Metropolis-Hastings Example 1: Simple Binomial logit model

Chang Tu

```
###Metropolis-Hastings Example 1: Simple Binomial logit model
library(MASS)
library(mvtnorm)  ##multivariate normal and t  distributions
library(arm)    #gives access to logit( and invlogit  functions)
```

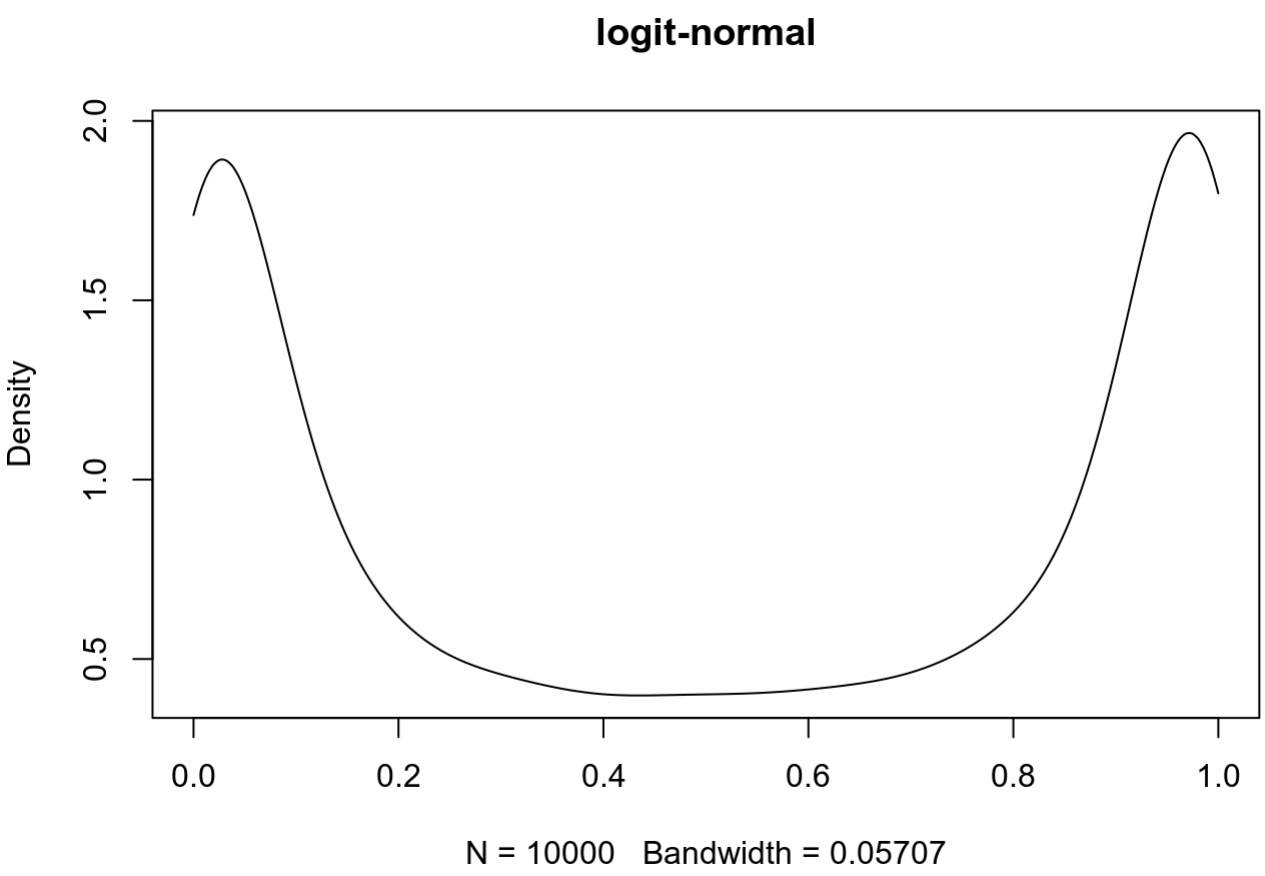
```
## Loading required package: Matrix
```

```
## Loading required package: lme4
```

```
##
## arm (Version 1.12-2, built: 2021-10-15)
```

```
## Working directory is /work/files/workspace/Baysian-inference/CODE
```

```r
library(LearnBayes)  #laplace function

## Binomial - logit-normal

# setwd("~/Patrick/Stat314-2017/code/")  #Should be unnecessary as
                                         ##data given in code

set.seed(29663)

### simple Binomial - logit-normal  example
## set-up the data
Y <- c(6,7,5,5,4,8)
N <- rep(10,length(Y))
NmY <- N-Y

#get some basic summary statistics
sumY <- sum(Y)
sumNmY <- sum(N-Y)

##We will assume a binomial likelihood but instead of a
##conjugate Beta(a,b) prior we will adopt a
##logit-normal prior for theta, i.e logit(theta) ~Normal(mu,sigma2)

#############################################################################
### Explore  the logit-normal density by plotting

#############################################################################
##first define a function to draw values from the logit-normal distribution.
##could also use the logitnorm package
rlogitnormal <- function(n,mu,sigma) {
  require(arm)
 l <- rnorm(n=n,mean=mu,sd=sigma)
 theta <- invlogit(l)
 return(theta)
}

##specify parameters of the prior for this illustration
mu_prior <- logit(0.5)
sigma_prior <- 4  ##Note this is sigma not sigma^2
testtheta <- rlogitnormal(n=10000,mu=mu_prior,sigma=sigma_prior)
plot(density(testtheta,from=0,to=1),main="logit-normal")
```




```
summary(testtheta)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 0.0000008 0.0642766 0.5183858 0.5056154 0.9418213 0.9999998
```

```
################################################################################
################

### Set up functions to evaluate the log-likelihood and log
#(unnormalized posterior)

################################################################################
################

###Function to compute the log-likelihood for a given
#logit(probability)

loglike_binom_logit <- function(eta,Y,N)  {
  theta <- invlogit(eta)
  logl <- dbinom(Y,N,prob=theta,log=TRUE)
  return(sum(logl))
}

##test this out
loglike_binom_logit(eta=0,Y=Y,N=N)
```

```
## [1] -11.2416
```

```
loglike_binom_logit(eta=-3,Y=Y,N=N)
```

