U20 Math 559 Bayesian Statistics Homework 2

Solution

Instruction: Please type or write your answers clearly and show your work. You are encouraged to use the Rmarkdown version of this assignment as a template to submit your work. Unless stated otherwise, all programming references in the assignment will be in `R`, and the predefined `R` functions used for the problems can all be found on our Canvas site under `DBDA2Eprograms.zip`, unless specified otherwise. For this assignment, problems roughly covers content from lectures 4-6.

Problem 1

At the end of the script `BernMetrop.R`, add these lines:

```
openGraph(height=7,width=3.5)
layout(matrix(1:2,nrow=2))
acf( acceptedTraj , lag.max=30 , col="skyblue" , lwd=3 )
Len = length( acceptedTraj )
Lag = 10
trajHead = acceptedTraj[  1       : (Len-Lag) ]
trajTail = acceptedTraj[ (1+Lag) :  Len       ]
plot( trajHead , trajTail , pch="." , col="skyblue" ,
      main=bquote( list( "Prpsl.SD" == .(proposalSD) ,
                          lag == .(Lag) ,
                          cor == .(round(cor(trajHead,trajTail),3)))) )
```

a) Before each line, add a comment that explains what the line does. Include the commented code in your write-up.

Answer:

Let's begin with the first code-chunk, which reads in the pre-defined utility functions:

```
graphics.off()
# remove existing environments
rm(list=ls(all=TRUE))
fileNameRoot="BernMetrop" # for output filenames
#load in pre-defined utility functions
source("../DBDA2E-utilities.R")
```

```
##
## *********************************************************************
## Kruschke, J. K. (2015). Doing Bayesian Data Analysis, Second Edition:
## A Tutorial with R, JAGS, and Stan. Academic Press / Elsevier.
## *********************************************************************
```

In the next step, simulated data is defined. Note that `c(rep(0,6), rep(1,14))` represents a sequence of 6 tails and 14 heads.

```
# Specify the data, to be used in the likelihood function.
myData = c(rep(0,6),rep(1,14))
```

Defining the likelihood function for the data, which in this case is the Bernoulli distribution:

```
# Define the Bernoulli likelihood function, p(D|theta).
# The argument theta could be a vector, not just a scalar.
likelihood = function( theta , data ) {
  z = sum( data )
  N = length( data )
  pDataGivenTheta = theta^z * (1-theta)^(N-z)
  # The theta values passed into this function are generated at random,
  # and therefore might be inadvertently greater than 1 or less than 0.
  # The likelihood for theta > 1 or for theta < 0 is zero:
  pDataGivenTheta[ theta > 1 | theta < 0 ] = 0
  return( pDataGivenTheta )
}
```

Defining the prior density function, which is a Beta distribution with parameters a and b

```
# Define the prior density function.
prior = function( theta ) {
  pTheta = dbeta( theta , 1 , 1 )
  # The theta values passed into this function are generated at random,
  # and therefore might be inadvertently greater than 1 or less than 0.
  # The prior for theta > 1 or for theta < 0 is zero:
  pTheta[ theta > 1 | theta < 0 ] = 0
  return( pTheta )
}
```

Defining the relative probability of the target distribution, $p(D|\theta) * p(\theta)$ (likelihood times prior), which is proportional to the posterior distribution

```
# Define the relative probability of the target distribution,
# as a function of vector theta. For our application, this
# target distribution is the unnormalized posterior distribution.
targetRelProb = function( theta , data ) {
  targetRelProb =  likelihood( theta , data ) * prior( theta )
  return( targetRelProb )
}
```

Configure parameters for the Markov chain by defining the length of the trajectory and burn-in period

```
# Specify the length of the trajectory, i.e., the number of jumps to try:
trajLength = 50000 # arbitrary large number
# Initialize the vector that will store the results:
trajectory = rep( 0 , trajLength )
# Specify where to start the trajectory:
trajectory[1] = 0.01 # arbitrary value
# Specify the burn-in period:
burnIn = ceiling( 0.0 * trajLength ) # arbitrary number, less than trajLength
# Initialize accepted, rejected counters, just to monitor performance:
nAccepted = 0
nRejected = 0
```

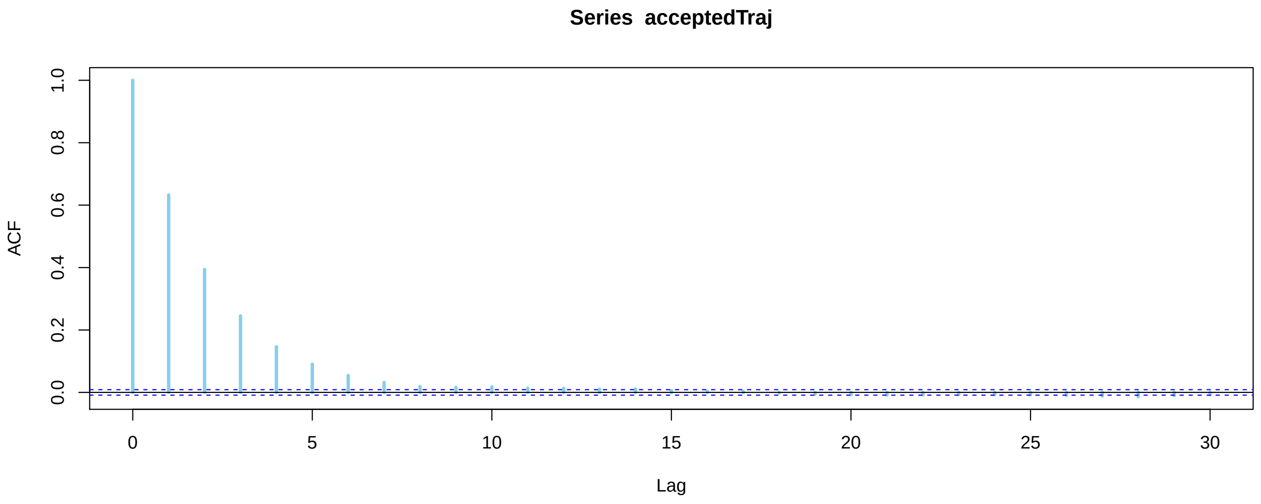
Generating the random walk - this is how theta gets updated in each iteration:

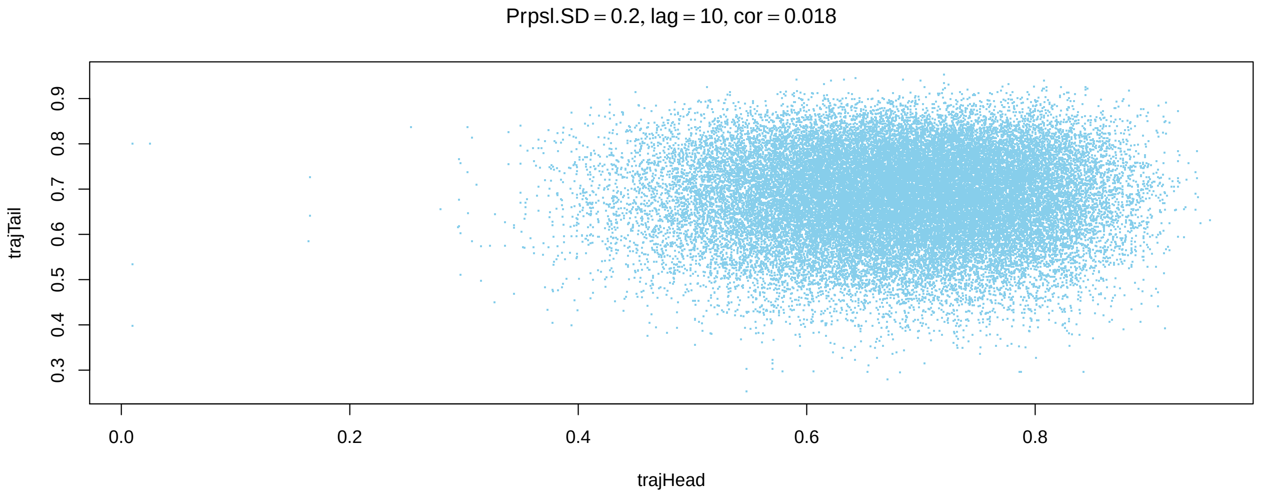
```
# Now generate the random walk. The 't' index is time or trial in the walk.
# Specify seed to reproduce same random walk:
set.seed(47405)
# Specify standard deviation of proposal distribution:
proposalSD = c(0.02,0.2,2.0)[2]
```

```r
for ( t in 1:(trajLength-1) ) {
  currentPosition = trajectory[t]
  # Use the proposal distribution to generate a proposed jump.
  proposedJump = rnorm( 1 , mean=0 , sd=proposalSD )
  # Compute the probability of accepting the proposed jump.
  probAccept = min( 1,
                    targetRelProb( currentPosition + proposedJump , myData )
                    / targetRelProb( currentPosition , myData ) )
  # 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 }
  }
}
```

Collect the trajectory of values for θ and visualize the results:

```r
plot.new()
# Extract the post-burnIn portion of the trajectory.
acceptedTraj = trajectory[ (burnIn+1) : length(trajectory) ]
#openGraph(height=7,width=3.5)
# set matrix size to allocate figures
layout(matrix(1:2,nrow=2))
# call acf (auto-correlation function) for the trajectory of theta from the Markov chain
acf( acceptedTraj , lag.max=30 , col="skyblue" , lwd=3 )
# define length of the chain
Len = length( acceptedTraj )
# set lag size
Lag = 10
# set sequence of accepted trajectories to visualize the following pattern:
## (\theta_{10i}} vs theta_{i}, i = 1,2,3,...size of the chain)
trajHead = acceptedTraj[  1      : (Len-Lag) ]
trajTail = acceptedTraj[ (1+Lag) :  Len      ]

plot( trajHead , trajTail , pch="." , col="skyblue" ,
      main=bquote( list( "Prpsl.SD" == .(proposalSD) ,
                         lag == .(Lag) ,
                         cor == .(round(cor(trajHead,trajTail),3)))) )
```



