Assignment 12.a for **STATGR6103**

submitted to Professor Andrew Gelman

Advait Rajagopal

2 December 2016

1 Question 1

Consider a simple one-parameter model of independent data, $y_i \sim \text{Cauchy}(\theta, 1), i = 1, ..., n$, with uniform prior density on θ and two data points, $y_1 = 1.3$, $y_2 = 15.0$.

1.1 Part A

Graph the posterior density.

We have the following information regarding the prior and joint likelihood respectively;

$$p(\theta) \propto 1$$
$$p(y|\theta) \propto \prod_{i=1}^{n} \frac{1}{1 + (y_i - \theta)^2}$$

Thus the posterior density is;

$$\begin{aligned} p(\theta|y) &\propto p(\theta)p(y|\theta) \\ &= \prod_{i=1}^{n} \frac{1}{1 + (y_i - \theta)^2} \end{aligned}$$

Figure 1 shows a graph of the posterior density;

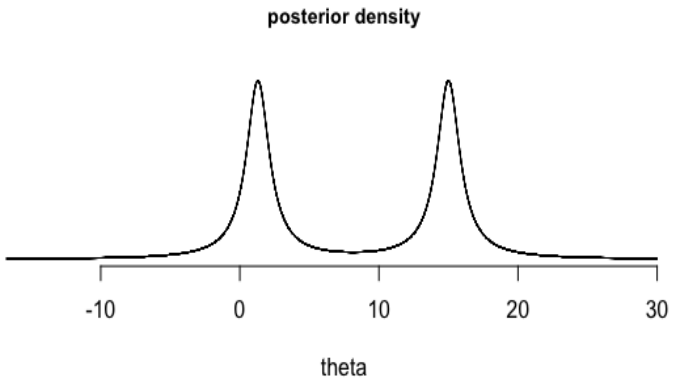

Figure 1: *The posterior density is bimodal, peaking at the two data points 1.5 and 15.0*

1.2 Part B

Program the Metropolis algorithm for this problem using a symmetric Cauchy jumping distribution. Tune the scale parameter of the jumping distribution appropriately.
The scale parameter for random walk Metropolis algorithms should be tuned to have an acceptance rate of 0.234 [1]. I program a Metropolis algorithm[2] that had an acceptance probability of approximately 0.25. The symmetric Cauchy proposal distribution is as follows;

$$proposal(\hat{\theta}) \sim \text{Cauchy}(\hat{\theta}, 10)$$

I test it with 10000 iterations and 3 chains. The values of θ drawn from the posterior density as estimated by the Metropolis algorithm are displayed along with the original posterior in Figure 2.

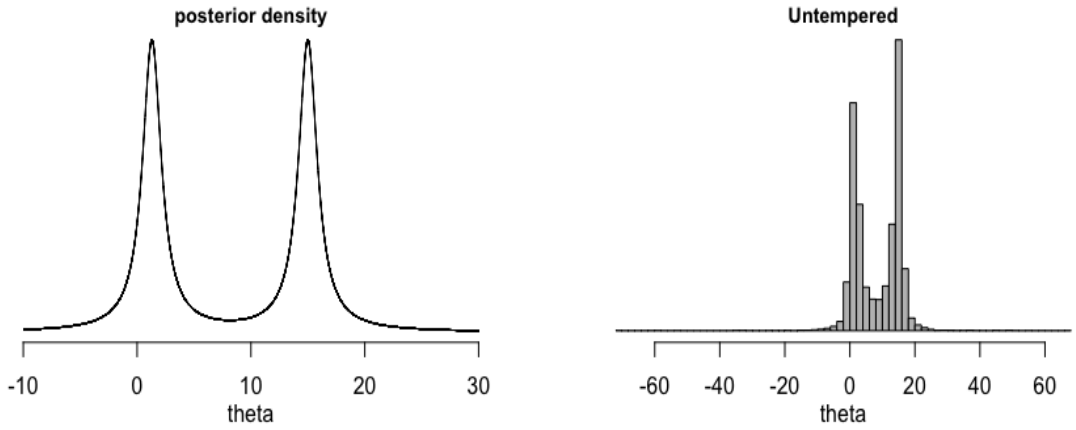

Figure 2: *Figure 2(a) shows the original or true posterior density of θ and Figure 2(b) shows the same as estimated by a standard Metropolis algorithm with a Cauchy jumping distribution.*

We see that the Metropolis algorithm captures the bimodal nature of the distribution but overestimates the density at one of the modes.