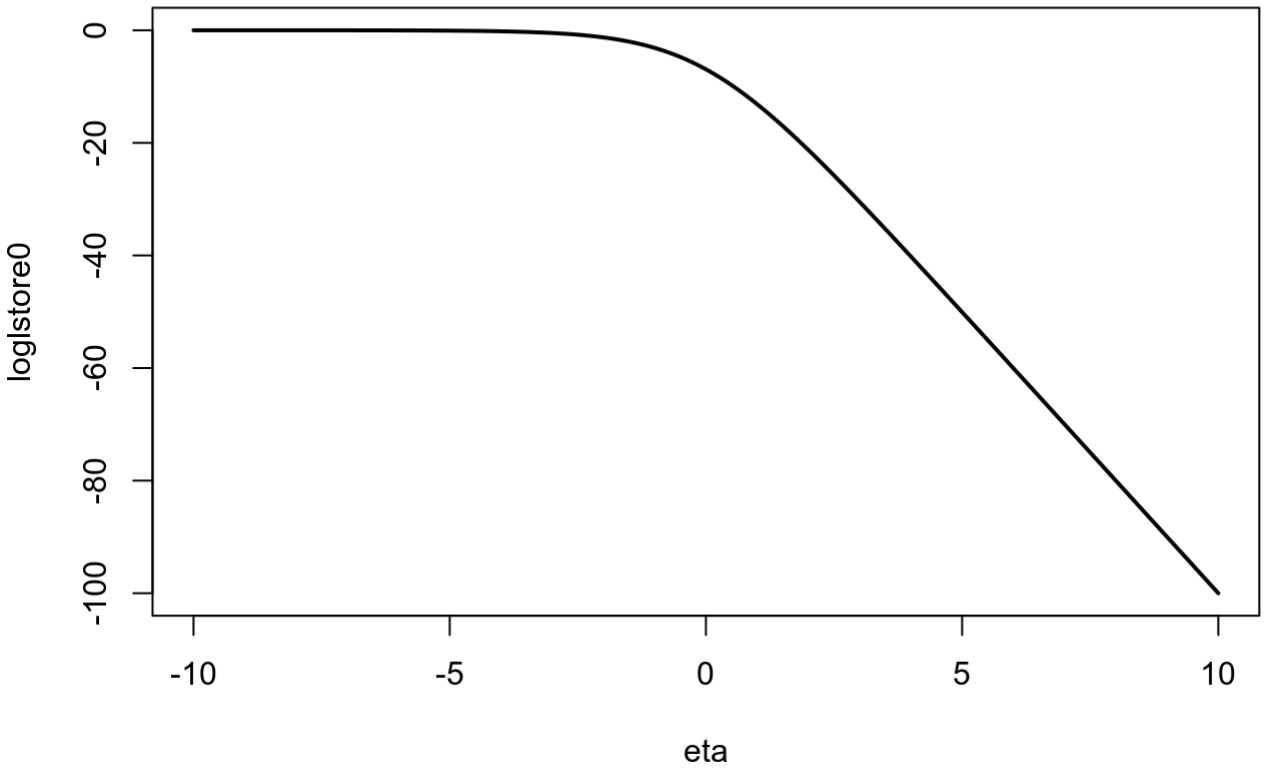
```
## [1] -77.56801
```

```
loglike_binom_logit(eta=3,Y=Y,N=N)
```

```
## [1] -47.56801
```

```
###plot the log likelihood as a function of eta

eta <- seq(from=-3,to=3,by=0.1) ##range over  which to evaluate the
                                ##log-likelihood
loglstore <- rep(NA,length(eta))  #object to store the log-likelihood
                                  #values
for (i in 1:length(eta)) {
  loglstore[i] <- loglike_binom_logit(eta=eta[i],Y=Y,N=N)
}


plot(eta,loglstore,type="l",lwd=2)
```

```
##Aside ###What if observe a sequence of zeros
Y0 <- rep(0,10)
N0 <- rep(1,10)

eta <- seq(from=-10,to=10,by=0.1)
loglstore0 <- rep(NA,length(eta))
for (i in 1:length(eta)) {
  loglstore0[i] <- loglike_binom_logit(eta=eta[i],Y=Y0,N=N0)
}

###plot the log likelihood as a function of eta

plot(eta,loglstore0,type="l",lwd=2)
```




```
arm::invlogit(-10)
```

```
## [1] 4.539787e-05
```

```
arm::invlogit(-5)
```

```
## [1] 0.006692851
```

```
loglike_binom_logit(eta=-10,Y=Y0,N=N0)
```

```
## [1] -0.000453989
```

```
loglike_binom_logit(-5,Y0,N0)
```

```
## [1] -0.06715348
```

```
loglike_binom_logit(0,Y0,N0)
```

```
## [1] -6.931472
```

```
###set-up a function to compute the log-unnormalised posterior
##note the likelihhod function is passed in as a parameter

logpost_binom_logit <- function(eta,likefunc,priormean,priorsd,Y,N) {
  logpost <- likefunc(eta,Y,N) +
    dnorm(eta,mean=priormean,sd=priorsd,log=TRUE)
  return(logpost)
}


##try out the log - posterior at a few values
testpost1 <- logpost_binom_logit(eta=logit(0.1),
                          likefunc=loglike_binom_logit,
                          priormean=mu_prior,priorsd=sigma_prior,Y=Y,N=N)
testpost1
```

```
## [1] -55.33337
```

