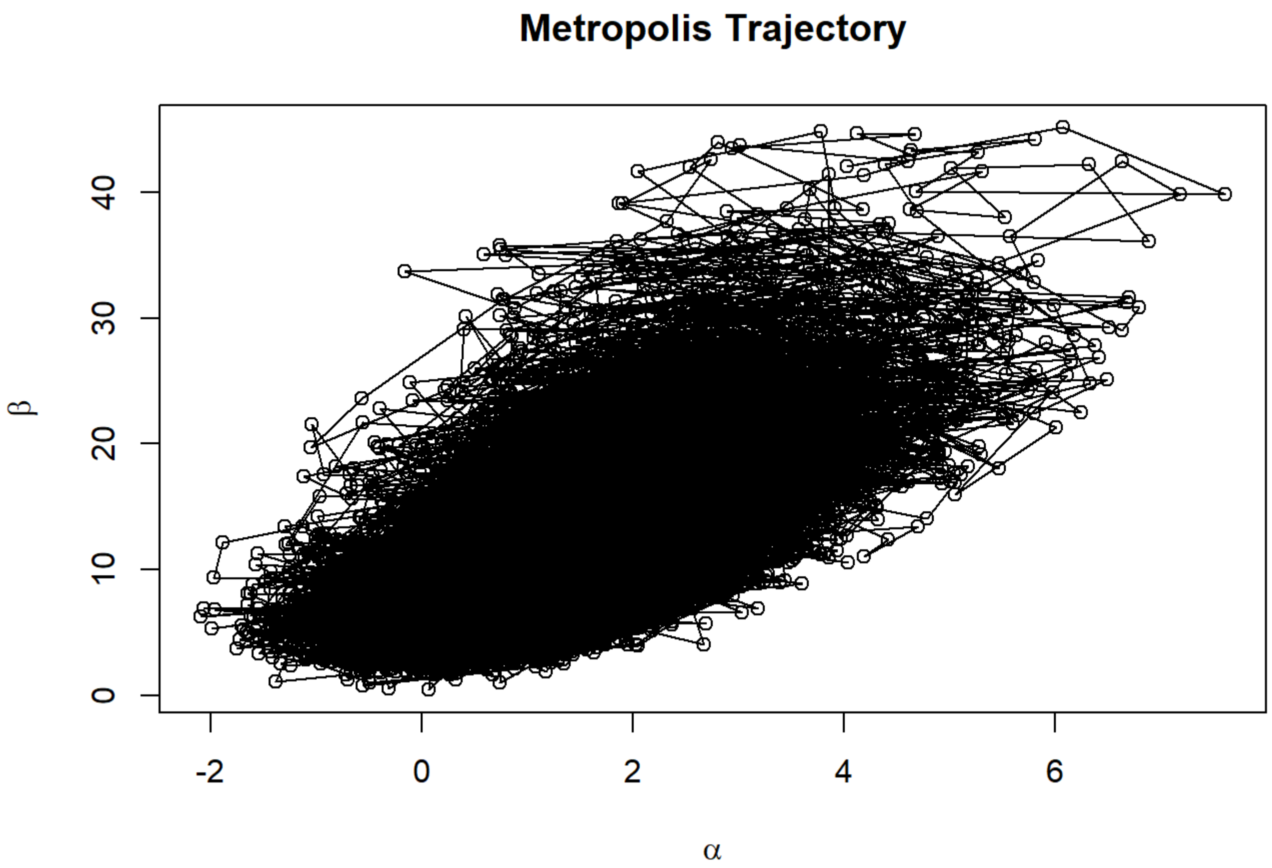

```
# Trajectory, a.k.a. trace plot, beginning of chain:
idxToPlot = 1:burnIn

plot( trajectory[1, idxToPlot] , idxToPlot , main="Beginning of Chain" ,
      xlab=bquote(alpha) , xlim=c(-4,10) , ylab="Step in Chain" ,
      type="o" , pch=20 , col="skyblue" , cex.lab=1.5 )
# Indicate burn in limit (might not be visible if not in range):
if ( burnIn > 0 )  {
  abline(h=burnIn,lty="dotted")
  text( 0.5 , burnIn+1 , "Burn In" , adj=c(0.5,1.1) )
 }
```



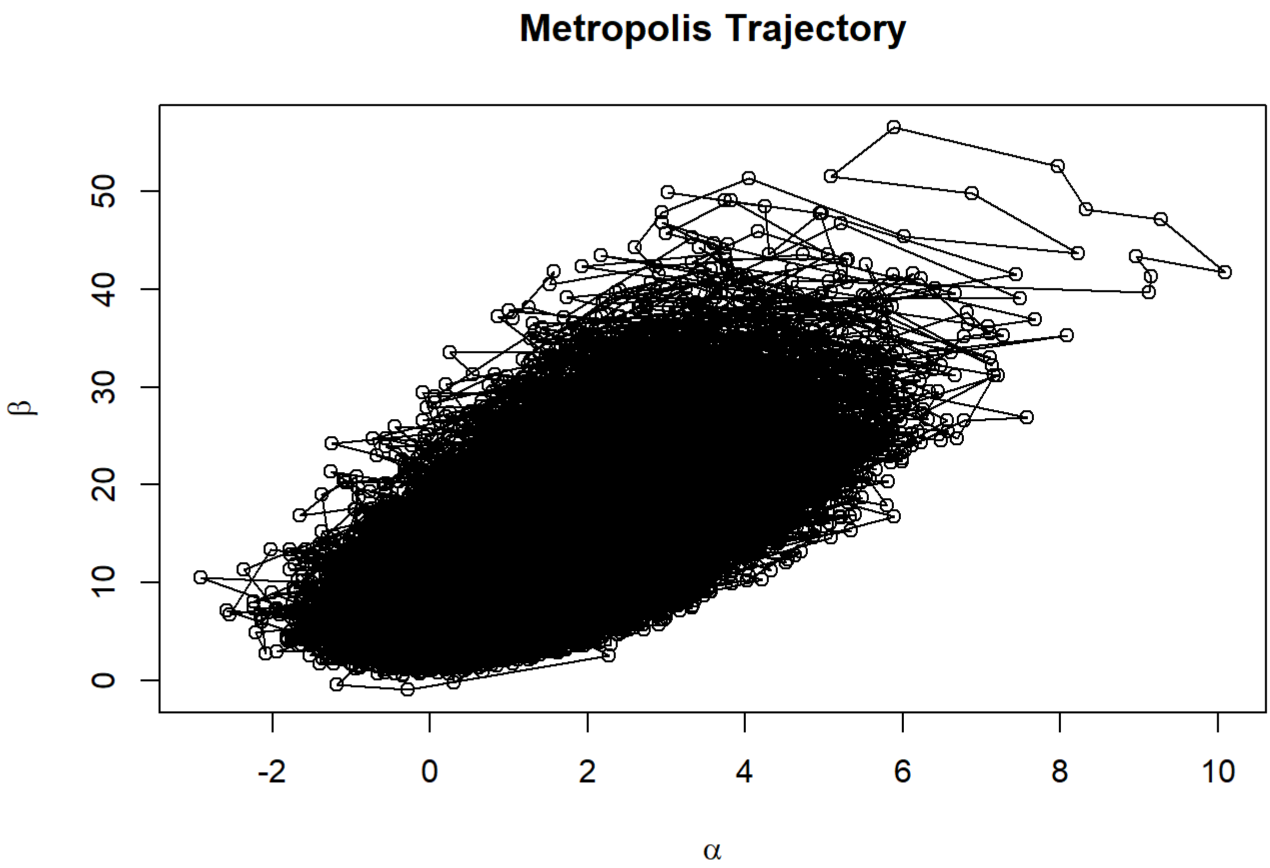

```
plot( trajectory[2, idxToPlot] , idxToPlot , main="Beginning of Chain" ,
