STATS 451 Homework 5

Yuzhou Peng

2024-03-20

Problem 1

$$\text{The prior distribution is given by: } (\theta_1, \theta_2, \theta_3) \sim Dirichlet(1, 1, 1)$$
$$\mathbb{P}(\theta_1, \theta_2, \theta_3) = 2$$
$$\text{The likelihood is given by: } \mathbb{P}(y_1 \ldots y_n | \theta_1, \theta_2, \theta_3) = C_1 \theta_1^{n_1} \theta_2^{n_2} \theta_3^{n_3}$$
$$\text{where } C_1 \text{is the normalizing term and } n_i \text{ is the counts of people that vote for candidate i (i = 1,2,3) and } n_1 = 727, n_2 = 583, n_3 = 13$$
$$\text{The posterior is given by: } \mathbb{P}(\theta_1, \theta_2, \theta_3 | y_1 \ldots y_n) = C_1 \theta_1^{728-1} \theta_2^{584-1} \theta_3^{138-1}$$
$$\theta_1, \theta_2, \theta_3 | y_1 \ldots y_n \sim Dirichlet(728, 584, 138)$$

Problem 2: Metropolis Algorithm to sample from the posterior distribution

```
prior = function( theta )  {
  # flat prior, independent for theta1 and theta2
  # theta = (theta1, theta2)
  pTheta = 2

  # constraints
  pTheta[ theta[1] >= 1 | theta[1] <= 0 ] = 0
  pTheta[ theta[2] >= 1 | theta[2] <= 0 ] = 0
  pTheta[ 1 - theta[1]- theta[2] >= 1 | 1 - theta[1]- theta[2] <=0 ] = 0
  return( pTheta )
}

log_prior <- function( theta )  {

  log_pTheta = log(2)

  return( log_pTheta )
}

likelihood = function( theta, data )  {
  #data = (n1,n2,n3)
  #theta = (theta1, theta2)
  z <- sum(data)
  n1 <- data[1]
  n2 <- data[2]

  pDataGivenTheta = theta[1]^n1 * theta[2]^n2 * (1 - theta[2] -theta[1])^(z-n1-n2)

  # constraints so that the biases of each coin is in the interval (0, 1)
  pDataGivenTheta[ theta[1] > 1 | theta[1] < 0 ] = 0
  pDataGivenTheta[ theta[2] > 1 | theta[2] < 0 ] = 0
  pDataGivenTheta[ 1 - theta[1]- theta[2] > 1 | 1 - theta[1]- theta[2] < 0 ] = 0
  return( pDataGivenTheta )
 }

log_likelihood <- function( theta, data )  {
  #data = (n1,n2,n3)
  #theta = (theta1, theta2)
  z <- sum(data)
  n1 <- data[1]
  n2 <- data[2]

  if (theta[1] >= 1 | theta[1] <= 0 ){
    log_pDataGivenTheta = -10^(8)
  } else if (theta[2] > 1 | theta[2] < 0 ){
    log_pDataGivenTheta = -10^(8)
  } else if ( 1 - theta[1] -theta[2] >= 1 | 1 - theta[1] - theta[2] <= 0){
    log_pDataGivenTheta = -10^(8)
  } else {
    log_pDataGivenTheta = n1*log(theta[1]) + n2*log(theta[2]) + (z-n1-n2) * log(1- theta[1] - theta[2])
  }

  return( log_pDataGivenTheta )
 }

log_rel_posterior <- function(theta, data){
  return(log_prior(theta) + log_likelihood(theta, data))
}

rel_target_prob <- function(theta, data){
  return(log_rel_posterior(theta, data))
}

log_likelihood(theta = c(0.6, 0.3), data =c(727, 583, 137))
```

```
## [1] -1388.741
```

```
log_rel_posterior(theta = c(0.6, 0.3), data =c(727, 583, 137))
```

```
## [1] -1388.047
```

```
log_likelihood(theta = c(0.806, 0.248), data =c(727, 583, 137))
```

```
## [1] -1e+08
```