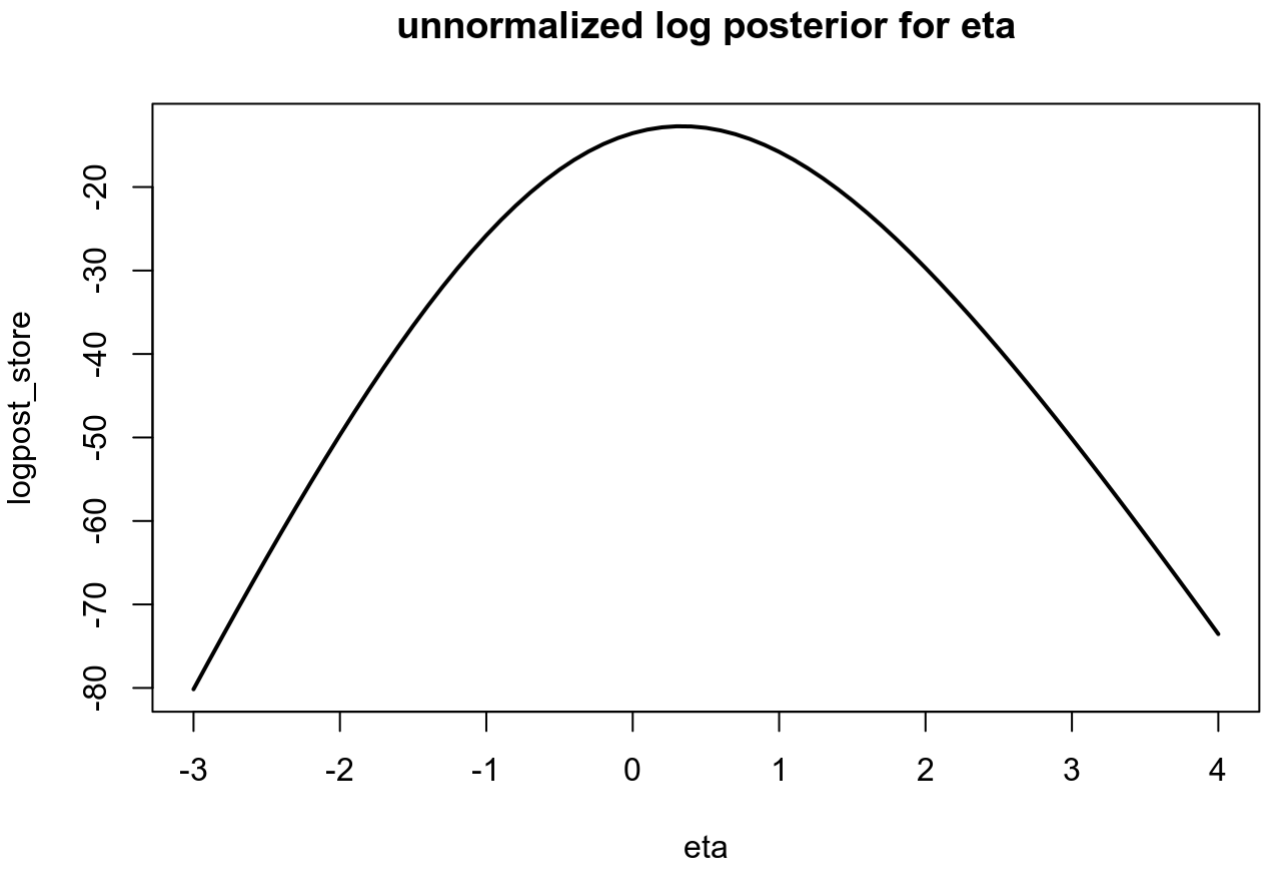
```
testpost5 <- logpost_binom_logit(eta=logit(0.5),
                              likefunc=loglike_binom_logit,
                              priormean=mu_prior,priorsd=sigma_prior,Y=Y,N=N)

testpost5
```

```
## [1] -13.54684
```

```
###plot the unormalized posterior for eta
eta <- seq(from=-3, to=4,by=0.1)
logpost_store <- rep(NA,length(eta))  ##vector to store values of the
                                      ## log-posterior
for (i in 1:length(eta)) {
  logpost_store[i] <- logpost_binom_logit(eta=eta[i],
                                  likefunc=loglike_binom_logit,
                                  priormean=mu_prior,priorsd=sigma_prior,
                                  Y=Y,N=N)
}

summary(logpost_store)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -80.15  -53.60  -33.41  -37.26  -18.44  -12.72
```

```
plot(cbind(eta,logpost_store),
     main="unnormalized log posterior for eta",type="l",lwd=2)
```




```
##############################################################################################
#####################
####Try and build an approximation to the posterior based on
#maximization of the log unnormalized posterior

#############################################################################
##try out laplace() from the LearnBayes packageto get posterior mode and
##variance approximation
##############################################################################################
############

posapprox <- laplace(logpost_binom_logit,mode=logit(0.5),
                     likefunc=loglike_binom_logit,priormean=mu_prior,
                     priorsd=sigma_prior,Y=Y,N=N)
posapprox
```

```
## $mode
## [1] 0.3349609
##
## $var
##             [,1]
## [1,] 0.06826173
##
## $int
## [1] -13.13639
##
## $converge
## [1] TRUE
```

```
mean_approx_eta <- posapprox$mode
sd_approx_eta <- sqrt(posapprox$var)
sd_approx_eta
```

