(Question 4c, ,6d ,6e, 6f and convergence test)

library(rjags)

### Question 4(C)(Code Submitted in HW 7)

#data  
Y<- c(1:10)  
  
#n  
n<- length(Y)  
  
  
# of simulations  
S<- 25000  
  
samples<- matrix(NA, nrow = S, ncol = 11)  
  
#names to assign to matrix  
colnames(samples)<- c("b", "s1", "s2", "s3", "s4", "s5", "s6", "s7", "s8", "s9", "s10")  
  
  
#inital values   
sigma <- 1  
a<- 1  
b<- .1  
  
  
  
#Gibbs sampler   
for(s in 1:S){  
 for(i in 1:n){  
 #sigma  
 sigma[i] <- 1/(rgamma(1,.5 + a, ((Y[i]^2) + 2\* b)/2))  
 }  
 #b  
 b <- rgamma(1,n\*a + 1, 1+sum(1/sigma))   
 samples[s,] <- c(b, sigma)  
}  
  
median<- as.matrix(apply(samples,2,median))

### Question 4(D)

#setting values to be passed to jags  
data<- list(Y= Y, n = n)  
  
#model string construction   
model\_string <- textConnection("model{  
 # Likelihood  
 for(i in 1:n){  
 Y[i] ~ dnorm(0, tau[i])  
 }  
 # Priors  
 for(i in 1:n){  
 tau[i] ~ dgamma(1,b)  
 sigma[i]<- 1/tau[i]  
 }  
   
 b ~ dgamma(1,1)  
   
 }")  
  
#set inital value   
inits <- list( b = .1)  
model <- jags.model(model\_string,data = data, inits=inits, n.chains=2 ,quiet=TRUE)  
  
  
  
update(model, 10000, progress.bar="none")  
  
  
params <- c("sigma","b")  
samples <- coda.samples(model,   
 variable.names=params,   
 n.iter=25000, progress.bar="none")  
  
summary(samples)

##   
## Iterations = 10001:35000  
## Thinning interval = 1   
## Number of chains = 2   
## Sample size per chain = 25000   
##   
## 1. Empirical mean and standard deviation for each variable,  
## plus standard error of the mean:  
##   
## Mean SD Naive SE Time-series SE  
## b 5.552 1.98 0.008854 0.01155  
## sigma[1] 12.982 283.32 1.267054 1.26906  
## sigma[2] 13.915 49.65 0.222041 0.22303  
## sigma[3] 19.948 115.45 0.516317 0.52029  
## sigma[4] 28.106 245.90 1.099695 1.09971  
## sigma[5] 35.433 299.77 1.340597 1.34060  
## sigma[6] 49.154 599.44 2.680774 2.68080  
## sigma[7] 58.320 354.68 1.586194 1.58619  
## sigma[8] 73.020 288.33 1.289437 1.28940  
## sigma[9] 89.338 648.88 2.901881 2.90188  
## sigma[10] 109.430 518.04 2.316754 2.31677  
##   
## 2. Quantiles for each variable:  
##   
## 2.5% 25% 50% 75% 97.5%  
## b 2.461 4.124 5.289 6.687 10.09  
## sigma[1] 1.065 2.737 4.979 9.886 56.55  
## sigma[2] 1.398 3.474 6.231 12.205 68.99  
## sigma[3] 2.004 4.706 8.312 16.426 92.29  
## sigma[4] 2.758 6.465 11.341 22.327 125.41  
## sigma[5] 3.756 8.650 15.001 29.515 168.55  
## sigma[6] 5.013 11.377 19.953 39.089 221.46  
## sigma[7] 6.406 14.557 25.393 49.817 278.38  
## sigma[8] 7.959 18.112 31.764 62.116 349.69  
## sigma[9] 9.785 22.465 38.978 76.364 430.23  
## sigma[10] 11.826 27.175 46.715 90.797 533.67