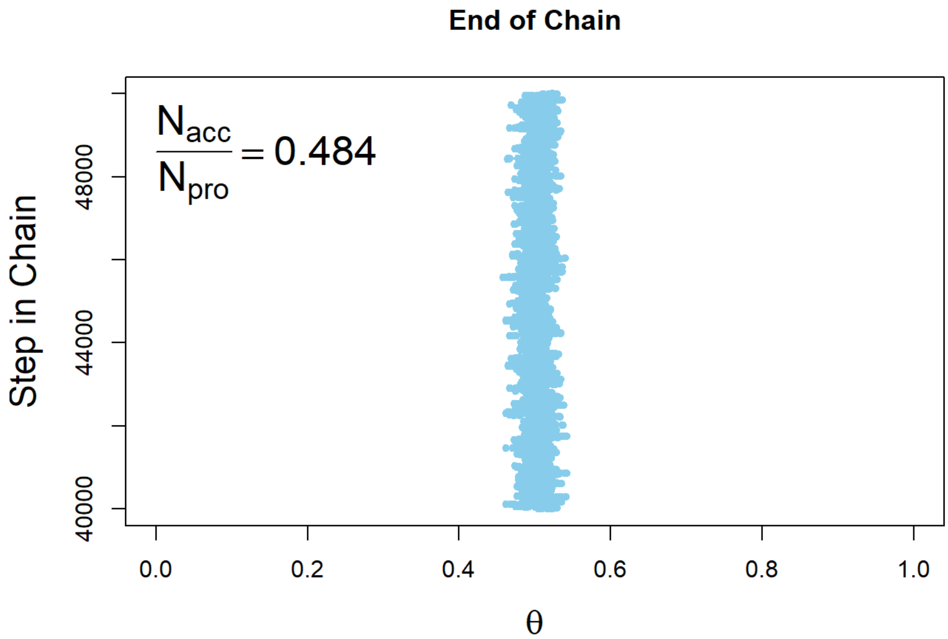
```
log_rel_posterior(theta = c(0.806, 0.248), data =c(727, 583, 137))
```

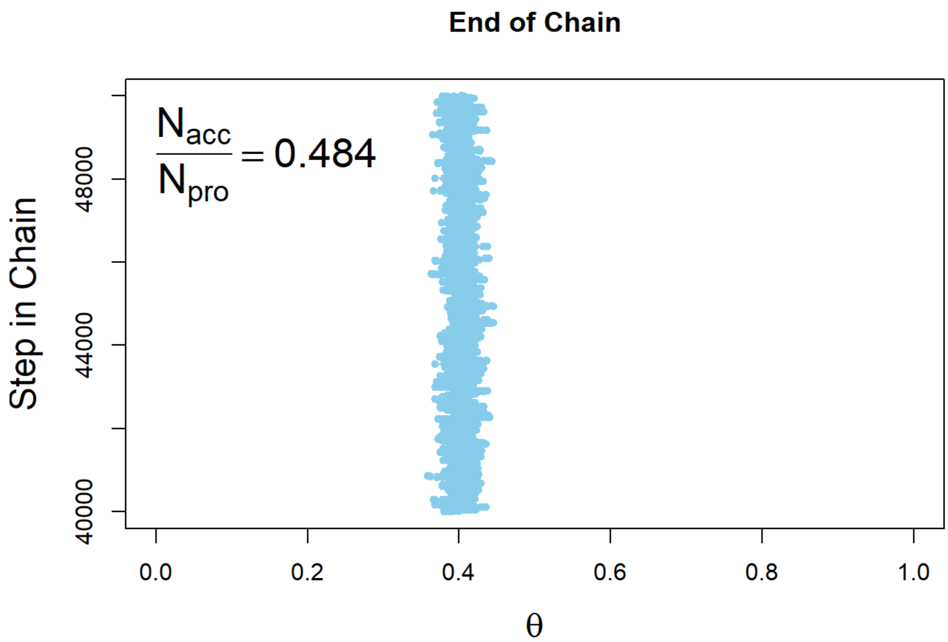
```
## [1] -1e+08
```

```
my_data <- c(727, 583, 137)

# 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/samples of theta1 and theta2:
trajectory = array( 0 , c(2, trajLength))
# Specify where to start the trajectory:
trajectory[,1] = c(0.1, 0.1)
# 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
```