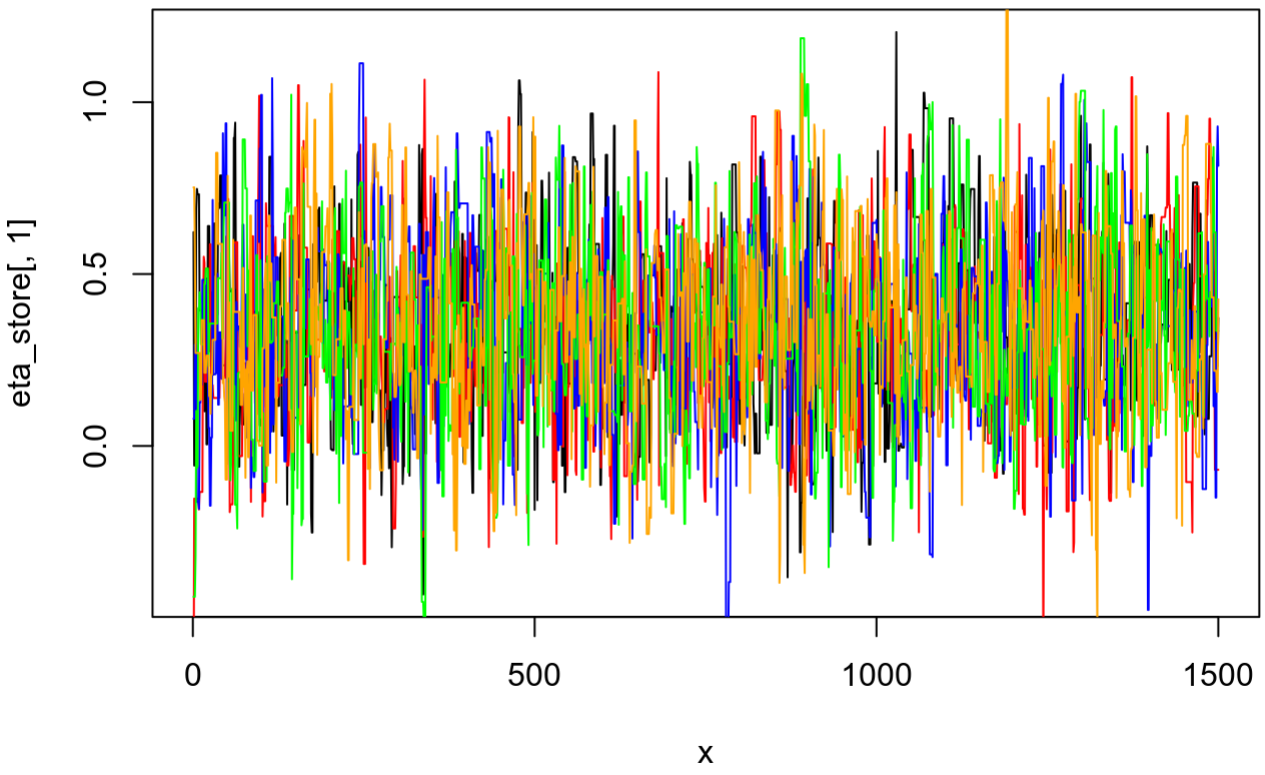
```
##           [,1]
## [1,] 0.2612695
```

```
###So our initial approximation is. Normal(mean_approx_eta,ad_approx_eta)

#We need an overdispersed approximation to the posterior to generate starting
# values hence:

sd_start <- 2 * sd_approx_eta   #Could inflate by more

##We will use random-walk Metropolis algorithm in
#which the jumping distribution is centred on the current value with
#variance  reccommended by Gelman et al: 2.4^2/d * Vapprox,
#d= dimension which is one in this case


sd_jump <- 2.4*sd_approx_eta

#sd_jump <- 0.01   ##experiment with poor jumping density

#############################################################################
## Set-up for Metropolis-Hastings
## We will use a symmetric jumping density -
##univariate normal in this case, as we are dealing with a scalar parameter
#############################################################################
nchains <- 5
simnum <- 1500
#burnin <- 500     ##hashed out because we will experiment with different
                   ##burn-in periods; display traceplots for


## set-up structures
eta_store <- matrix(nrow=simnum,ncol=nchains)
accept_store <- matrix(nrow=simnum,ncol=nchains)


for (j in 1:nchains) {
   ##draw initial value  ideally from over-dispersed approximation to
   ## posterior
  oldeta <- rnorm(n=1,mean=mean_approx_eta,sd=sd_start)
  for (i in 1:simnum) {
       ##draw proposal from the jumping distribution
    eta_prop <- rnorm(n=1,mean=oldeta,sd=sd_jump)
    ####symmetric jumping density so we can just us the metropolis version of the alg
orithm

    ##evaluate log posterior at the proposal  #log-likelihood + log prior
    logpost_prop <-  logpost_binom_logit(eta=eta_prop,
                                         likefunc=loglike_binom_logit,
                                         priormean=mu_prior,
                                         priorsd=sigma_prior,
                                         Y=Y,N=N)
    #
    ##evalute the log posterior at the current value
    ##evaluate log posterior at the proposal
    logpost_old <- logpost_binom_logit(eta=oldeta,
                                       likefunc=loglike_binom_logit,
                                       priormean=mu_prior,
                                       priorsd=sigma_prior,
                                       Y=Y,N=N)
```

```
    ##compute the acceptance ratio
    logrMH <- logpost_prop-logpost_old  #symmetric jumping density  so only need log
 posterior

    ##check whether  to accept
    accept <- (log(runif(1)) < logrMH)
    accept_store[i,j] <- as.numeric(accept)

    if (accept) {
      oldeta <- eta_prop
    }
    eta_store[i,j] <- oldeta
   } ###end loop over iterations
  }  ##end loop over chains

# check acceptance rates
colMeans(accept_store)
```

```
## [1] 0.4293333 0.4320000 0.4320000 0.4506667 0.4613333
```

```
###traceplots - all iterations
x <- seq(from=1,to=simnum,by=1)
plot(x,eta_store[,1],type="l")
lines(x,eta_store[,2],col="red")
lines(x,eta_store[,3],col="blue")
lines(x,eta_store[,4],col="green")
lines(x,eta_store[,5],col="orange")
```



```
##Gelman-rubin diagnostic
###drop burn_in
burnin <- 500
poseta <- eta_store[(burnin+1):simnum,]

##Chunk each  posterior  chains into  two pieces
## and reassemble as a matrix

possize <- nrow(poseta)

n1 <- round(possize/2)  ##size of first chunk

chunk1 <- poseta[1:n1,]
chunk2 <- poseta[(round(possize/2)+1):possize,]

poseta_chunked <- cbind(chunk1,chunk2)

str(poseta_chunked)
```

```
##  num [1:500, 1:10] 0.468 0.468 0.584 -0.158 0.366 ...
```

```
##obtain the chain means
chain_mean <- colMeans(poseta)
##obtain the within chain variance
chain_sd <- apply(poseta,MARGIN=2,FUN=sd)
chain_var <- chain_sd^2
W = mean(chain_var)
##get between chain variance

possize_chunked <- nrow(poseta_chunked)
B <- possize_chunked*(sd(chain_mean))^2

##Now build the components of the Gelman-Rubin statistic

Vplus <- ((possize_chunked-1)/possize_chunked) * W + (1/possize_chunked)*B
Rhat <- sqrt(Vplus/W)
Rhat
```

```
## [1] 1.003234
```

