Problem 1

Fit the following model to the NBA free throw data:

Player	Overall proportion	Clutch makes	Clutch attempts
Russell Westbrook	0.845	64	75
James Harden	0.847	72	95
Kawhi Leonard	0.880	55	63
LeBron James	0.674	27	39
Isaiah Thomas	0.909	75	83
Stephen Curry	0.898	24	26
Giannis Antetokounmpo	0.770	28	41
John Wall	0.801	66	82
Anthony Davis	0.802	40	54
Kevin Durant	0.875	13	16

$Y_i|\theta_i \sim Binomial(n_i;\theta_i)$ and $\theta_i|m \sim Beta[exp(m)q_i, exp(m)(1-q_i)]$, where Y_i is the number of made clutch shots for player $i = 1, ..., 10$, n_i is the number of attempted clutch shots, $q_i \in (0,1)$ is the overall proportion, and $m \sim Normal(0, 10)$.

a) Why this is a reasonable prior for θ_i.

Since the domain of the beta distribution covers 0 to 1 and the mean of the beta distribution is

$$\frac{e^m q_i}{e^m q_i + e^m(1-q_i)} = q_i$$

so distribution is centered at q_i. and takes values on [0,1].

b) What is the role of m in the prior.

This determines the spread of the distribution around q_i.

c) Derive the full conditional posterior for θ_1.

$$\begin{aligned}
f(\theta_1|Y_1, ..., Y_{10}, \theta_2, ...\theta_{10}, m) &= \frac{f(Y_1, ..., Y_{10}, \theta_1, ..., \theta_{10}, m)}{f(Y_1, ..., Y_{10}, \theta_2, ..., \theta_{10}, m)} \\
&\propto f(Y_1, ..., Y_{10}, \theta_1, ..., \theta_{10}, m) \\
&\propto f(Y_1, ..., Y_{10}|\theta_1, ..., \theta_{10}, m) f(\theta_1, ..., \theta_{10}, m) \\
&\propto f(Y_1, ..., Y_{10}|\theta_1, ..., \theta_{10}, m) f(\theta_1, ..., \theta_{10}|m) f(m) \\
&\propto f(Y_1, ..., Y_{10}|\theta_1, ..., \theta_{10}) f(\theta_1, ..., \theta_{10}|m) \\
&\propto \prod_{i=1}^{10} f(Y_i|\theta_i) f(\theta_i|m) \\
&\propto f(Y_1|\theta_1) f(\theta_1|m) \\
&\propto Beta(Y_1 + exp(m)q_1, n_1 - Y_1 + exp(m)(1-q_1))
\end{aligned}$$

d) Write your own MCMC algorithm to compute a table of posterior means and 95% credible intervals for all 11 model parameters $(\theta_1, ..., \theta_{10}, m)$.