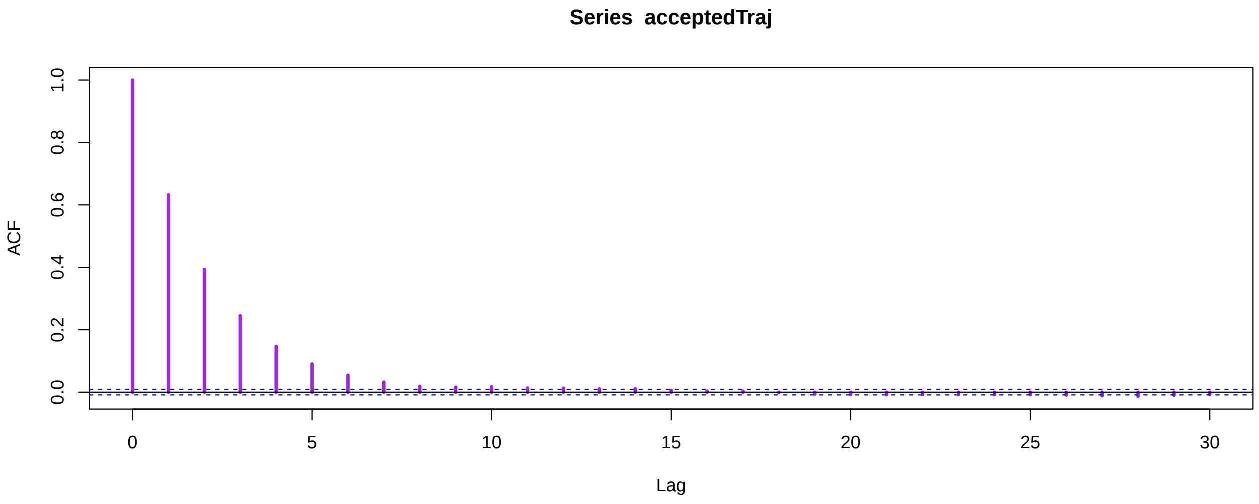


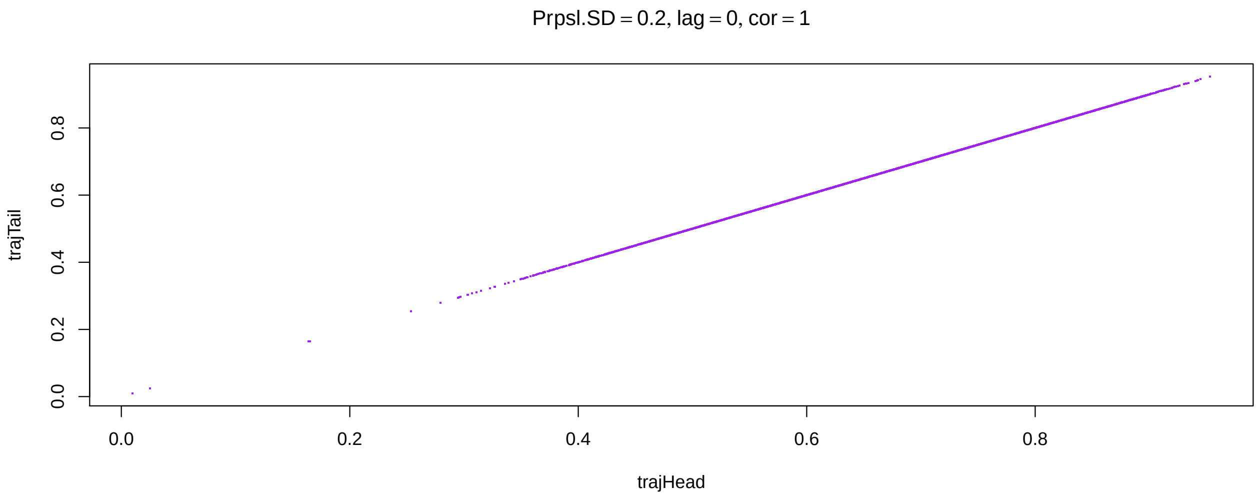
```
graphics.off()
```

b) Repeat the previous exercise, with the lines above appended to the script. Include the resulting new graphs in your write-up. For each run, verify that the height of the ACF bar at the specified lag matches the correlation in the scatterplot.

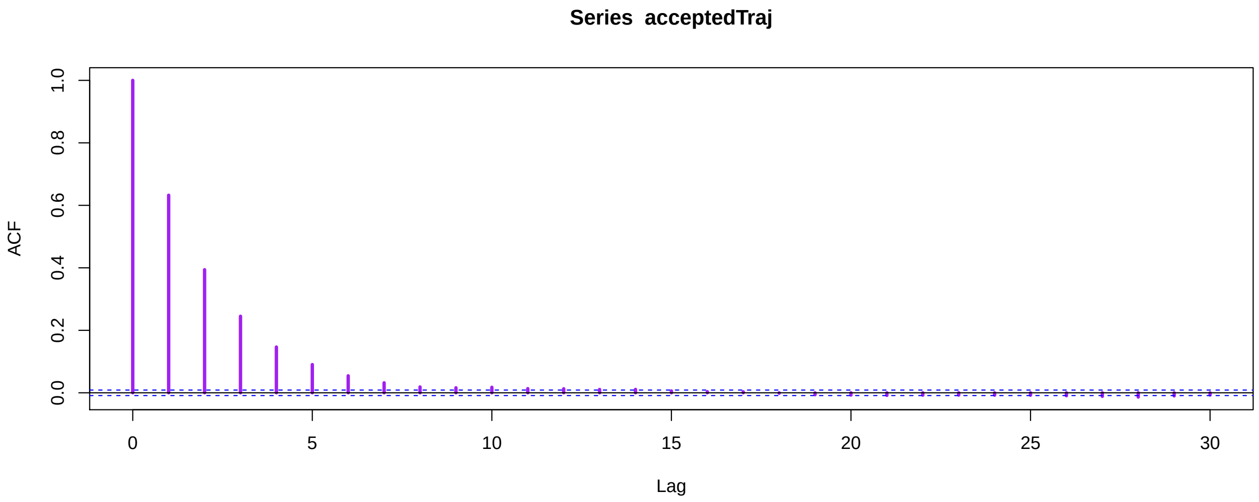
Let's set a few different values of the lag. This is done by modifying the value for `Lag` in the `BernMetrop.R` script:

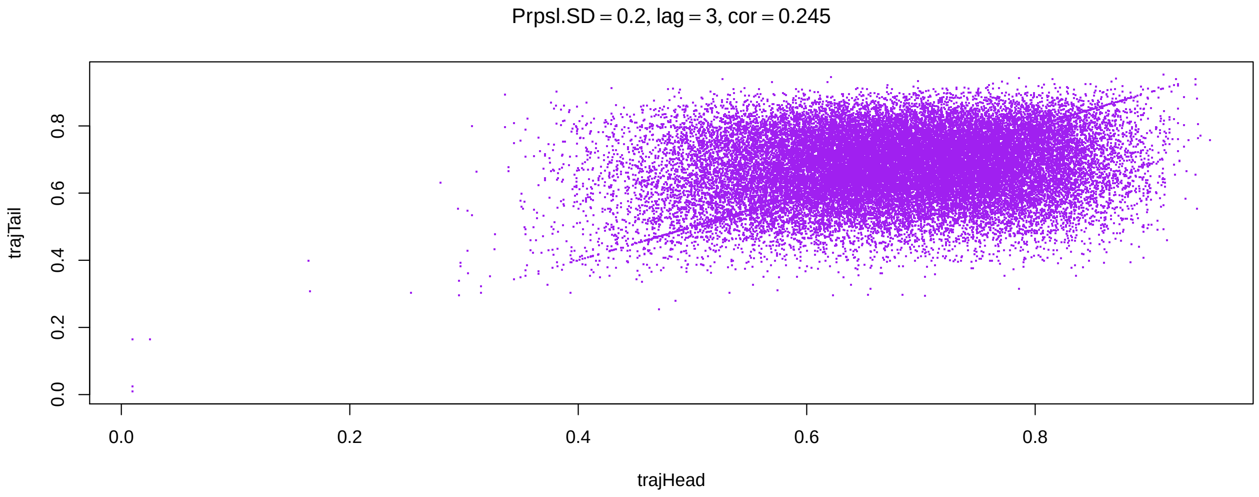
When lag = 0 (see code below), we can see that the scatterplot is an identity line, which corresponds to an autocorrelation of 1:


Series acceptedTraj
ACF
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30
Lag


Prpsl.SD = 0.2, lag = 0, cor = 1
trajTail
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
trajHead

lag $= 3$:

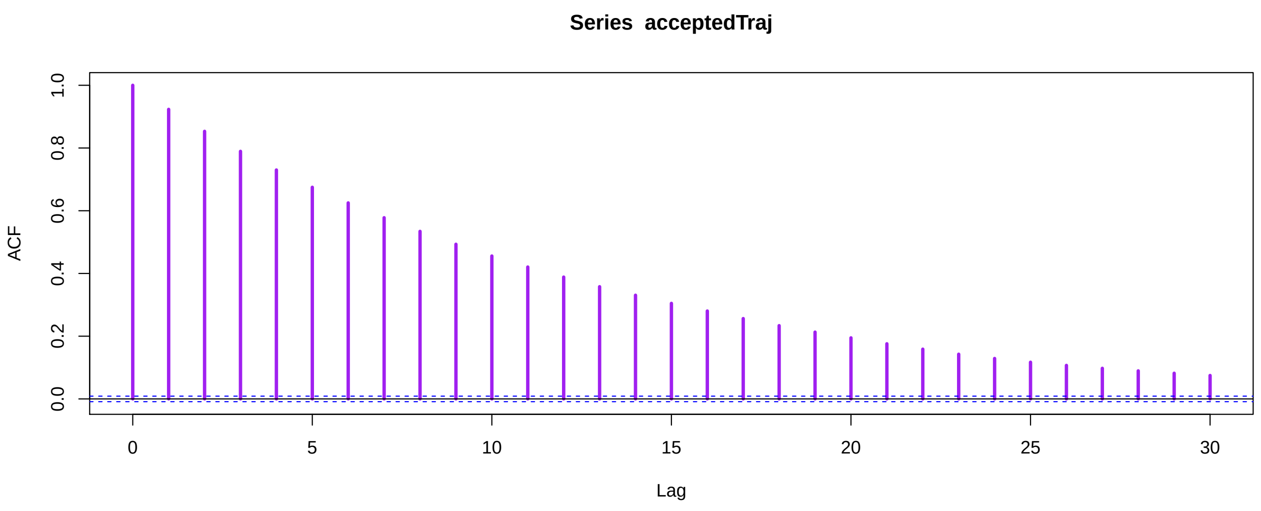


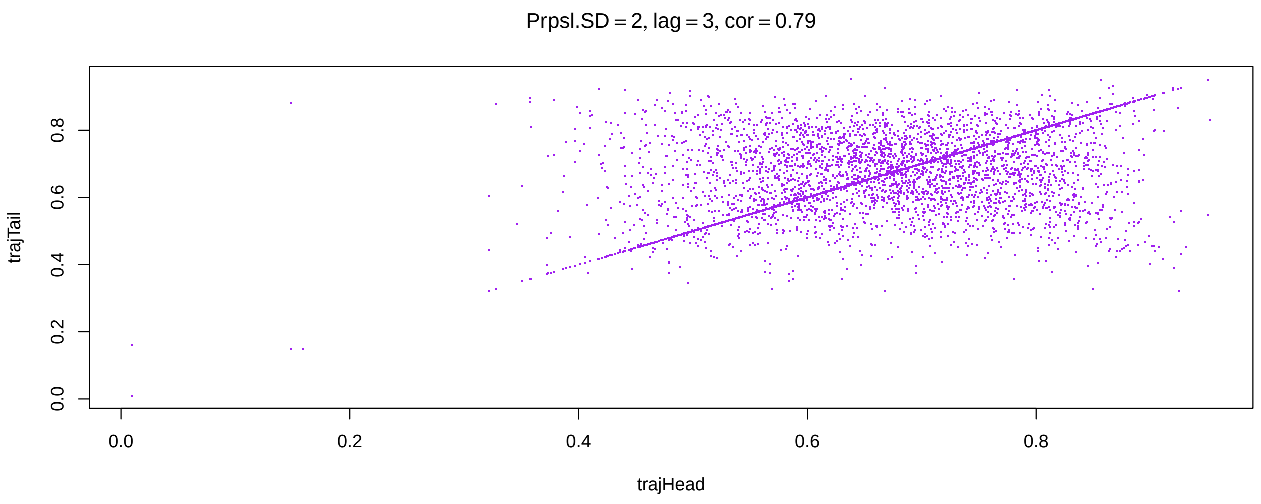


We can see from the above examples that the values of the ACF plots indeed correspond to those from the scatterplot.

c) When the proposal distribution has SD = 2, why does the scatter plot have a dense line of points on the diagonal?

Let's first set the proposal SD to 2, we can do that by replacing the term `proposalSD` in `BernMetrop.R` to 0.2