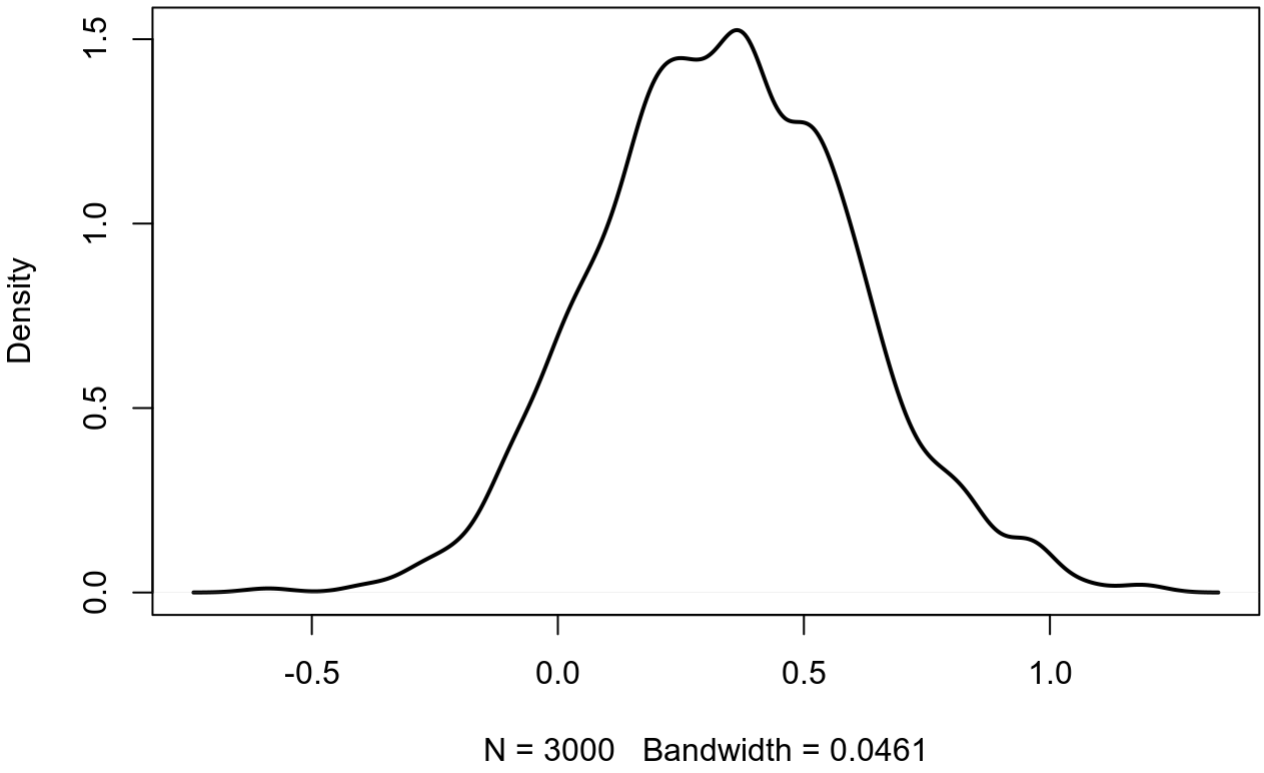
```
##compute  estimated effective sample size, using Coda

poseta.mcmc <- coda::as.mcmc(poseta_chunked )
poseta.mcmc.list <- coda::mcmc.list(poseta.mcmc[,1],
                                    poseta.mcmc[,2],
                                    poseta.mcmc[,3],
                                    poseta.mcmc[,4],
                                    poseta.mcmc[,5],
                                    poseta.mcmc[,6],
                                    poseta.mcmc[,7],
                                    poseta.mcmc[,8],
                                    poseta.mcmc[,9],
                                    poseta.mcmc[,10] )

codaNeff <- coda::effectiveSize(poseta.mcmc.list)

codaNeff
```

```
##     var1
## 1076.955
```

```
##summarise the posterior for eta
##combine posterior samples from the separate chains

poseta_pooled <- poseta[,1]
for (j in 2:nchains) {
  poseta_pooled <- c(poseta_pooled,poseta_chunked[,j])
}

summary(poseta_pooled)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.6022  0.1734  0.3424  0.3418  0.5138  1.2039
```