Similarly as previous part we obtain the full conditional posterior of each θ_i,

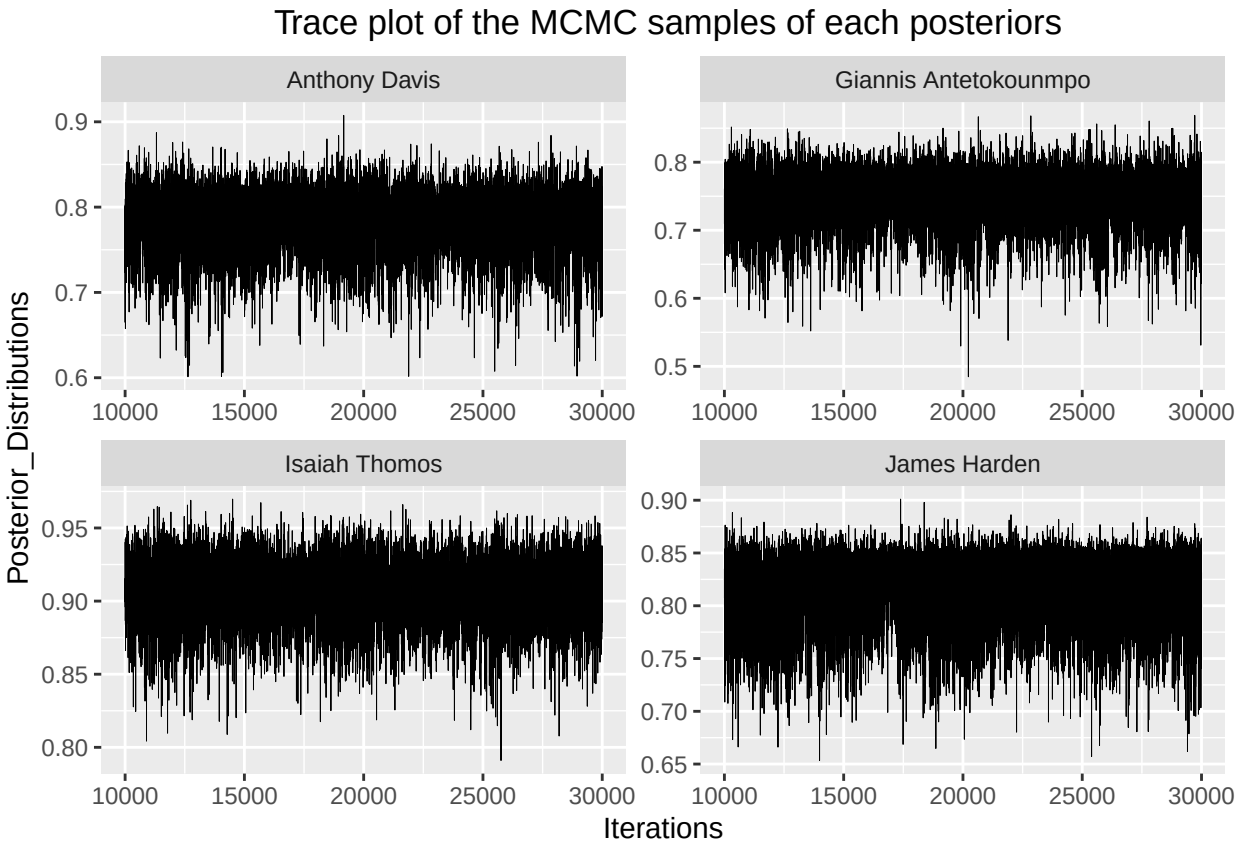
$$\theta_i|rest \propto Beta(Y_i + exp(m)q_i, n_i - Y_i + exp(m)(1-q_i)) \qquad i = 1, ..., 10.$$

However, we don't have nice form of the full conditional distribution for m,so we combine Gibbs and Metropolis algorithm.