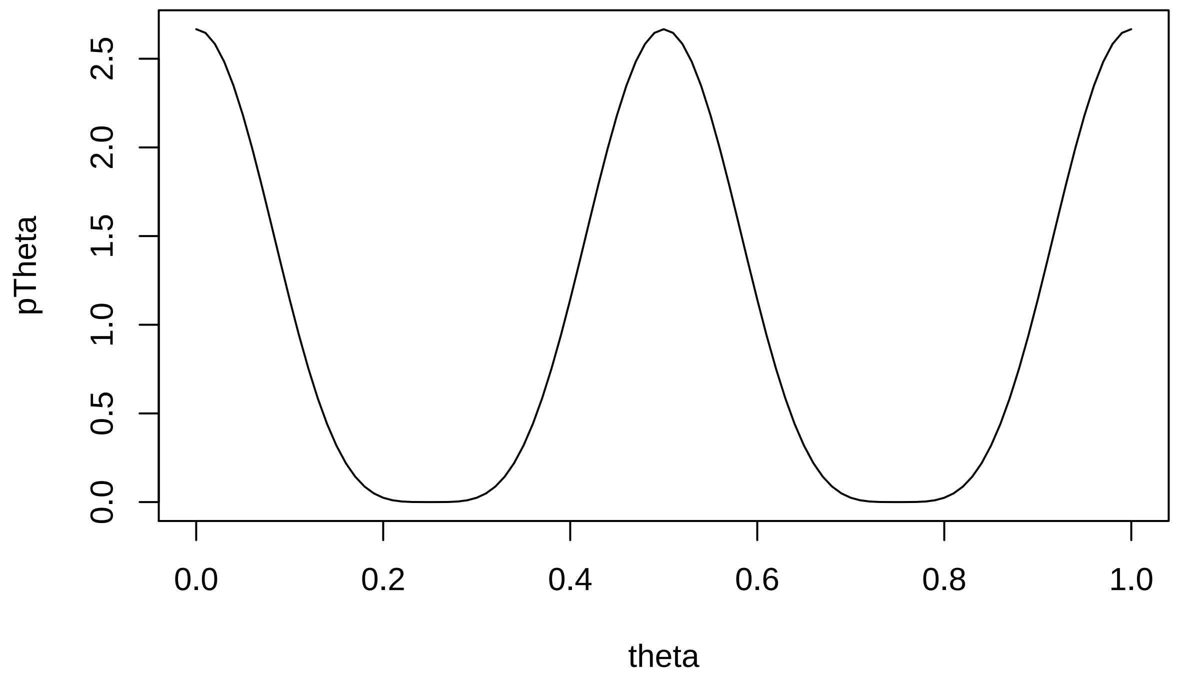
When the proposal SD is 2.0, the scatter plot has a dense line of points on the diagonal because of the relative high number of steps it takes to change a value. For most steps, the value of the parameters at step i remains the same as the value at step $i + 10$, that is, the values do not change even after 10 proposals. Thus, acceptedTraj[i] = acceptedTraj[i+10], and the plotted point falls on the main diagonal.

Problem 2

Consider a prior distribution on coin bias that puts most credibility at 0.0, 0.5, and 1.0, which we will formulate as $p(\theta) = \frac{(\cos(4\pi\theta)+1)^2}{1.5}$.

a) Plot the prior as a function of θ.

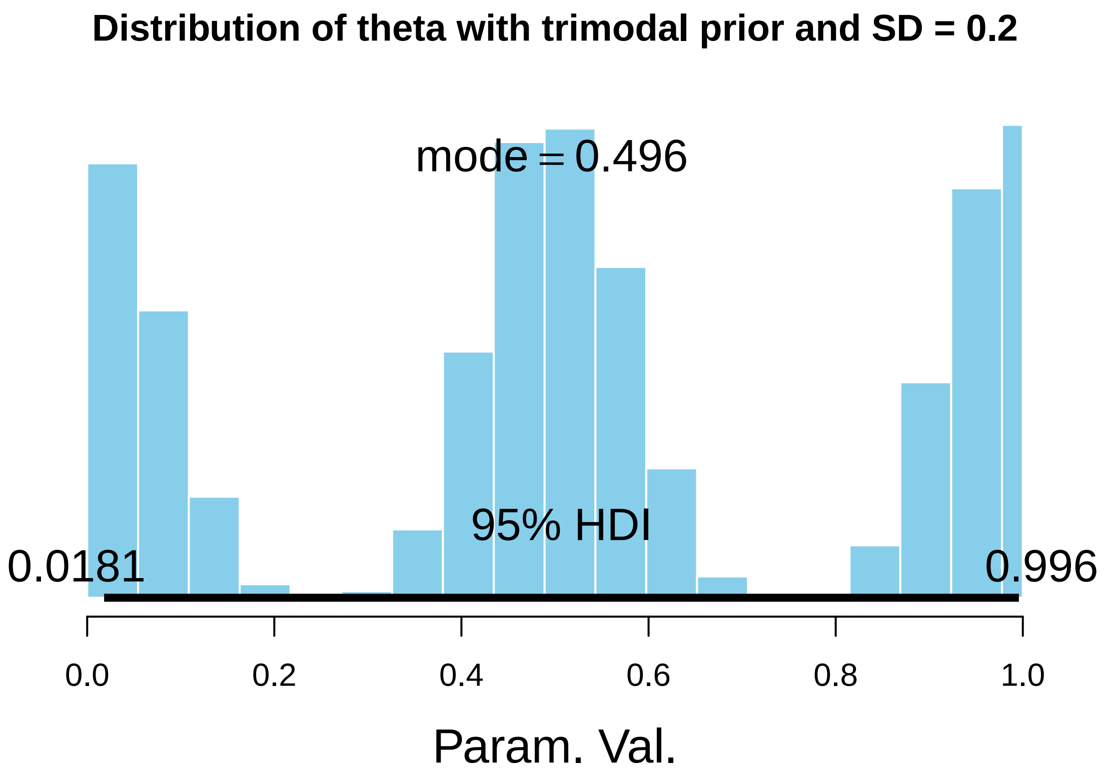
```r
theta <- seq(0,1,by=0.01)

pTheta <- ((cos(4*pi*theta)+1)^2)/1.5

plot(theta,pTheta,type = 'l')
```



b) In the script `BernMetrop.R`, find the function definition that specifies the prior distribution, replace it with the function defined in part a), set `myData = c()`. and run the script, using a proposal SD=0.2. Include the graphical output in your write-up. Does the histogram of the trajectory look like the graph of the previous part of the exercise?