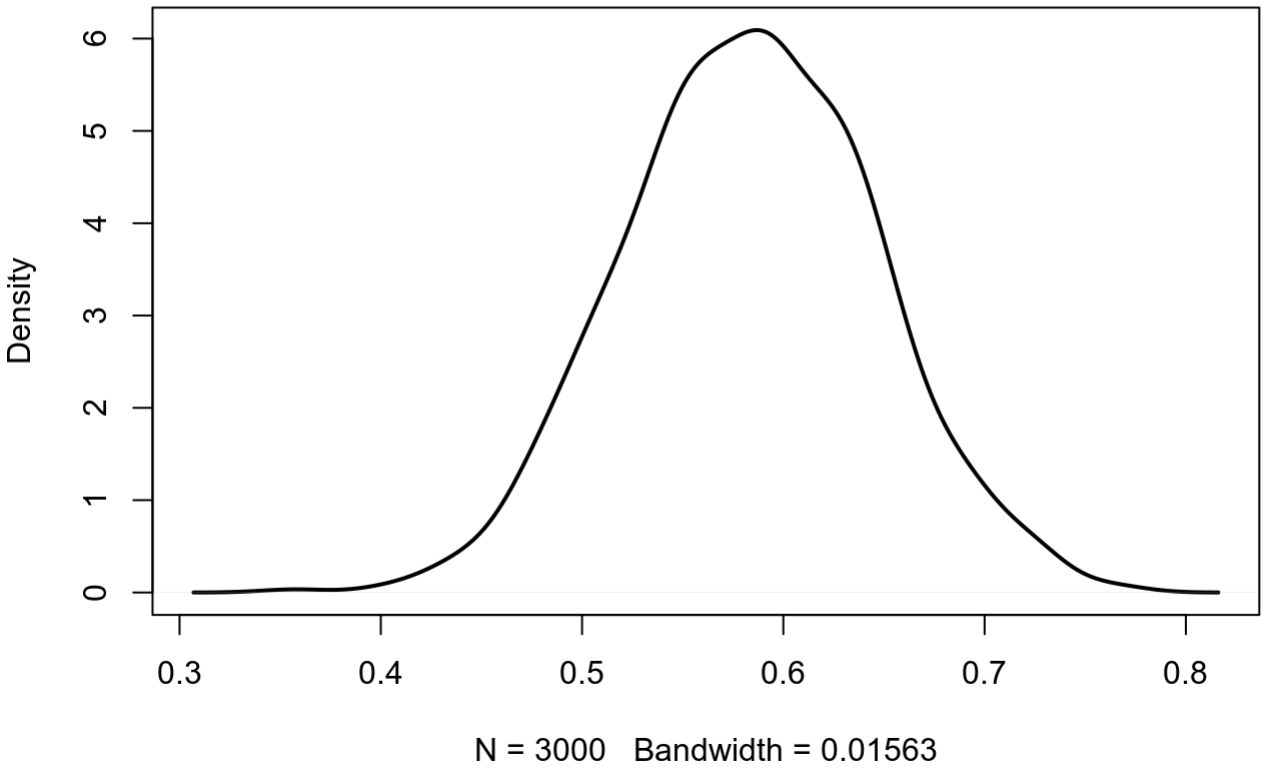
```
plot(density(poseta_pooled),main="posterior for eta",lwd=2)
```




```
##should transform back to probability scale
postheta_pooled <- invlogit(poseta_pooled)
summary(postheta_pooled)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.3538  0.5433  0.5848  0.5833  0.6257  0.7692
```

```
plot(density(postheta_pooled,adjust=1.4),main="posterior for theta",lwd=2)
```




```
##95% credible interval  for theta
quantile(postheta_pooled,probs=c(0.025,0.5,0.975))
```

```
##      2.5%       50%     97.5%
## 0.4650201 0.5847829 0.7029762
```

```
##probability theta > 0.75
mean((postheta_pooled > 0.75))
```

```
## [1] 0.002333333
```

```
summary(postheta_pooled)
```