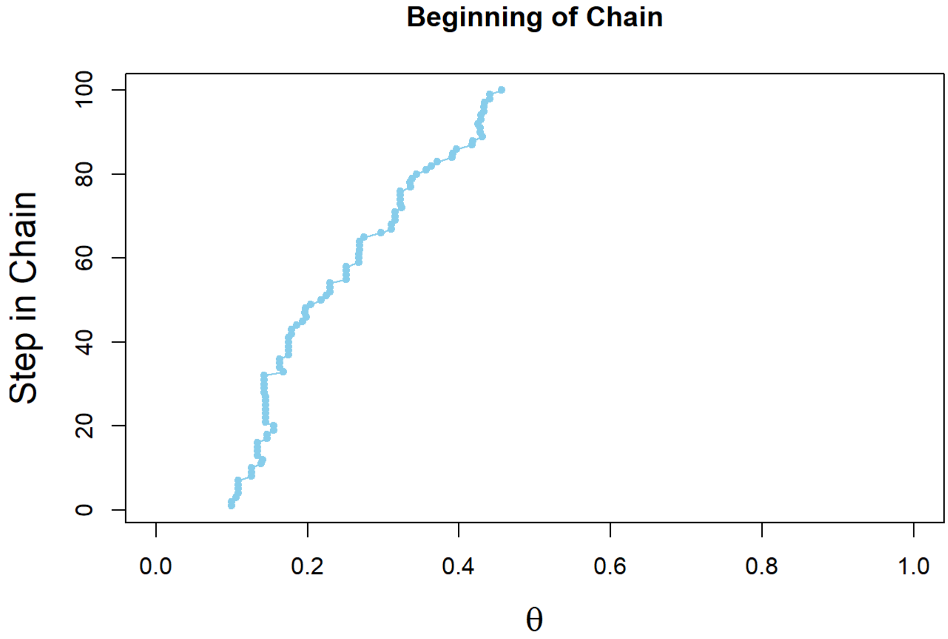
```
set.seed(451)

# 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(0.01, 0.01)

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_target_prob(proposedPosition, my_data) - rel_target_prob(currentPosition, my_data))
  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.
```

```
#------------------------------------------------------------------------
# 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
for(k in 1:2) {
  plot( trajectory[k,idxToPlot] , idxToPlot , main="End of Chain" ,
        xlab=bquote(theta) , xlim=c(0,1) , 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 ))))
}
```

