U20 Math 559 Bayesian Statistics Homework 2

Due: 3/27/2021

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:

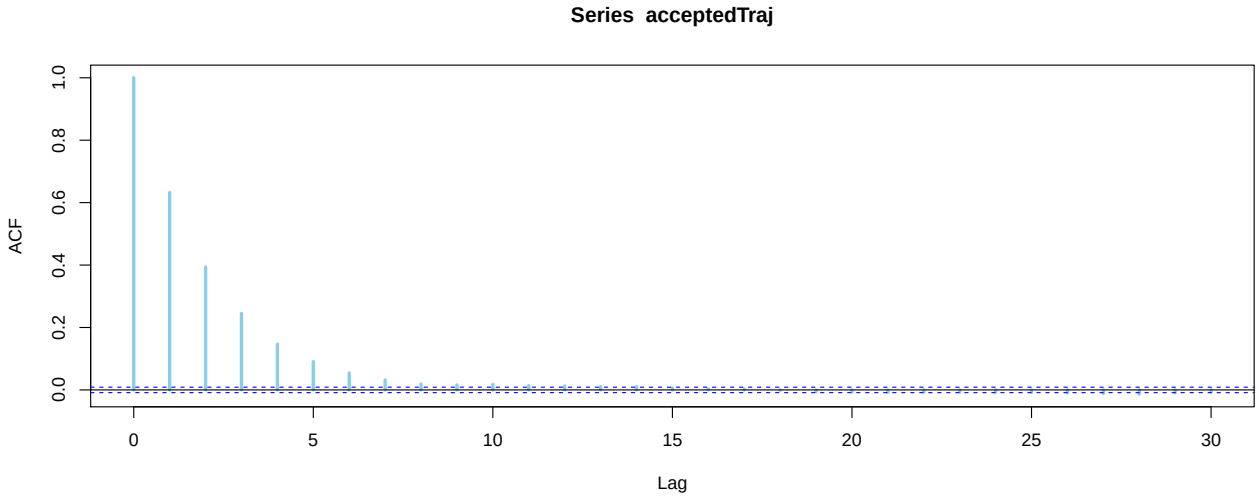
```
# Now generate the random walk. The 't' index is time or trial in the walk.
# Specify seed to reproduce same random walk:
set.seed(47405)
# Specify standard deviation of proposal distribution:
proposalSD = c(0.02,0.2,2.0)[2]
```

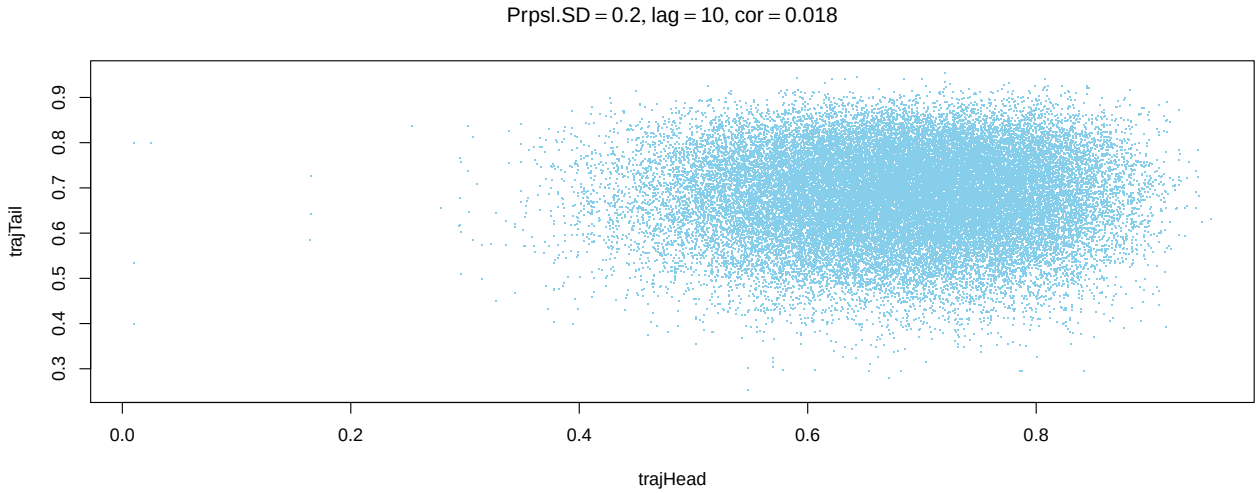
```
for ( t in 1:(trajLength-1) ) {
  currentPosition = trajectory[t]
  # Use the proposal distribution to generate a proposed jump.
  proposedJump = rnorm( 1 , mean=0 , sd=proposalSD )
  # Compute the probability of accepting the proposed jump.
  probAccept = min( 1,
                    targetRelProb(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

```
# 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}
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)))) )
```





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.

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

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?

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.

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?

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?

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?

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?

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.

```
# Load the functions genMCMC, smryMCMC, and plotMCMC:
source("hw2_mcmc_generation_jags.R")

## initial example


p <-0.1

n <- 10

myData <-  rbinom(n,1,p)

z <- sum(myData)

mcmcCoda = genMCMC( data=myData , numSavedSteps=10000, a = 30, b =50 )

# 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 )
```

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.