      xlab=bquote(beta) , xlim=c(-10,60) , ylab="Step in Chain" ,
      type="o" , pch=20 , col="skyblue" , cex.lab=1.5 )
# Indicate burn in limit (might not be visible if not in range):
if ( burnIn > 0 )  {
  abline(h=burnIn,lty="dotted")
  text( 0.5 , burnIn+1 , "Burn In" , adj=c(0.5,1.1) )
 }
```




```
# the beginning of the trajectory
idxBegin <- 1:burnIn
plot(trajectory[1, idxBegin], trajectory[2, idxBegin],
     xlab = expression(alpha), ylab = expression(beta),
     main = "Metropolis Trajectory")
lines(trajectory[1, idxBegin], trajectory[2, idxBegin])
```




```
# after converging
idxToPlot <- burnIn:trajLength
idxSimple <- seq(idxToPlot[1], idxToPlot[length(idxToPlot)])
plot(trajectory[1, idxSimple], trajectory[2, idxSimple],
     xlab = expression(alpha), ylab = expression(beta),
     main = "Metropolis Trajectory")
lines(trajectory[1, idxSimple], trajectory[2, idxSimple])
```




```
mcmc_obj <- as.mcmc(t(acceptedTraj))

ess <- effectiveSize(mcmc_obj)

print(ess)
```

```
##      var1      var2
## 9507.520 4754.931
```

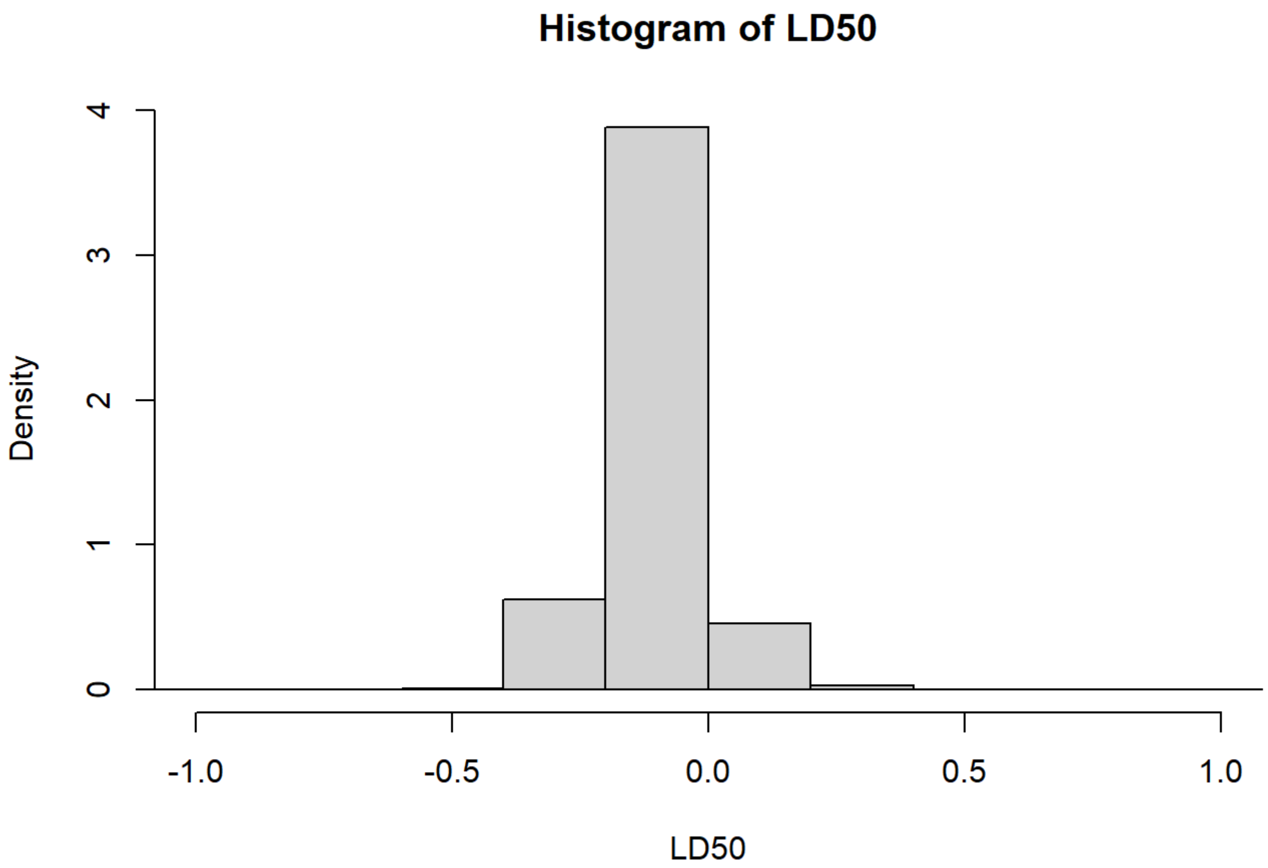
The random walk follows bivariate normal distribution with standard deviation (2,4).