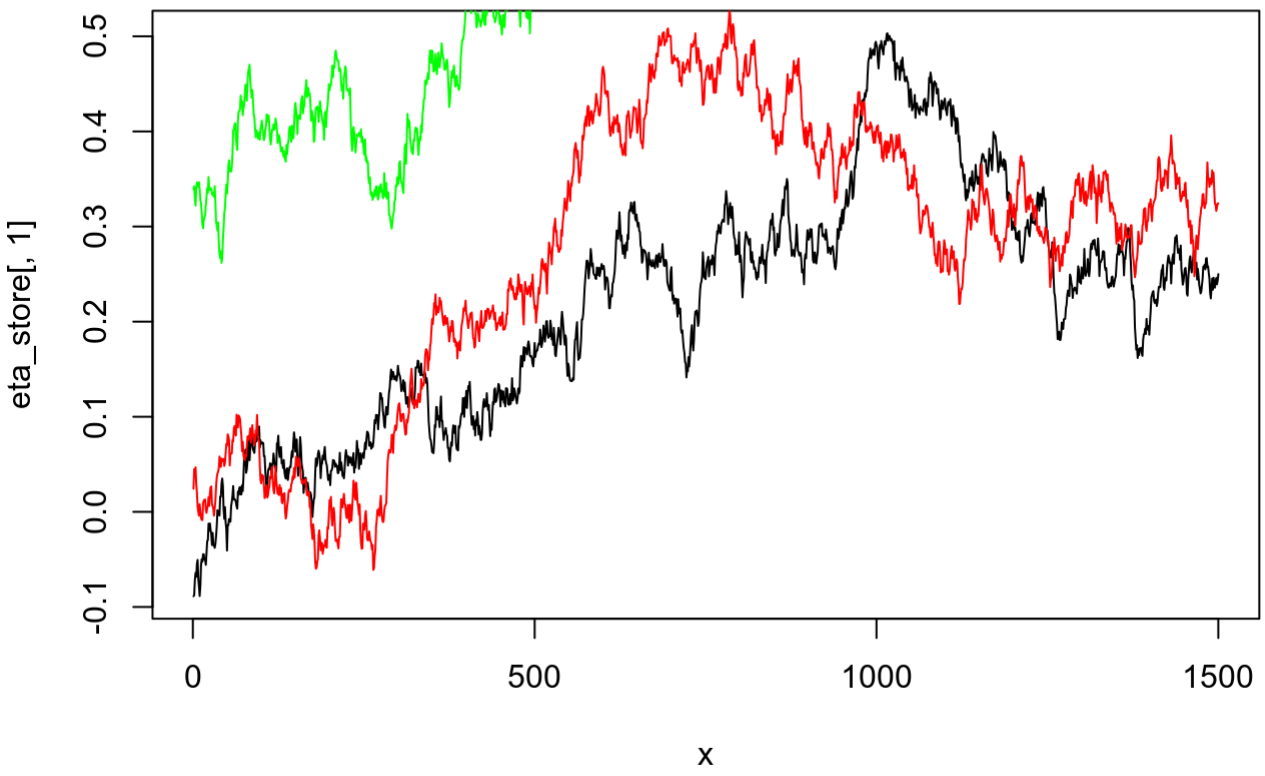
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.3538  0.5433  0.5848  0.5833  0.6257  0.7692
```