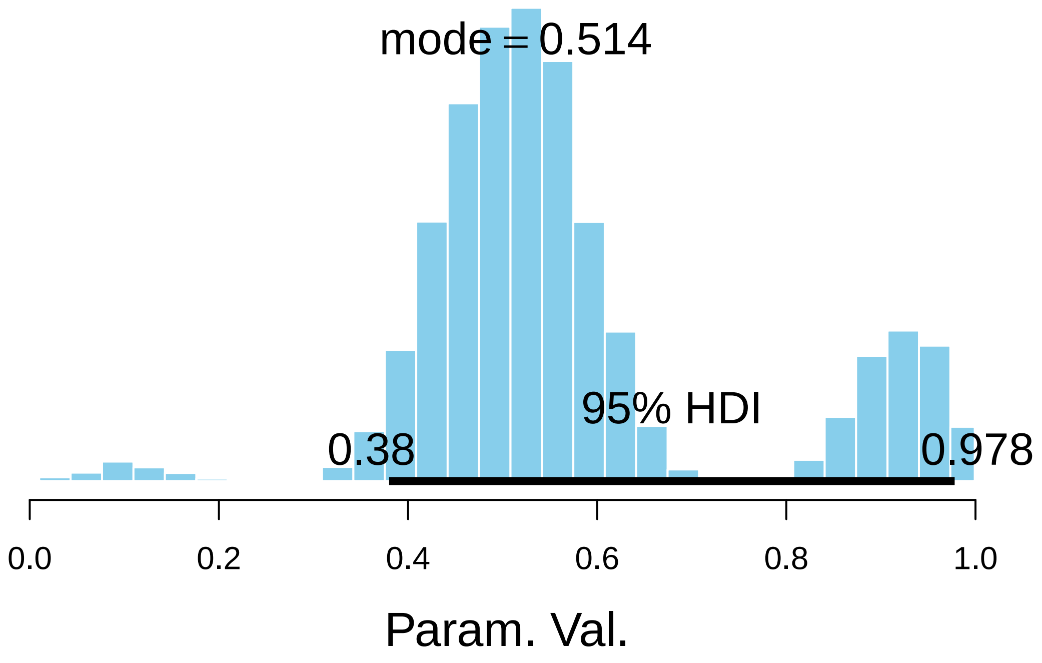
Answer, replace the prior function in the `BernMetrop.R` to the following, we'll notice a trimodel prior with the most credibilities allocated around $\theta = 0$, $\theta = 0.5$ and $\theta = 1$.

```r
# Define the prior density function.
prior = function( theta ) {
  pTheta = ((cos(4*pi*theta)+1)^2)/1.5 # replace pTheta with the function provided
  # The prior for theta > 1 or for theta < 0 is zero:
  pTheta[ theta > 1 | theta < 0 ] = 0
  return( pTheta )
}
```



```
##                    ESS      mean    median      mode hdiMass    hdiLow   hdiHigh
## Param. Val. 1274.139 0.5046057 0.5010816 0.4964752    0.95 0.0180977 0.9957966
##              compVal pGtCompVal ROPElow ROPEhigh pLtROPE pInROPE pGtROPE
## Param. Val.       NA         NA      NA       NA      NA      NA      NA
```

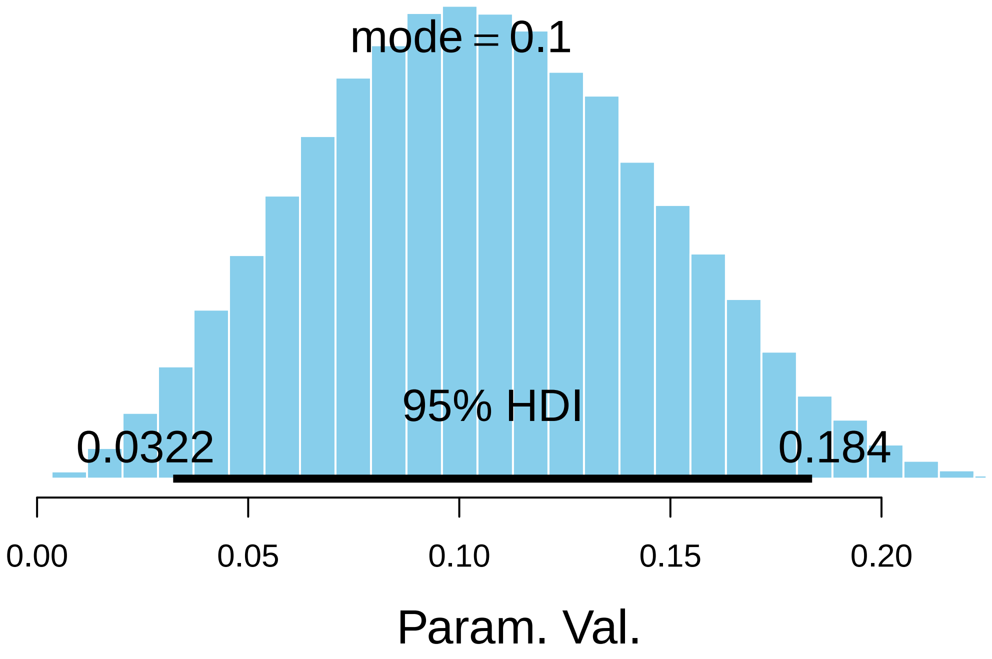
c) Repeat the previous part but now with `myData = c(0,1,1)`. Include the graphical output in your write-up. Does the posterior distribution make sense? Explain why.



```
##                    ESS      mean    median      mode hdiMass    hdiLow   hdiHigh
## Param. Val. 2743.738 0.5707842 0.5280971 0.5138546    0.95 0.3799788 0.9779718
##              compVal pGtCompVal ROPElow ROPEhigh pLtROPE pInROPE pGtROPE
## Param. Val.      NA         NA      NA       NA      NA      NA      NA
```

d) Repeat the previous part but now with proposal SD=0.02. Include the graphical output in your write-up. Does the posterior distribution make sense? Explain why not; what has gone wrong? If we did not know from the previous part that this output was unrepresentative of the true posterior, how could we try to check?