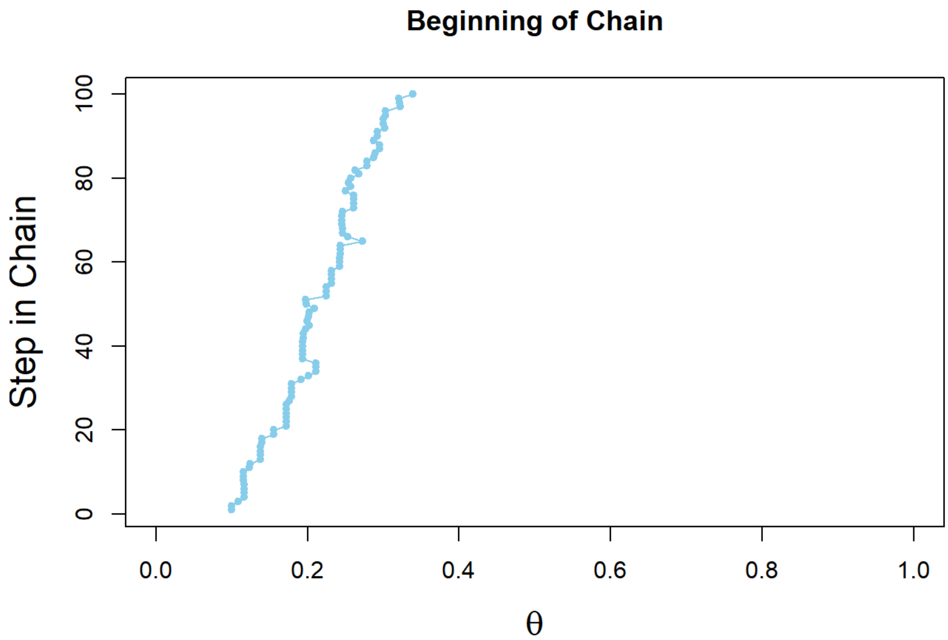




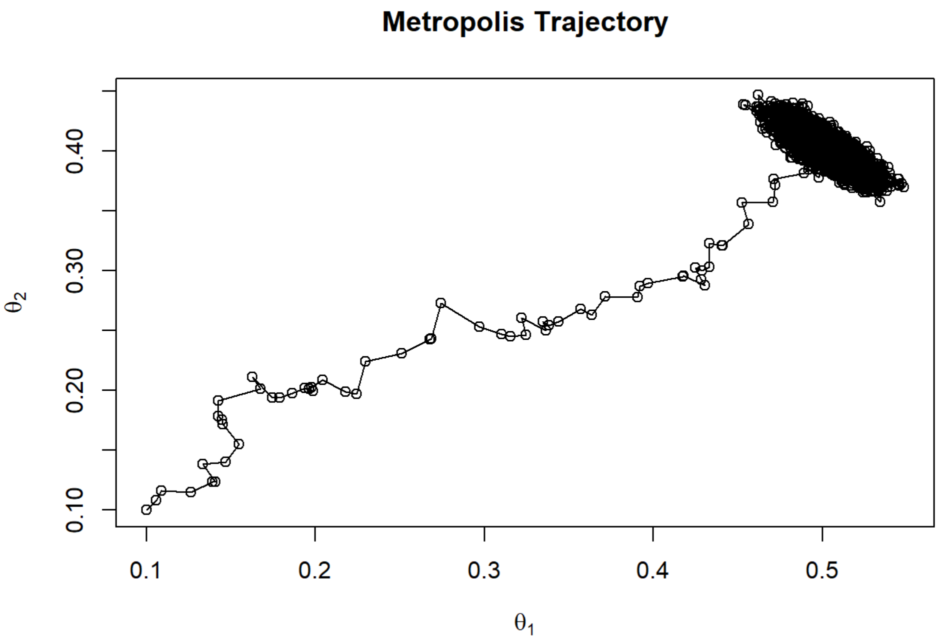
```
# Trajectory, a.k.a. trace plot, beginning of chain:
idxToPlot = 1:100
for(k in 1:2)
plot( trajectory[k, idxToPlot] , idxToPlot , main="Beginning of Chain" ,
      xlab=bquote(theta) , xlim=c(0,1) , ylab="Step in Chain" ,
      type="o" , pch=20 , col="skyblue" , cex.lab=1.5 )
```



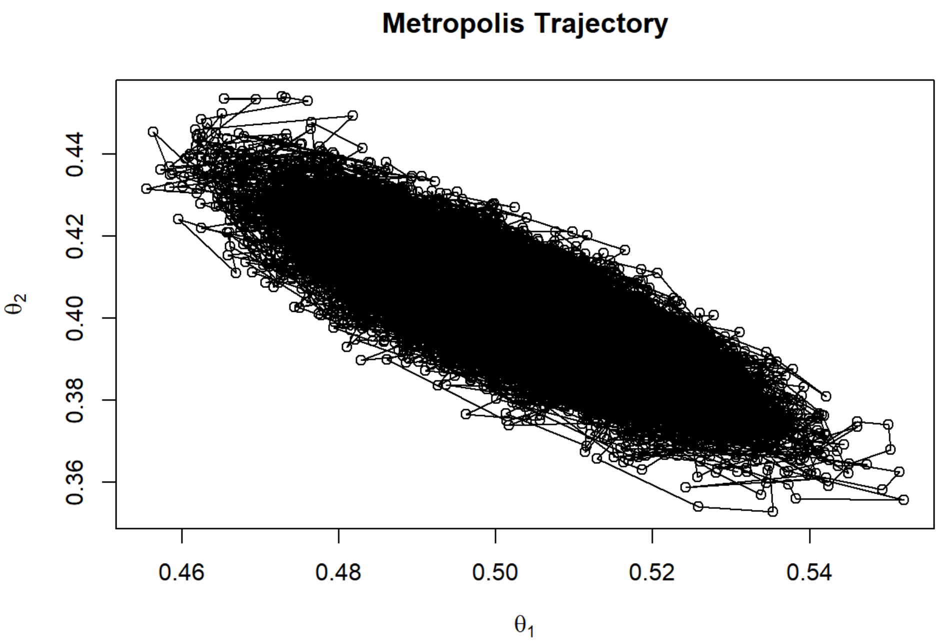

```
# Indicate burn in limit (might not be visible if not in range):
if ( burnIn > 0 )  {
  abline(h=burnIn,lty="dotted")
  text( 0.5 , burnIn+1 , "Burn In" , adj=c(0.5,1.1) )
 }
```