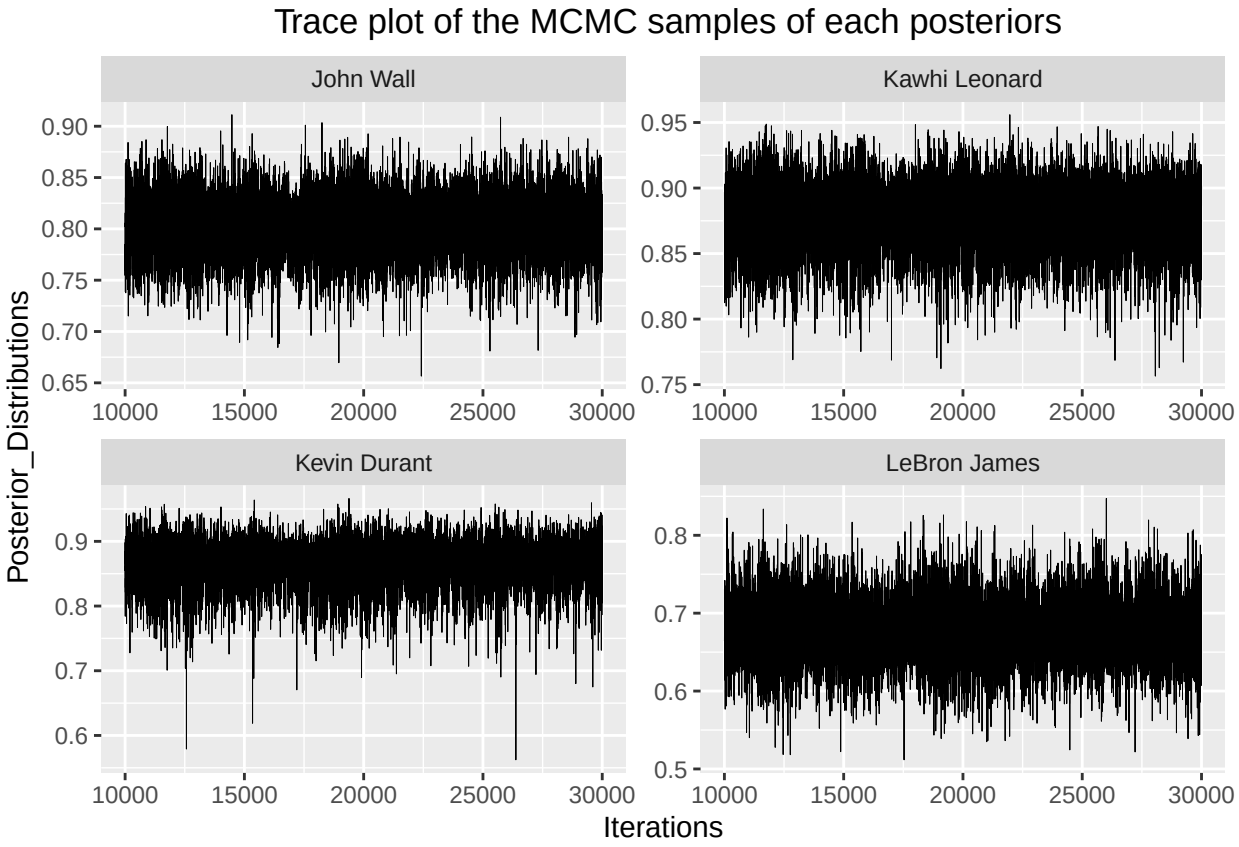
```
set.seed(100)
#given values in table
Y = c(64,72,55,27,75,24,28,66,40,13)
n = c(75,95,63,39,83,26,41,82,54,16)
q = c(0.845, 0.847, 0.880, 0.674, 0.909, 0.898, 0.770, 0.801, 0.802, 0.875)
N=10
#parameters to start MCMC
m = 0;
theta = q;
iters = 3*10^4;
burn= 10^4;
MCMC=matrix(0,iters-burn,N+1); # save only after burn-in
can_sd = 1
#log posterior for m
log_post_m = function(theta, Y, n, q, m){
  like = 0
  for(i in 1:10){
  like = like + dbeta(theta[i],exp(m)*q[i],exp(m)*(1-q[i]),log = TRUE )
  }
  prior = dnorm(m,0,sqrt(10),log=TRUE)
  return(like + prior)}

for(iter in 1:iters){
   # Gibbs for each theta
  for(i in 1:N){
    alpha = Y[i] + exp(m) * q[i]
    beta = n[i] - Y[i] + exp(m) * (1 - q[i])
    theta[i] = rbeta(1, alpha, beta)
  }
  # Metropolis for m
  can = rnorm(1, m, can_sd) #proposal distribution
  logR = log_post_m(theta, Y, n, q, can) - log_post_m(theta, Y, n, q, m)
  R=exp(logR)
  if(runif(1) < R){m = can}
  if(iter>burn){
  MCMC[iter - burn, ] = c(theta,m)}
}

acc_rate = mean(MCMC[2:(iters-burn),11]!=MCMC[1:(iters-burn-1),11])
MCMC=data.frame(MCMC)
colnames(MCMC)=c("Russell Westbrook","James Harden","Kawhi Leonard","LeBron James",
"Isaiah Thomos","Stephen Curry","Giannis Antetokounmpo","John Wall",
"Anthony Davis","Kevin Durant","m")
```

The trace plots of MCMC sample of parameters are presented below.