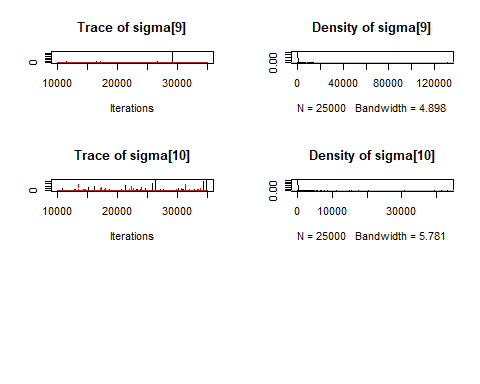
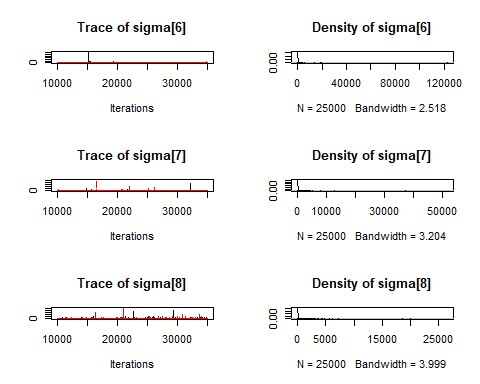
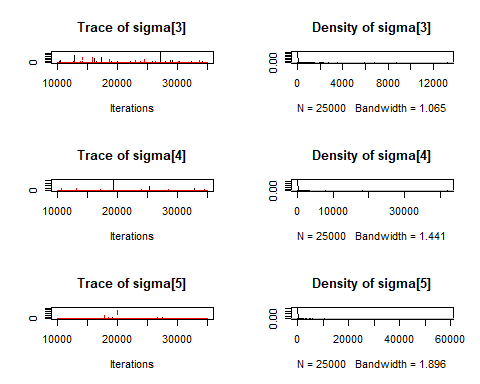
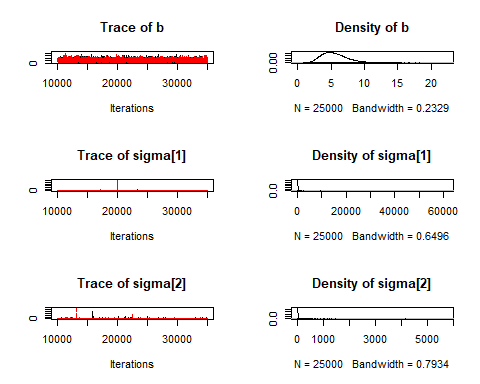
#compared JAGS median and R median   
data<- as.data.frame((summary(samples)[2])$quantiles[,3])  
colnames(data)[1]<- "Median\_Jags"  
data$Median\_R<- median  
data

## Median\_Jags Median\_R  
## b 5.289115 5.242234  
## sigma[1] 4.979287 4.848783  
## sigma[2] 6.230594 6.152924  
## sigma[3] 8.312432 8.228937  
## sigma[4] 11.341198 11.334370  
## sigma[5] 15.000504 15.192146  
## sigma[6] 19.952828 19.922581  
## sigma[7] 25.393440 25.058208  
## sigma[8] 31.764361 31.518908  
## sigma[9] 38.977932 39.107855  
## sigma[10] 46.714566 46.515393

Since the parameters are right skewed I used the median from jags and R(part c). The table above shows that the two method produce near identical medians.

### Question 4(Convergence Test)

#convergence test  
# visual   
plot(samples)



# #auto corr  
# autocorr.plot(samples)  
#   
# #chain 1  
# autocorr(samples[[1]], lag = 1)  
#   
# #chain 2  
# autocorr(samples[[2]], lag = 1)  
  
# Low ESS indicates poor convergence, size sample apperas to be large  
effectiveSize(samples)

## b sigma[1] sigma[2] sigma[3] sigma[4] sigma[5] sigma[6]   
## 29430.52 47672.60 49457.87 48924.71 50000.00 50000.00 50000.00   
## sigma[7] sigma[8] sigma[9] sigma[10]   
## 50000.00 50000.00 50000.00 50000.00

# R greater than 1.1 indicates poor convergence   
gelman.diag(samples)

## Potential scale reduction factors:  
##   
## Point est. Upper C.I.  
## b 1.00 1.00  
## sigma[1] 1.27 1.27  
## sigma[2] 1.00 1.00  
## sigma[3] 1.06 1.06  
## sigma[4] 1.21 1.21  
## sigma[5] 1.24 1.24  
## sigma[6] 1.04 1.04  
## sigma[7] 1.00 1.01  
## sigma[8] 1.02 1.02  
## sigma[9] 1.26 1.26  
## sigma[10] 1.09 1.09  
##   
## Multivariate psrf  
##   
## 1