The total number of iterations are 200000 and effective sample size is 9507 and 4755.

Problem 3 LD50

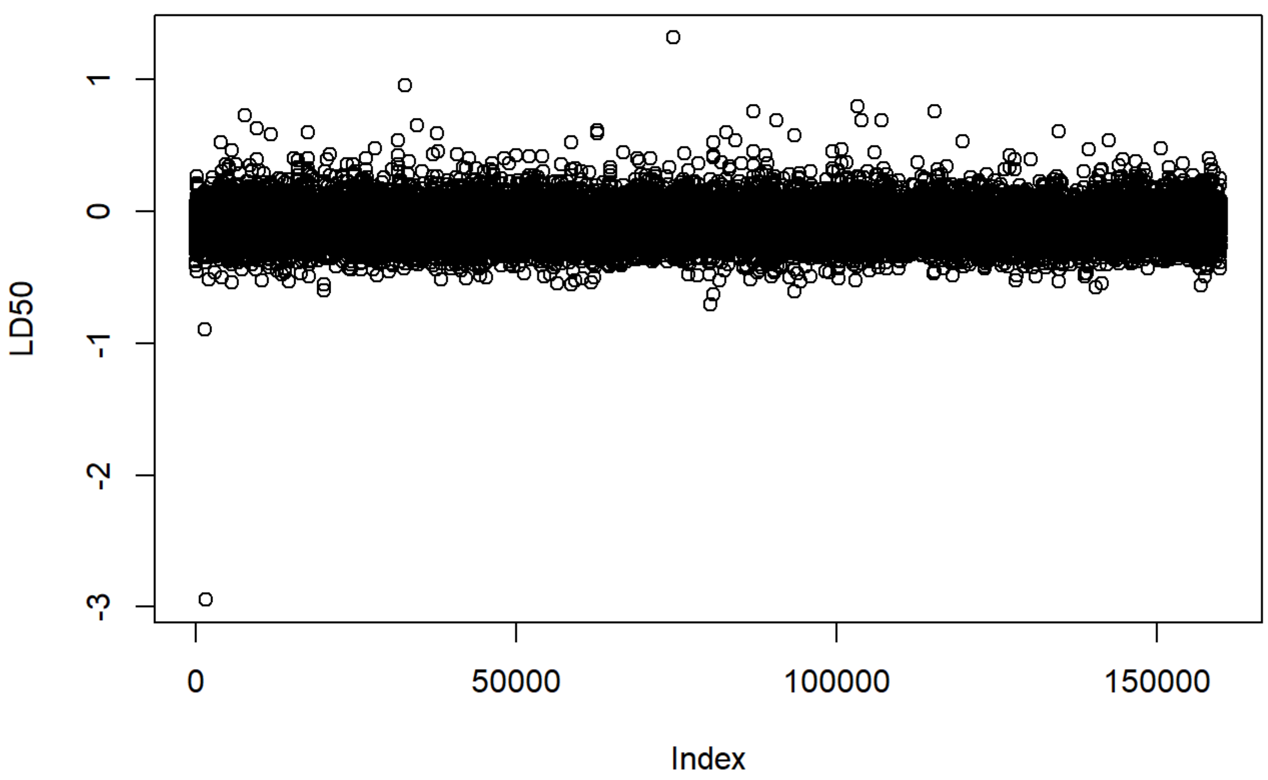
```
alpha_sample <- acceptedTraj[1,]
beta_sample <- acceptedTraj[2,]

LD50 <- -alpha_sample/beta_sample

summary(LD50)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -2.94590 -0.16281 -0.11163 -0.10776 -0.06064  1.31933
```

```
hist(LD50, freq= F, xlim = c(-1,1))
```


Histogram of LD50
Density
0
1
2
3
4
-1.0
-0.5
0.0
0.5
1.0
LD50

```
plot(LD50)
```


LD50
1
0
-1
-2
-3
0
50000
100000
150000
Index