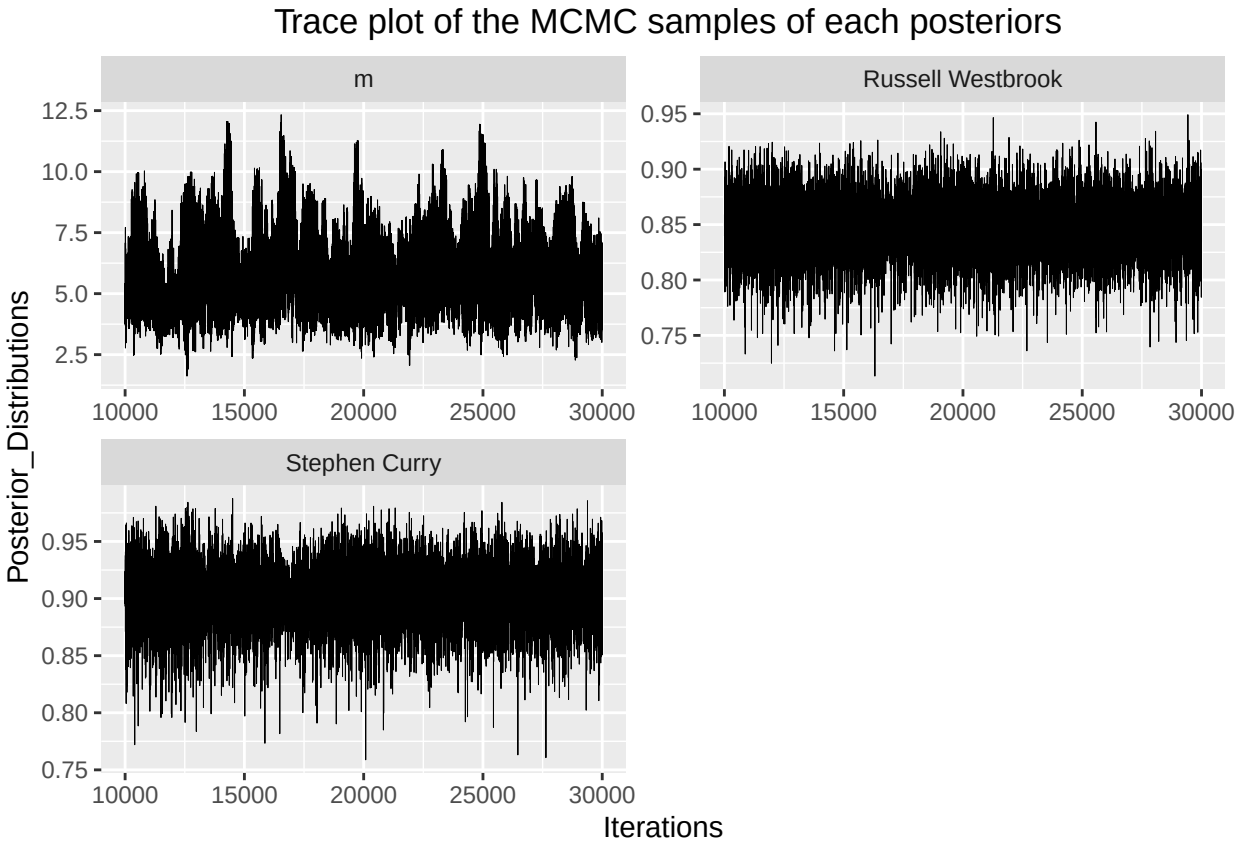
```
MCMC%>%
pivot_longer(cols = "Russell Westbrook":"m", names_to = "Parameters",
             values_to = "Posterior_Distributions")%>%
  ggplot(aes(x=rep(seq(burn+1,iters), 11), y = Posterior_Distributions))+
  xlab("Iterations")+
  geom_line(size=.1)+theme(plot.title = element_text(hjust = 0.5))+
  ggtitle(" Trace plot of the MCMC samples of each posteriors")+
  facet_wrap_paginate(~Parameters,scales = "free", ncol = 2, nrow = 2, page = 1)
```



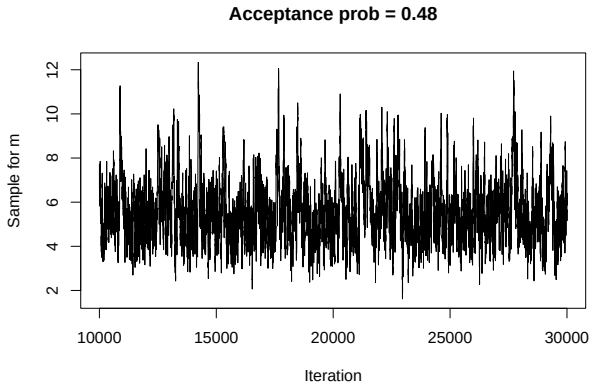

```
MCMC%>%
pivot_longer(cols = "Russell Westbrook":"m", names_to = "Parameters",
             values_to = "Posterior_Distributions")%>%
  ggplot(aes(x=rep(seq(burn+1,iters), 11), y = Posterior_Distributions))+
  xlab("Iterations")+
  geom_line(size=.1)+theme(plot.title = element_text(hjust = 0.5))+
  ggtitle(" Trace plot of the MCMC samples of each posteriors")+
  facet_wrap_paginate(~Parameters,scales = "free", ncol = 2, nrow = 2, page = 2)
```




```
MCMC%>%
pivot_longer(cols = "Russell Westbrook":"m", names_to = "Parameters",
             values_to = "Posterior_Distributions")%>%
  ggplot(aes(x=rep(seq(burn+1,iters), 11), y = Posterior_Distributions))+
  xlab("Iterations")+
  geom_line(size=.1)+theme(plot.title = element_text(hjust = 0.5))+
  ggtitle(" Trace plot of the MCMC samples of each posteriors")+
  facet_wrap_paginate(~Parameters,scales = "free", ncol = 2, nrow = 2, page = 3)
```




```
plot(seq(burn+1,iters),MCMC[ ,11],type="l",xlab="Iteration",ylab="Sample for m",
main=paste("Acceptance prob =",round(acc_rate,2)))
```




```
table=sapply(MCMC, quantile,  probs = c(.5, 0.025, 0.975))
rownames(table) = c("Means","2.5 %","97.5 %")
knitr::kable(t(table))
```