# |z| greater than 2 indicates poor convergence  
# chain 1  
geweke.diag(samples[[1]])

##   
## Fraction in 1st window = 0.1  
## Fraction in 2nd window = 0.5   
##   
## b sigma[1] sigma[2] sigma[3] sigma[4] sigma[5] sigma[6]   
## -0.946161 -0.919236 0.391435 -0.008326 -1.620096 -0.311569 0.360606   
## sigma[7] sigma[8] sigma[9] sigma[10]   
## -0.784960 -1.762610 -0.019404 -1.894560

#chain 2  
geweke.diag(samples[[2]])

##   
## Fraction in 1st window = 0.1  
## Fraction in 2nd window = 0.5   
##   
## b sigma[1] sigma[2] sigma[3] sigma[4] sigma[5] sigma[6]   
## 1.1567 -0.5748 0.9379 -1.4697 0.2032 -4.0349 1.3727   
## sigma[7] sigma[8] sigma[9] sigma[10]   
## -1.2218 -1.2509 0.6848 -1.7080

From the Geweke test sigma 8 has poor convergence in chain 1. This is also confirmed with the Rubin test across all two chains , where sigma 8 is greater than 1.1. The other parameters show good convergence across both chains according to the Gelman Rubin test. . Also our sample size are large enough based on effective sample size test.

### Question 6(D)

set.seed(1)  
# overall proportions   
q<- c(.845, .847, .880, .674, .909, .898, .770, .801, .802, .875)  
# number of made  
Y<- c(64, 72, 55, 27, 75, 24, 28, 66, 40, 13)  
# number of attempts  
n<- c(75,95,63, 39, 83, 26, 41, 82, 54, 16)  
  
N<-length(Y)  
  
S<- 50000  
#set intial values   
  
theta<- .5  
  
m<- 0.5  
  
#log posterior  
log\_post<- function(theta, n, q, m, Y){  
 like= sum(dbinom(Y, size = n,theta, log = T ))  
 a = q \* exp(m)  
 b = (1-q)\* exp(m)  
 prior1= sum(dbeta(theta, a,b, log = T))  
 prior2= dnorm(m,0, 10, log = T)  
 return(like + prior1 + prior2)  
}  
  
  
#matrix to hold results  
samples<- matrix(NA, nrow = S, ncol = 11)  
colnames(samples)<- c("m", paste("Theta\_", sep= "", 1:10))  
  
  
#canidate std  
can\_sd<- .011  
  
#tuning variables   
burn <- 5000 # Length of burn-in period for tuning  
check <- 100 # Iterations between checks of the acceptance rate   
att <- 0 # Keep track of the number of MH attempts  
acc <- 0  
  
  
  
  
for(i in 1: S){  
   
 #metro sampling   
 can = rnorm(1,m, can\_sd)  
 logR <- log\_post(theta, n, q, can, Y)-log\_post(theta, n, q, m, Y)   
 #record attemps  
 att = att + 1  
 if(log(runif(1))<logR){  
 m <- can  
 acc<- acc + 1  
 }  
  
   
 #tunning  
 if(i<burn & att==check){  
 if(acc/att<0.2){can\_sd<-can\_sd\*0.02}  
 if(acc/att>0.6){can\_sd<-can\_sd\*.06}  
 acc <- att <- 0  
 }  
  
 #gibbs  
 for( j in 1: N){  
 a<- Y[j]+ q[j]\* exp(m)  
 b<- n[j] - Y[j] + exp(m) \* (1-q[j])  
 theta[j]= rbeta(1,a, b)  
 }  
   
 samples[i,]= c(m, theta)  
}  
  
  
x<- merge(as.matrix(colMeans(samples)),t(as.matrix(apply(samples, 2,quantile, probs = c(0.025, 0.975)))), by = "row.names",all = TRUE )  
  
x$id<- ifelse(substring(x$Row.names, 7) != "", as.double(substring(x$Row.names, 7)), 0)  
  
colnames(x)[1]<- "para"  
colnames(x)[2]<- "mean"  
  
sum<- (x[order(x$id), ])[,1:4]  
  
  
sum

## para mean 2.5% 97.5%  
## 1 m 0.6343345 0.6345580 0.6345580  
## 2 Theta\_1 0.8534135 0.7672704 0.9225228  
## 4 Theta\_2 0.7592785 0.6701674 0.8385276  
## 5 Theta\_3 0.8734014 0.7832360 0.9421668  
## 6 Theta\_4 0.6917797 0.5441084 0.8224357  
## 7 Theta\_5 0.9037503 0.8334156 0.9566370  
## 8 Theta\_6 0.9212671 0.7974477 0.9886383  
## 9 Theta\_7 0.6862729 0.5408174 0.8140205  
## 10 Theta\_8 0.8047330 0.7136983 0.8815787  
## 11 Theta\_9 0.7430570 0.6219455 0.8478757  
## 3 Theta\_10 0.8186820 0.6162911 0.9545422

Table above shows the mean and 95% CI for my hand written MCMC.