1.3 Part C

Program simulated tempering with a ladder of 10 inverse-temperatures, 0.1, . . . , 1.
I incorporate the ladder of inverse-temperatures as the scale parameter of the jumping distribution by sampling a different temperature value from the 10 values between 0.1 and 1 (inclusive). This is achieved by the simulated tempering. This allows the Metropolis algorithm to sample from both modes equally and gives a better estimate of the posterior density.

1.4 Part D

Compare your answers in (b) and (c) to the graph in (a).
Figure 3 shows the original unnormalized posterior density, the untempered estimate and the estimate

[1] G.O Roberts, A. Gelman and W.R Gilks, "Weak Convergence and Optimal Scaling of Random Walk Metropolis Algorithms", *The Annals of Applied Probability*, Vol. 7, No. 1, 110 120, (1997),

[2] See code attached

of the posterior using simulated tempering.

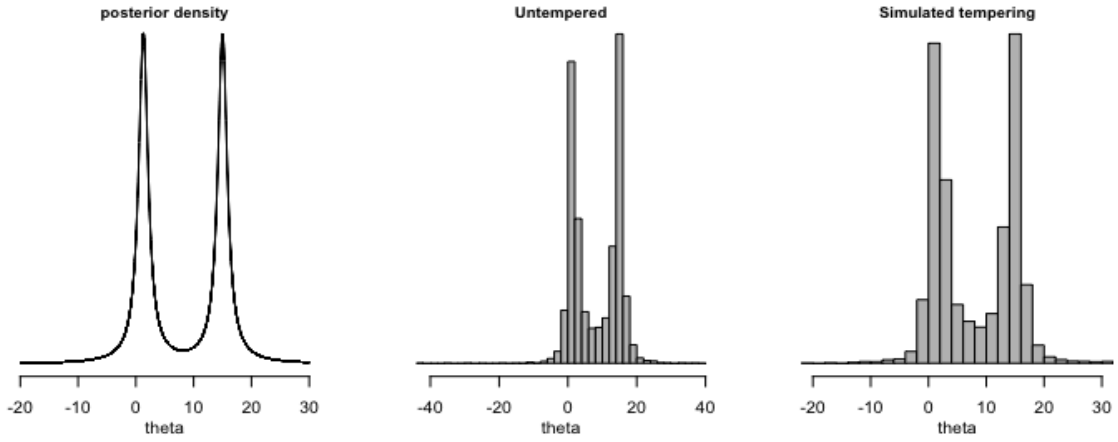

Figure 3: *Figure 3(a) shows the original or true posterior density of θ and Figure 3(b) shows samples of θ from the untempered estimate of the posterior density, Figure 3(c) shows the samples of θ from the posterior density that is estimated using* ***simulated tempering****.*