	Means	2.5 %	97.5 %
Russell Westbrook	0.8471779	0.8000826	0.8928168

	Means	2.5 %	97.5 %
James Harden	0.8247978	0.7447209	0.8582036
Kawhi Leonard	0.8794682	0.8305523	0.9171627
LeBron James	0.6761736	0.6101976	0.7474104
Isaiah Thomos	0.9087091	0.8672360	0.9403370
Stephen Curry	0.9008188	0.8544199	0.9473161
Giannis Antetokounmpo	0.7598172	0.6676155	0.8044549
John Wall	0.8023497	0.7518042	0.8514071
Anthony Davis	0.7932858	0.7154929	0.8340382
Kevin Durant	0.8724762	0.8009057	0.9160987
m	5.4168050	3.3610179	9.0036515

e) Fit the same model in JAGS. Turn in commented code, and comment on whether the two algorithms returned the same results.

```
#given data
Y = c(64,72,55,27,75,24,28,66,40,13)
n = c(75,95,63,39,83,26,41,82,54,16)
q= c(0.845, 0.847, 0.880, 0.674, 0.909, 0.898, 0.770, 0.801, 0.802, 0.875)
N = 10
# define string model
model_string = textConnection("model{
   # Likelihood
    for(i in 1:N){
      Y[i] ~ dbin(theta[i], n[i])
    }
   # Priors
    for(i in 1:N){
      theta[i] ~ dbeta(exp(m)*q[i], exp(m)*(1-q[i]))
    }

    m   ~  dnorm(0, 0.1)
 }")
# Load the data and compile the MCMC code
 inits = list(theta=q, m = 0)
 data  = list(Y = Y, n = n, q = q, N= N)
 model = jags.model(model_string,data = data, inits=inits, n.chains=2)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 10
##    Unobserved stochastic nodes: 11
##    Total graph size: 76
##
## Initializing model
```