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

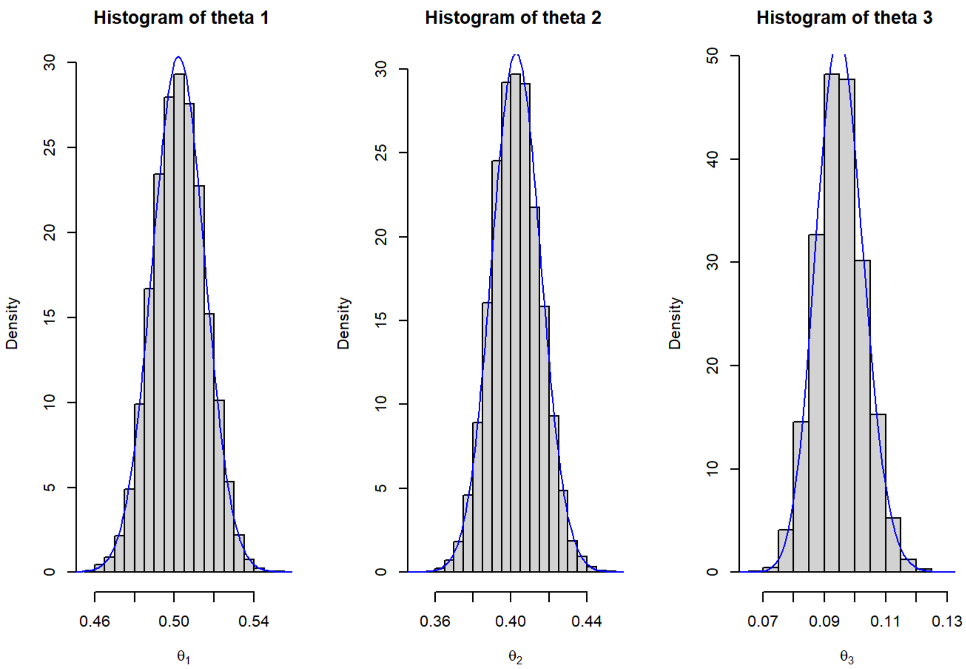


The visualization result shows that the sample from Metropolis algorithm displays a stationary distribution and mixes well.
We can consider it as valid sample from the posterior distribution.

Problem 3: Comparison with true posterior distribution

The theoretical distribution corresponding to sample is the marginal distribution of the true posterior.
Moreover, for Dirichlet distribution, the marginal distributions are beta distribution.
In particular, $\theta_1|y_1 \dots y_n \sim Beta(728, 584+138)$, $\theta_2|y_1 \dots y_n \sim Beta(584, 728+138)$ and $\theta_3|y_1 \dots y_n \sim Beta(138, 584+728)$

```
theta1_sample <- acceptedTraj[1,]
theta2_sample <- acceptedTraj[2,]


par(mfrow = c(1, 3))

beta_step <- seq(0.01, 1, by = 0.001)

hist(theta1_sample, freq = F, main = "Histogram of theta 1 ",
     xlab = expression(theta[1]))
lines(beta_step, dbeta(beta_step, shape1 = 728, shape2 = 584 + 138), col = "blue")
hist(theta2_sample, freq = F, main = "Histogram of theta 2 ",
     xlab = expression(theta[2]))
lines(beta_step, dbeta(beta_step, shape1 = 584, shape2 = 728 + 138), col = "blue")
hist(1-theta1_sample-theta2_sample, freq = F, main = "Histogram of theta 3 ",
     xlab = expression(theta[3]))
lines(beta_step, dbeta(beta_step, shape1 = 138, shape2 = 728 + 584), col = "blue")
```



Compare the sample histograms and theoretical curves, we see they are close to each other.

Problem 4: Inference

4.1

```
sample_difference <- theta1_sample - theta2_sample
mean(sample_difference > 0)
```

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

The estimated probability of the event “George Bush wins” is 1.

4.2

```
mean(sample_difference > 0.1)
```

```
## [1] 0.499375
```

The estimated probability of the event “George Bush wins with a marginal of 0.1” is 0.499.

4.3

```
mean(sample_difference > 0.2)
```

```
## [1] 0
```

The estimated probability of the event “George Bush wins with a marginal of 0.2” is 0.