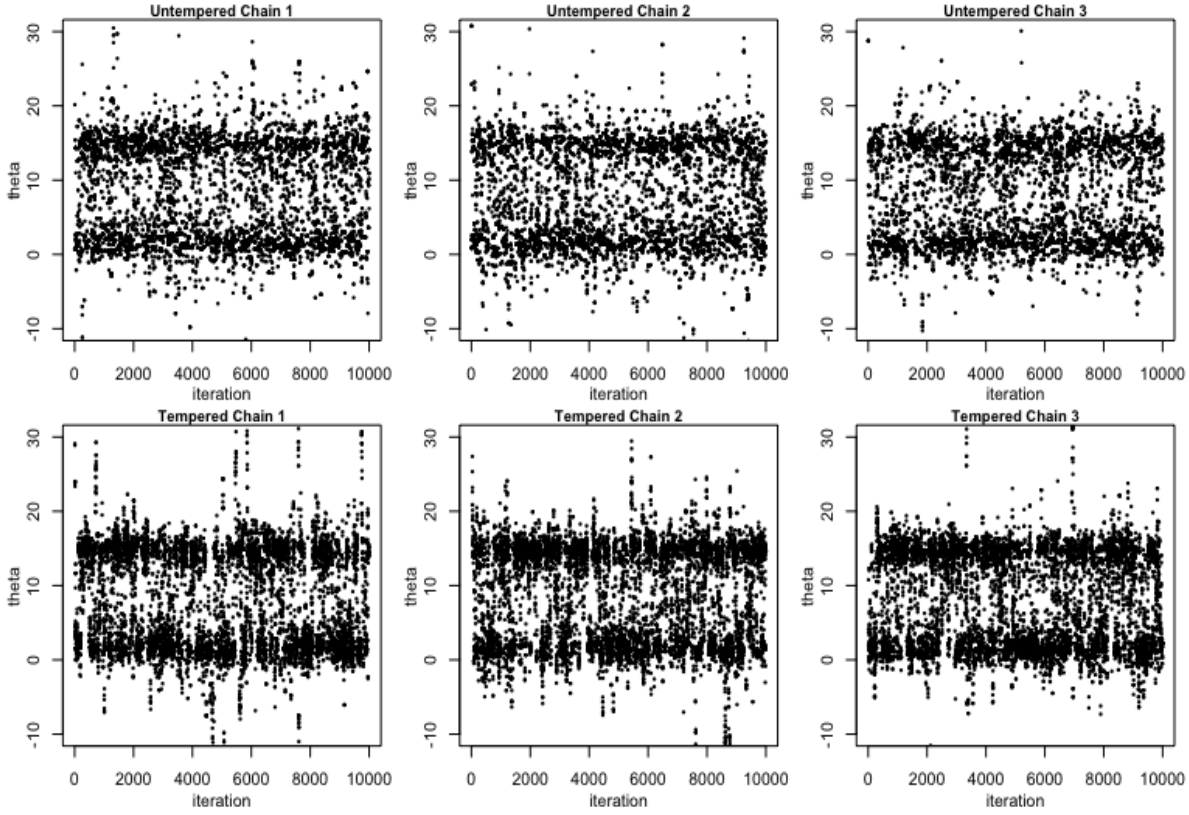

Figure 4: *Figure 3(a) shows the original or true posterior density of θ and Figure 3(b) shows samples of θ from the untempered estimate of the posterior density, Figure 3(c) shows the samples of θ from the posterior density that is estimated using* ***simulated tempering****.*

Figure 4 shows the convergence of the 3 chains and compares the untempered Metropolis algorithm with a single scale parameter to the 3 chains of the Metropolis algorithm that uses the ladder of scale parameters. It is clear from the figure that the algorithm with simulated tempering not only converges around the modes better but is able to ascribe more of the posterior mass to the two modes. The untempered Metropolis algorithm is able to capture the two modes, but the points are a lot more dispersed indicating the difficulty it had in converging to the true modes.

2 Code

2.1 R Code

Listing 1: r

```
m(list = ls())
setwd("/Users/Advait/Desktop/New␣School/Fall16/BDA/Class22")
library(rstan)
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())
##################################################
#Part A
post <- function(theta){
  ((1 + (1.3 - theta)^2)^-1) + ((1 + (15 - theta)^2)^-1)
}
arg <- seq(-20,30,0.001)
post(arg)

plot(arg, post(arg),type = "l", xlim = c(-15,30),
     yaxt = "n",
     ylab = NA,
     bty = "n",
     xlab = "theta",
     main = "posterior␣density",
     cex.main = 0.9)
##################################################
#Part B
#log posterior function
y <- c(1.3,15)
scl <- 10
#
log_post <- function(theta){
  lpost = dcauchy(y,
                  theta,
                  1, log = T)
  return(sum(lpost))
}

#proposal function
proposal <- function(theta){
  initial = rcauchy(1,theta, scl)
  return(initial)
}
```