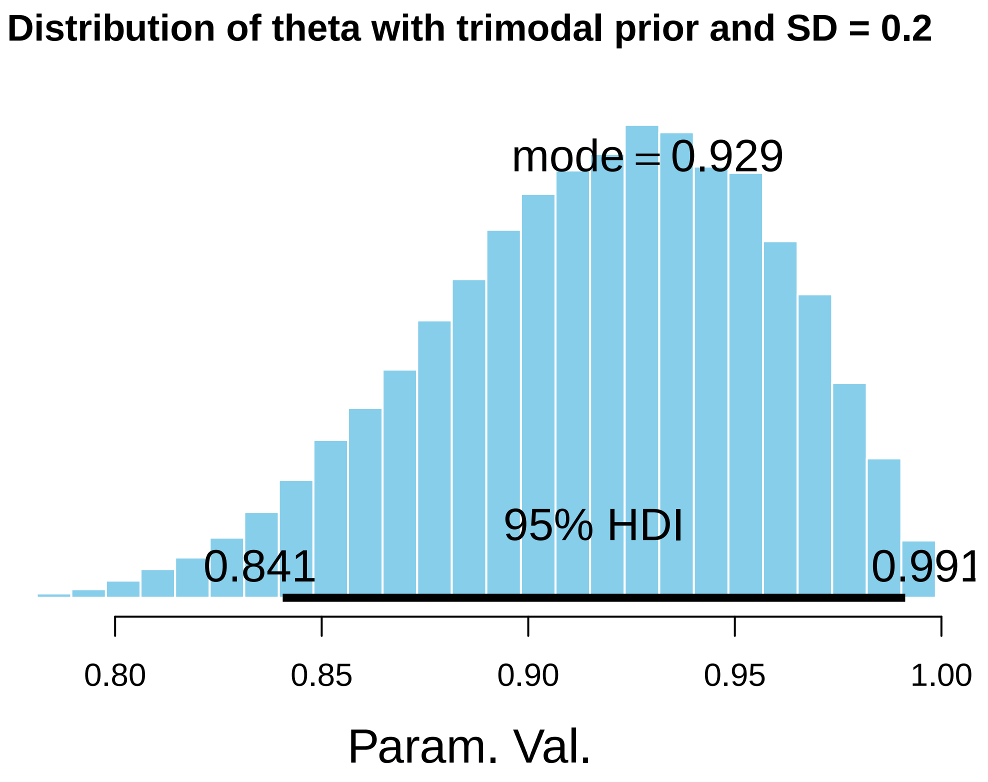
Answer: In here we keep `myData = c(0,1,1)` and set `SD = 0.02`. Notice that in our data, the prior assumption of θ should be closer to 2/3, yet in the posterior distribution we observe a value that is relatively closer to 0. This is because the arbitrary initial value of the chain is set to very close to 0, and the very small proposal SD has a difficult time squeezing under the very low ceiling between modes of the prior. With a long enough wait, there might be a series of proposals that just happen to make it under the low ceiling, but it would take much more than the 50000 steps we allocated.



```
##                  ESS      mean    median     mode hdiMass     hdiLow   hdiHigh
## Param. Val. 2366.936 0.1057002 0.1044647 0.100428    0.95 0.03223543 0.1835666
##              compVal pGtCompVal ROPElow ROPEhigh pLtROPE pInROPE pGtROPE
## Param. Val.      NA         NA      NA       NA      NA      NA      NA
```

e) Repeat the previous part but now with the initial position at 0.99. In conjunction with the previous part, what does this result tell us?

Answer: replacing the value for `trajectory[1]` from the previous 0.01 to 0.99, and keeping the `proposalSD` at 0.02 (so we take tiny step for θ at each iteration), we see in this case the mode of the posterior distribution is much closer to 0.99 instead. This indicates that the posterior distribution is highly sensitive to the choice of initial value, and in the case of the sampling algorithm, is also highly dependent on the standard deviation which forms the distribution where we sample the step size.



```
##                    ESS      mean    median      mode hdiMass    hdiLow   hdiHigh
## Param. Val. 2271.765 0.9167814 0.9204456 0.9289348    0.95 0.8405652 0.991233
##             compVal pGtCompVal ROPElow ROPEhigh pLtROPE pInROPE pGtROPE
## Param. Val.      NA         NA      NA       NA      NA      NA      NA
```

Problem 3

Recall that one way to estimate the bias of a coin using the Bayesian approach is by modeling the series of coin flips as i.i.d. Bernoulli trials, and using the beta distribution to model its prior.

a) What do the Beta distribution parameters `a` and `b` represent in the context of our prior belief about the bias of a coin?

Answer: `a` represents the number of 'heads' we have observed (or belived) based on prior knowledge or experience, while `b` represents the tail we have observed. This could be explained as our understanding of a specific type of coins, for example, that a US dime tends to be a fair coin compared to a quarter based on the physical attributes of the coins.

b) Suppose we have a data set of N coin flips where we observed z heads, assuming the prior distribution is Beta with parameters a and b, what can we say about the mean and the mode for the resulting posterior distribution?

Answer: First of all, since we have the pair of conjugate distributions (Beta prior + Bernoulli likelihood), the posterior distribution will end up being a Beta distribution as well, with parameters $Beta(z + a, N - z + b)$. Therefore, the posterior mean is:$\frac{z+a}{N+a+b}$, and the mode is $\frac{z+a-1}{N+a+b-2}$.

c) For exercise, make sure the scripts named `DBDA2E-utilities.R` and `hw2_mcmc_generation_jags.R` are loaded in your Rstudio environment. Add comments to each line of the following code, evaluate the results, and confirm that the conclusion from part `b)` is consistent in our numerical example (hint: look at the attributes from `chain_summary`).

Please note that, in the case the value p dictates how many heads we should be getting in our simulated data (observed trials), which is used to generate the likelihood function, p is not to be confused with z which is the number of heads we get in our observed trials.