### Question 6(E)

#data passed to JAGS  
data<- list(Y= Y, n = n, q = q , N=N)  
  
  
  
model\_string <- textConnection("model{  
 # Likelihood  
 for(i in 1:N){  
 Y[i] ~ dbinom(theta[i], n[i] )  
 }  
 # Priors  
 for(i in 1:N){  
 theta[i] ~ dbeta(exp(m)\* q[i], exp(m)\* (1-q[i]))  
 }  
  
 m ~ dnorm(0,10)  
  
 }")  
  
  
model <- jags.model(model\_string,data = data, n.chains=2 ,quiet=TRUE)  
  
update(model, 10000, progress.bar="none")  
  
  
params <- c("theta","m")  
samples <- coda.samples(model,   
 variable.names=params,   
 n.iter=25000, progress.bar="none")  
  
#summary output  
summary(samples)

##   
## Iterations = 11001:36000  
## Thinning interval = 1   
## Number of chains = 2   
## Sample size per chain = 25000   
##   
## 1. Empirical mean and standard deviation for each variable,  
## plus standard error of the mean:  
##   
## Mean SD Naive SE Time-series SE  
## m 0.6588 0.29438 0.0013165 0.0016896  
## theta[1] 0.8529 0.04010 0.0001793 0.0002319  
## theta[2] 0.7597 0.04290 0.0001919 0.0002408  
## theta[3] 0.8728 0.04123 0.0001844 0.0002519  
## theta[4] 0.6912 0.07157 0.0003201 0.0004078  
## theta[5] 0.9038 0.03185 0.0001425 0.0001932  
## theta[6] 0.9211 0.04978 0.0002226 0.0003599  
## theta[7] 0.6868 0.06977 0.0003120 0.0003975  
## theta[8] 0.8046 0.04273 0.0001911 0.0002453  
## theta[9] 0.7433 0.05835 0.0002610 0.0003317  
## theta[10] 0.8193 0.08822 0.0003945 0.0005591  
##   
## 2. Quantiles for each variable:  
##   
## 2.5% 25% 50% 75% 97.5%  
## m 0.07935 0.4607 0.6584 0.8553 1.2383  
## theta[1] 0.76659 0.8274 0.8560 0.8813 0.9226  
## theta[2] 0.67058 0.7317 0.7615 0.7896 0.8393  
## theta[3] 0.78105 0.8472 0.8767 0.9026 0.9416  
## theta[4] 0.54218 0.6440 0.6948 0.7416 0.8219  
## theta[5] 0.83247 0.8842 0.9070 0.9269 0.9563  
## theta[6] 0.79961 0.8938 0.9304 0.9584 0.9886  
## theta[7] 0.54250 0.6406 0.6897 0.7358 0.8148  
## theta[8] 0.71484 0.7771 0.8070 0.8346 0.8812  
## theta[9] 0.62227 0.7049 0.7458 0.7849 0.8493  
## theta[10] 0.61730 0.7658 0.8311 0.8849 0.9552

Summary output from JAGS.