Listing 2: R code contd.

```
#Metropolis algorithm
metropolis <- function(startvalue, iterations, n_chains){
  chains <- list()
  for (j in 1:n_chains){
   chain = NULL
   chain = array(dim = c(iterations+1,length(startvalue)))
   chain[1,] = startvalue
 for(i in 1:iterations){
   prop = proposal(chain[i,])
   probab = exp(log_post(prop) - log_post(chain[i,]))
   if (runif(1) < probab){
     chain[i+1,] = prop
   }else{
     chain[i+1,] = chain[i,]
   }
   chains[[j]] <- chain
   }
  }
 return(chains)
}

fit <- metropolis(50,10000, 3)
warmup <- 5000
acceptance.rate <- NULL
for (i in 1:3){
  acceptance.rate[i] <- 1 - mean(duplicated(fit[[i]][-(1:warmup),]))
}
acceptance.rate
par (mfrow = c ( 1 , 2) ,
     mar = c ( 3 , 3 , 1 , 1) ,
     oma = c ( 0.5 , 0.5 , 0.5 , 0.5 ) ,
     mgp = c ( 2 , 1 , 0 ) )
plot(arg, post(arg),type = "l",
     main = "posterior density",
     cex.main = 0.9,
     xlab = "theta",
     bty = "n",
     yaxt = "n",
     ylab = NA)
```

Listing 3: R code contd.

```
hist(c(fit[[1]][-(1:warmup),], fit[[2]][-(1:warmup),],
          fit[[3]][-(1:warmup),]), breaks = 50, freq = F,
     col = "gray",
     main = "Untempered",
     xlab = "theta",
     cex.main = 0.9, ylab = NA, yaxt = "n")
################################################
#Part C
#run metropolis for simulated tempering
metropolis_temp <- function(startvalue, iterations, n_chains){
  chains <- list()
  for (j in 1:n_chains){
    chain = NULL
    chain = array(dim = c(iterations+1,length(startvalue)))
    chain[1,] = startvalue
    temp  = NULL
    for(i in 1:iterations){
      temp[i] <- 1/sample(seq(0.1,1,0.1),1, replace = T)
      proposal1 <- function(theta) {
      rcauchy (1,theta ,
                scale = temp[i] )
      }
      prop = proposal1(chain[i,])
      probab = exp(log_post(prop) - log_post(chain[i,]))
      if (runif(1) < probab){
        chain[i+1,] = prop
      }else{
        chain[i+1,] = chain[i,]
      }
      chains[[j]] <- chain
    }
  }
  return(chains)
}
fit_2 <- metropolis_temp(50,10000, 3)
warmup <- 5000
```

Listing 4: R code contd.

```
acceptance.rate <- NULL
for (i in 1:3){
  acceptance.rate[i] <- 1 - mean(duplicated(fit_2[[i]][-(1:warmup),]))
}
acceptance.rate
par (mfrow = c ( 1 , 3) ,
     mar = c ( 3 , 3 , 1 , 1) ,
     oma = c ( 0.5 , 0.5 , 0.5 , 0.5 ) ,
     mgp = c ( 2 , 1 , 0 ) )
plot(arg, post(arg),type = "l",
     main = "posterior␣density",
     cex.main = 0.9,
     xlab = "theta",
     bty = "n",
     yaxt = "n",
     ylab = NA)
hist(c(fit[[1]][-(1:warmup),], fit[[2]][-(1:warmup),],
       fit[[3]][-(1:warmup),]), breaks = 50, freq = F,
     col = "gray",
     main = "Untempered",
     xlab = "theta",
     cex.main = 0.9, ylab = NA, yaxt = "n")

hist(c(fit_2[[1]][-(1:warmup),],
       fit_2[[2]][-(1:warmup),],
       fit_2[[3]][-(1:warmup),]),
     xlab = "theta",
     ylab = "␣",
     yaxt = 'n',
     xlim = c(-20, 30),
     main = "Simulated␣tempering",
     breaks = 50,
     col = "grey",
     freq = FALSE,
     cex.main = 0.9)
########
##Checking convergence
par (mfrow = c ( 2 , 3) ,
     mar = c ( 3 , 3 , 1 , 1) ,
     oma = c ( 0.5 , 0.5 , 0.5 , 0.5 ) ,
     mgp = c ( 2 , 1 , 0 ) )
```

Listing 5: R code contd.

```
for(i in 1:3){
  plot(fit[[i]],
       cex = 0.5,
       pch = 16,
       main = paste("Untempered Chain", i),
       ylab = "theta",
       xlab = "iteration",
       ylim = c(-10,30),
       cex.main = 0.9)
       }
for(i in 1:3){
  plot(fit_2[[i]],
       cex = 0.5,
       pch = 16,
       main = paste("Tempered Chain", i),
       ylab = "theta",
       xlab = "iteration",
       ylim = c(-10,30),
       cex.main = 0.9)
}
```