```
# Illustrate what  happens with  a  poorly chosen jumping distribution --------

sd_jump <- 0.01   ##experiment with poor jumping density

#########################################################################
## Set-up for Metropolis-Hastings
## We will use a symmetric jumping density -
##univariate normal in this case as we are dealing with a scalar parameter
#########################################################################
nchains <- 5
simnum <- 1500
#burnin <- 500     ##hashed out because we will experiment with different burn - in
##periods; display traceplots for


## set-up structures
eta_store <- matrix(nrow=simnum,ncol=nchains)
accept_store <- matrix(nrow=simnum,ncol=nchains)


for (j in 1:nchains) {
  ##draw initial value  ideally from over-dispersed approximation to
  ## posterior
  oldeta <- rnorm(n=1,mean=mean_approx_eta,sd=sd_start)
  for (i in 1:simnum) {
    ##draw proposal from the jumping distribution
    eta_prop <- rnorm(n=1,mean=oldeta,sd=sd_jump)
    ####symmetric jumping density so we can just us the metropolis version of the alg
orithm

    ##evaluate log posterior at the proposal  #log-likelihood + log prior
    logpost_prop <-  logpost_binom_logit(eta=eta_prop,
                                         likefunc=loglike_binom_logit,
                                         priormean=mu_prior,
                                         priorsd=sigma_prior,
                                         Y=Y,N=N)
    #
    ##evalute the log posterior at the current value
    ##evaluate log posterior at the proposal
    logpost_old <- logpost_binom_logit(eta=oldeta,
                                       likefunc=loglike_binom_logit,
                                       priormean=mu_prior,
                                       priorsd=sigma_prior,
                                       Y=Y,N=N)

    ##compute the acceptance ratio
    logrMH <- logpost_prop-logpost_old  #symmetric jumping density  so only need log
 posterior

    ##check whether  to accept
    accept <- (log(runif(1)) < logrMH)
    accept_store[i,j] <- as.numeric(accept)

    if (accept) {
      oldeta <- eta_prop
    }
    eta_store[i,j] <- oldeta
```

```
  } ###end loop over iterations
}  ##end loop over chains

# check acceptance rates
colMeans(accept_store)
```

```
## [1] 0.9853333 0.9920000 0.9653333 0.9840000 0.9580000
```

```
###traceplots - all iterations
x <- seq(from=1,to=simnum,by=1)
plot(x,eta_store[,1],type="l")
lines(x,eta_store[,2],col="red")
lines(x,eta_store[,3],col="blue")
lines(x,eta_store[,4],col="green")
lines(x,eta_store[,5],col="orange")
```



```
##Gelman-rubin diagnostic
###drop burn_in
burnin <- 500
poseta <- eta_store[(burnin+1):simnum,]

##Chunk each  posterior  chains into  two pieces
## and reassemble as a matric

possize <- nrow(poseta)

n1 <- round(possize/2)  ##size of first chunk

chunk1 <- poseta[1:n1,]
chunk2 <- poseta[(round(possize/2)+1):possize,]

poseta_chunked <- cbind(chunk1,chunk2)

str(poseta_chunked)
```

```
##  num [1:500, 1:10] 0.163 0.168 0.169 0.17 0.175 ...
```

```
##obtain the chain means
chain_mean <- colMeans(poseta)
##obtain the within chain variance
chain_sd <- apply(poseta,MARGIN=2,FUN=sd)
chain_var <- chain_sd^2
W = mean(chain_var)
##get between chain variance

possize_chunked <- nrow(poseta_chunked)
B <- possize_chunked*(sd(chain_mean))^2

##Now build the components of the Gelman-Rubin statistic

Vplus <- ((possize_chunked-1)/possize_chunked) * W + (1/possize_chunked)*B
Rhat <- sqrt(Vplus/W)
Rhat
```

```
## [1] 5.370172
```

```
##compute  estimated effective sample size, using Coda

poseta.mcmc <- coda::as.mcmc(poseta_chunked )
poseta.mcmc.list <- coda::mcmc.list(poseta.mcmc[,1],
                                    poseta.mcmc[,2],
                                    poseta.mcmc[,3],
                                    poseta.mcmc[,4],
                                    poseta.mcmc[,5],
                                    poseta.mcmc[,6],
                                    poseta.mcmc[,7],
                                    poseta.mcmc[,8],
                                    poseta.mcmc[,9],
                                    poseta.mcmc[,10] )

codaNeff <- coda::effectiveSize(poseta.mcmc.list)

codaNeff
```

```
##     var1
## 55.35286
```

```
##so high acceptance rates are no necessarily a good thing
```