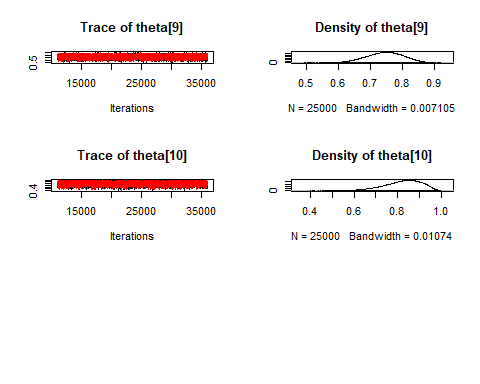
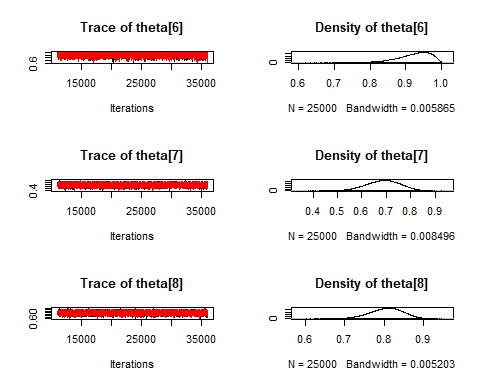
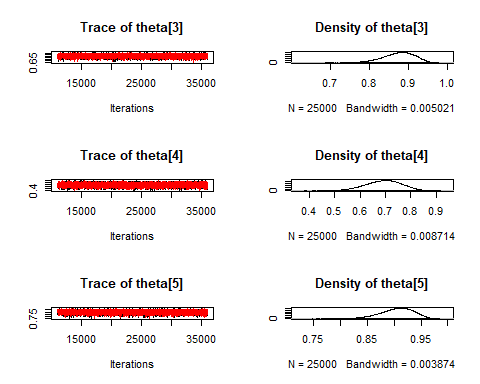
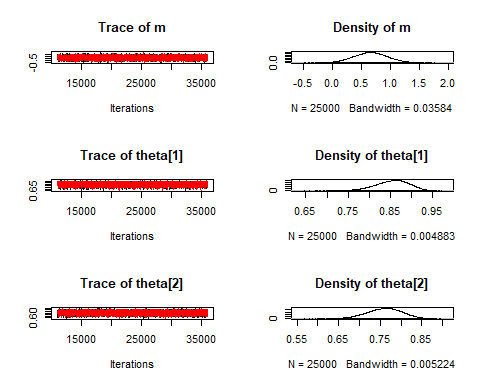
#compariosn of mean  
  
com<- as.data.frame(sum[1:2])  
colnames(com)[2]<-"Mean\_R"  
d<- summary(samples)[1]  
com$Mean\_jags<- d$statistics[,1]  
  
com

## para Mean\_R Mean\_jags  
## 1 m 0.6343345 0.6587725  
## 2 Theta\_1 0.8534135 0.8528748  
## 4 Theta\_2 0.7592785 0.7596951  
## 5 Theta\_3 0.8734014 0.8728050  
## 6 Theta\_4 0.6917797 0.6911805  
## 7 Theta\_5 0.9037503 0.9037628  
## 8 Theta\_6 0.9212671 0.9210517  
## 9 Theta\_7 0.6862729 0.6867638  
## 10 Theta\_8 0.8047330 0.8045720  
## 11 Theta\_9 0.7430570 0.7433087  
## 3 Theta\_10 0.8186820 0.8192602

I used the mean to do a comparison. The mean values for each of the 11 parameters are very close across both methods. As shown in the table above.

### Question 6(Convergence Test)

#convergence test  
# visual   
plot(samples)



# #auto corr  
# autocorr.plot(samples)  
  
# Low ESS indicates poor convergence, size sample apperas to be large  
effectiveSize(samples)

## m theta[1] theta[2] theta[3] theta[4] theta[5] theta[6]   
## 30362.33 29923.43 31759.43 26796.40 30793.45 27179.17 19161.78   
## theta[7] theta[8] theta[9] theta[10]   
## 30812.31 30342.69 30963.77 24897.17

# R greater than 1.1 indicates poor convergence, therefore we have good convergence.   
gelman.diag(samples)

## Potential scale reduction factors:  
##   
## Point est. Upper C.I.  
## m 1 1  
## theta[1] 1 1  
## theta[2] 1 1  
## theta[3] 1 1  
## theta[4] 1 1  
## theta[5] 1 1  
## theta[6] 1 1  
## theta[7] 1 1  
## theta[8] 1 1  
## theta[9] 1 1  
## theta[10] 1 1  
##   
## Multivariate psrf  
##   
## 1

# |z| greater than 2 indicates poor convergence  
geweke.diag(samples[[1]])

##   
## Fraction in 1st window = 0.1  
## Fraction in 2nd window = 0.5   
##   
## m theta[1] theta[2] theta[3] theta[4] theta[5] theta[6]   
## 1.0577 1.4417 -1.6308 -1.3575 -0.3272 0.5397 0.3655   
## theta[7] theta[8] theta[9] theta[10]   
## -0.8405 1.2779 -0.4503 -1.7493

Chain converge according to Gelman Test and our sample size is adequately large enough to indicate convergence.

### Question 6(F)

Advantage of writing your own code

* Great understanding of MCMC to write your own codes.
* You can set the values of certain parameters in comparison to jags setting the values for these parameters, more control.
* Seems much faster to run

Disadvantages

* More coding
* More prone to mistakes since you have to write your own posterior.
* A pain setting candidate values and figuring out some values.
* Harder to debug