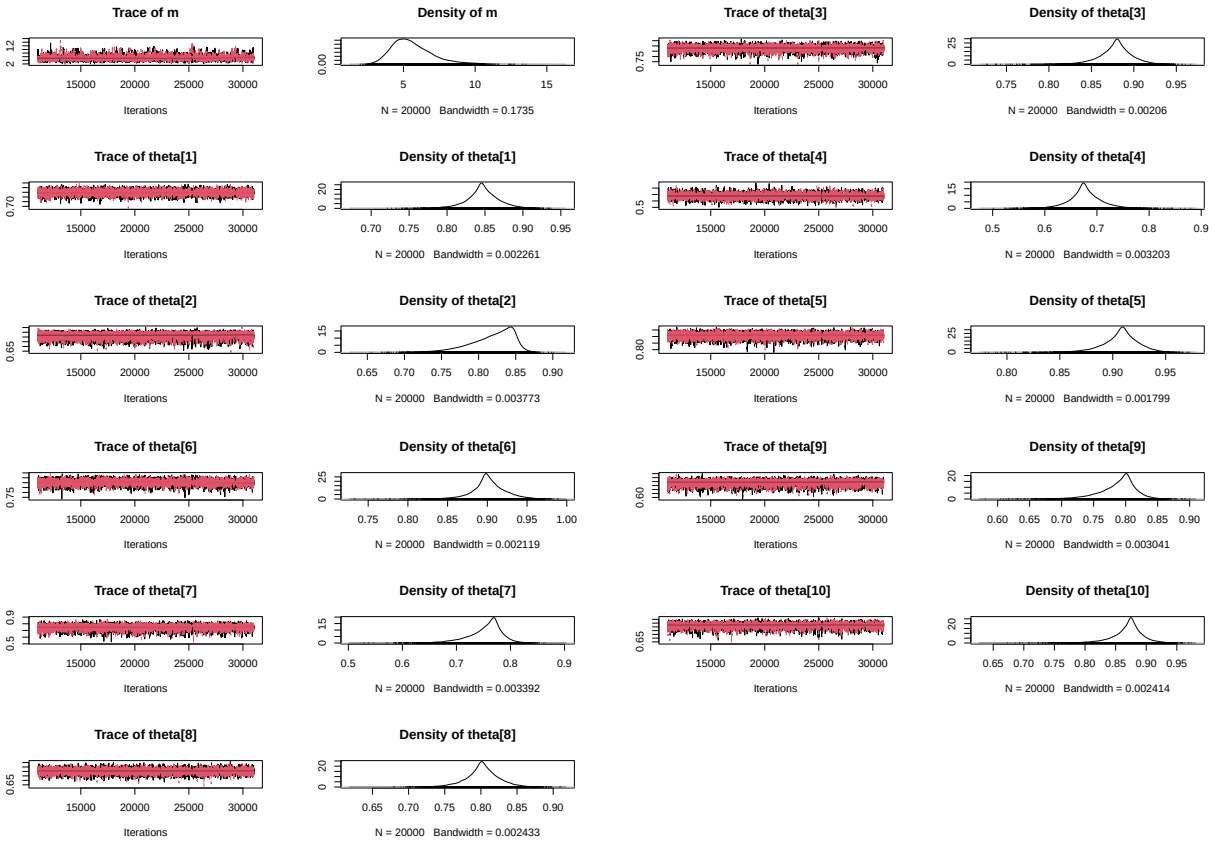
```
 #Burn-in for 10000 samples
update(model, 10000, progress.bar="none")
# Generate 20000 post-burn-in samples
```

```
params  = c("theta","m")
samples = coda.samples(model,
           variable.names=params,
           n.iter=20000, progress.bar="none")
```

```
summary(samples)
```

```
##
## Iterations = 11001:31000
## Thinning interval = 1
## Number of chains = 2
## Sample size per chain = 20000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##             Mean      SD  Naive SE Time-series SE
## m         5.6384 1.49567 7.478e-03      0.0454158
## theta[1]  0.8471 0.02218 1.109e-04      0.0001573
## theta[2]  0.8183 0.02979 1.489e-04      0.0005226
## theta[3]  0.8781 0.02088 1.044e-04      0.0001538
## theta[4]  0.6774 0.03257 1.629e-04      0.0002410
## theta[5]  0.9073 0.01764 8.821e-05      0.0001269
## theta[6]  0.9017 0.02194 1.097e-04      0.0001706
## theta[7]  0.7530 0.03296 1.648e-04      0.0003829
## theta[8]  0.8020 0.02373 1.187e-04      0.0001633
## theta[9]  0.7877 0.02935 1.467e-04      0.0003295
## theta[10] 0.8691 0.02672 1.336e-04      0.0002146
##
## 2. Quantiles for each variable:
##
##             2.5%    25%    50%    75%  97.5%
## m         3.3943 4.5947 5.3974 6.4206 9.3517
## theta[1]  0.7986 0.8356 0.8467 0.8594 0.8929
## theta[2]  0.7465 0.8014 0.8247 0.8411 0.8584
## theta[3]  0.8308 0.8680 0.8796 0.8896 0.9177
## theta[4]  0.6101 0.6605 0.6759 0.6942 0.7476
## theta[5]  0.8671 0.8986 0.9087 0.9175 0.9400
## theta[6]  0.8544 0.8911 0.9008 0.9134 0.9470
## theta[7]  0.6714 0.7375 0.7599 0.7732 0.8047
## theta[8]  0.7508 0.7897 0.8021 0.8153 0.8502
## theta[9]  0.7152 0.7735 0.7934 0.8055 0.8353
## theta[10] 0.8034 0.8580 0.8728 0.8834 0.9170
```

```
plot(samples)
```



f) What are the advantages and disadvantages of writing your own code as opposed to using JAGS in this problem and in general?

The benefit of doing your own coding is that you have more control over the algorithm; the drawbacks are that it often takes longer and is more error-prone.