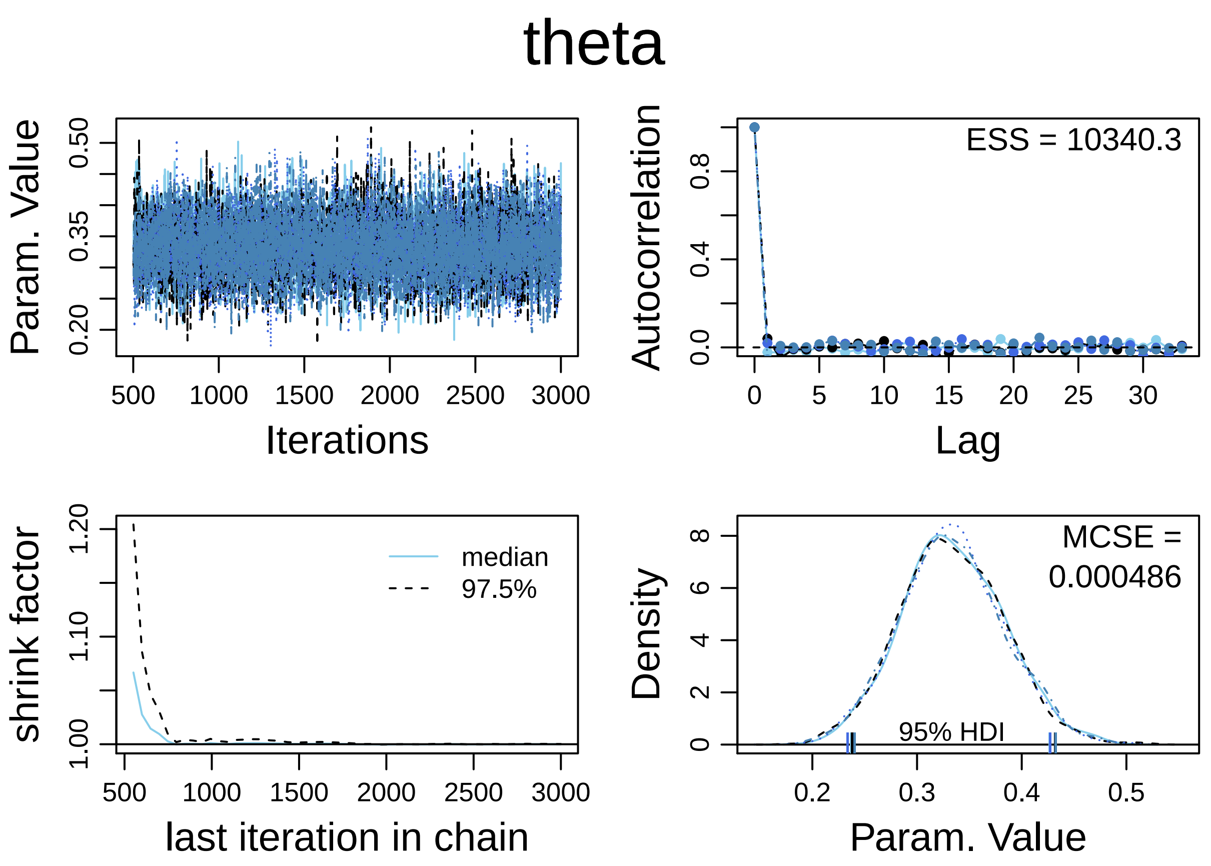
```r
# Load the functions genMCMC, smryMCMC, and plotMCMC:

source("hw2_mcmc_generation_jags.R")
```

```
##
## *********************************************************************
## Kruschke, J. K. (2015). Doing Bayesian Data Analysis, Second Edition:
## A Tutorial with R, JAGS, and Stan. Academic Press / Elsevier.
## *********************************************************************
```

```r
## initial example

# simulation parameter for Bernoulli random variables
# here we choose p = 0.1 as our simulated bias
p <-0.1

# set n = 10
n <- 10

# simulate data:
myData <-  rbinom(n,1,p)

# number of heads from the trials
z <- sum(myData)

# generate MCMC chain of theta with a beta distribution as prior
# the parameters for beta are a =30 and b = 50 respetively
mcmcCoda = genMCMC( data=myData , numSavedSteps=10000, a = 30, b =50 )
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 10
##    Unobserved stochastic nodes: 1
##    Total graph size: 14
##
## Initializing model
##
## Burning in the MCMC chain...
## Sampling final MCMC chain...
```

```r
# Display diagnostics of the Markov chain in the MCMC simulation,
# for specified parameter theta:
diagMCMC( mcmcCoda , parName="theta" )
```



```
# Display numerical summary statistics of chain:
chain_summary <-  smryMCMC( mcmcCoda )
```

```
##             Mean    Median      Mode   ESS HDImass    HDIlow   HDIhigh CompVal
## theta 0.3339711 0.3321419 0.3232078 10000     0.95 0.2411789 0.4345415      NA
##       PcntGtCompVal ROPElow ROPEhigh PcntLtROPE PcntInROPE PcntGtROPE
## theta            NA      NA       NA         NA         NA         NA
```

d) Repeat part c), but with a set of different values of n ranging from 2 to 1000, you can pick 5 or 6 of them spaced across the interval. Collect the summary value for each case of n. Demonstrate the effect on the posterior mean and mode in each case via a summary table or a plot.

Answer. Using the code below, we create a for-loop to collect summary values of the MCMC simulation at each size of n. From this experiment, we can see that the number of datapoints in the likelihood function relates directly to how close the mean of the posterior distribution is to what is observed in the data, compared to what is assumed in the prior distribution.

```
myData <-list()
mcmcCoda <- list()
chain_diagnostic <-list()
chain_summary <-list()
i <-1

set.seed(2021)

# set different data size for the likelihood function
data_size <- c(2,10,100,1000,5000)

for (n in data_size){
  print(paste('n = ',n))

  myData[[i]] <-  rbinom(n,1,p)

  mcmcCoda[[i]] = genMCMC( data=myData[[i]] , numSavedSteps=10000, a = 3, b =5 )

  # Display diagnostics of chain, for specified parameter:
  #diagMCMC( mcmcCoda[[i]] , parName="theta" )

  # Display numerical summary statistics of chain:
  chain_summary0 <- as.data.frame(smryMCMC( mcmcCoda[[i]] ))

  chain_summary0[,'likelihood.n']<-data_size[i]
  chain_summary<-  rbind(chain_summary,chain_summary0)
  i <- i+1
  print(paste('i = ', i))


}


# collect result summary

test_summary<-data.frame()
for (i in (1:length(data_size))){
  print(paste('Data size for likelihood = ', data_size[i]))
  test_summary0 <-as.data.frame(t(summary(mcmcCoda[[i]])$statistics))
  test_summary0[,'likelihood.size']<-data_size[i]
  test_summary<-rbind(test_summary,test_summary0)

}
```

```
knitr::kable(test_summary)
```

Mean	SD	Naive SE	Time-series SE	likelihood.size
0.3000078	0.1378155	0.0013782	0.0013781	2
0.2227699	0.0964215	0.0009642	0.0009280	10
0.1113302	0.0299262	0.0002993	0.0003039	100
0.1100901	0.0098882	0.0000989	0.0001001	1000
0.1108362	0.0044096	0.0000441	0.0000450	5000