ST540 HW8

4. (d) The code for running JAGS model is attached at the end of the solution. One thing that needs attention is that JAGS uses mean-precision parameterization for normal distribution.

   We can compare the results from JAGS and the results in (c) using some summary statistics. For illustration purpose, only part of the parameters are shown here.

Parameter	n_eff	mean	sd	2.5%	25%	50%	75%	97.5%
b	29696	5.5	2.0	2.5	4.1	5.3	6.7	10.1
σ_3^2	49984	19.8	103.8	2.0	4.8	8.4	16.5	95.3
σ_5^2	49554	35.4	230.4	3.8	8.7	15.1	29.7	167.1
σ_7^2	50000	58.3	269.4	6.4	14.5	25.3	49.4	288.9

Table 1: Summary Statistics from JAGS

Parameter	n_eff	mean	sd	2.5%	25%	50%	75%	97.5%
b	29300	5.6	2.0	2.4	4.1	5.3	6.7	10.1
σ_3^2	49760	20.9	215.2	2.0	4.8	8.4	16.5	95.9
σ_5^2	50000	34.9	132.5	3.8	8.7	15.2	29.7	171.4
σ_7^2	50000	58.7	302.9	6.4	14.6	25.4	49.3	279.3

Table 2: Summary Statistics from hand-crafted Gibbs sampler

   As shown in the tables, the summary statistics of the posterior samples from JAGS and the hand-crafted Gibbs sampler are very similar, except for the standard deviance. A possible reason is that R and JAGS use different random number generators.

   I recommend to use coda library to sample from a JAGS model, which saves the samples in a mcmc.list object. Then you can make use of the functions provided by coda to visualize the results, perform convergence check, etc. The effective sample sizes are also calculated and included in the table. Again, we find that they are quite close.

6. (a) First of all, θ_i is the probability of making clutch in each attempt, and the support of Beta distribution is $(0, 1)$. And the expectation of the prior is:

$$E(\theta_i|m) = \frac{e^m q_i}{e^m q_i + e^m(1-q_i)} = q_i \tag{1}$$

   Thus, we believe that θ_i is around q_i that is the overall proportion of clutch makes. It's obviously a reasonable assumption.

   (b) The variance of the prior is:

$$Var(\theta_i|m) = \frac{e^m q_i \cdot e^m(1-q_i)}{(e^m q_i + e^m(1-q_i))^2(e^m q_i + e^m(1-q_i)+1)} = \frac{q_i(1-q_i)}{e^m+1} \tag{2}$$

   The role of m is to adjust the strength of the prior. For example, with very large m, the variance of the prior would be very small. It means that we strongly believe that θ_i is close to q_i, and such prior information will dominate the posterior. On the hand, if m is very small, we would have a weak prior on m and the information from data will dominate the posterior.

   (c)

$$P(\theta_1^2, ..., \theta_{10}^2, m|Y_1, ..., Y_{10}, n_1, ..., n_{10}) \propto [\prod_{i=1}^{10} P(Y_i|n_i, \theta_i)][\prod_{i=1}^{10} P(\theta_i|m)]P(m) \tag{3}$$

   After dropping all the term not related to θ_1, we have:

$$P(\theta_1^2|\theta_2^2, ..., \theta_{10}^2, m, Y_1, ..., Y_{10}, n_1, ..., n_{10}) \propto P(Y_1|n_1, \theta_1)P(\theta_1|m) \tag{4}$$

   This is a Binomial-Beta conjugate pair and the conditional posterior of θ_1 is:

$$\theta_1^2|\theta_2^2, ..., \theta_{10}^2, m, Y_1, ..., Y_{10}, n_1, ..., n_{10} \sim Beta(Y_1 + e^m q_1, n_1 - Y_1 + e^m(1-q_1)) \tag{5}$$

(d) From (c), we are able to derive the full conditional posterior for $\{\theta_i\}_{i=1}^{10}$ easily. However, it's not the case for m:

$$P(m|\theta_1, ..., \theta_{10}, Y_1, ..., Y_{10}, n_1, ..., n_{10}) \propto [\prod_{i=1}^{10} P(\theta_i|m)]P(m) \tag{6}$$

In this situation, we can use Metropolis–Hastings algorithm to sample from the posterior of m, because we have the kernel of the full conditional posterior of m.

Therefore, our MCMC algorithm is a combination of Gibbs sampler(for $\{\theta_i\}_{i=1}^{10}$) and MH algorithm(for m). To sample m, $\text{N}(\cdot|m^t, 1.3^2)$ can be a good proposal density, where m^t is the sample of m in the last iteration. Note that you may need to try different standard deviance for proposal density to make the acceptance rate around 0.4.

Parameter	n_eff	mean	sd	2.5%	50%	97.5%
m	3193	5.63	1.45	3.38	5.42	9.06
θ_1	194422	0.85	0.02	0.80	0.85	0.89
θ_2	9511	0.82	0.03	0.75	0.82	0.86
θ_3	192769	0.88	0.02	0.83	0.88	0.92
θ_4	185338	0.68	0.03	0.61	0.68	0.75
θ_5	188996	0.91	0.02	0.87	0.91	0.94
θ_6	121962	0.90	0.02	0.86	0.90	0.95
θ_7	23352	0.75	0.03	0.67	0.76	0.81
θ_8	197935	0.80	0.02	0.75	0.80	0.85
θ_9	31911	0.79	0.03	0.72	0.79	0.84
θ_{10}	85359	0.87	0.03	0.80	0.87	0.92

Table 3: Posterior summary statistics(1000000 iterations, 800000 burn-ins)

The effective sample size is well above 1000(Note that burn-ins are dropped when calculating ESS). And the acceptance rate is 0.40 that is pretty ideal.

(e) The results from JAGS are very similar with those in (d).

Parameter	Rhat	n_eff	mean	sd	2.5%	50%	97.5%
m	1.00	6348	5.60	1.44	3.37	5.39	9.05
θ_1	1.00	181218	0.85	0.02	0.80	0.85	0.89
θ_2	1.00	18963	0.82	0.03	0.74	0.82	0.86
θ_3	1.00	183539	0.88	0.02	0.83	0.88	0.92
θ_4	1.00	184759	0.68	0.03	0.61	0.68	0.75
θ_5	1.00	173816	0.91	0.02	0.87	0.91	0.94
θ_6	1.00	141278	0.90	0.02	0.85	0.90	0.95
θ_7	1.00	44865	0.75	0.03	0.67	0.76	0.81
θ_8	1.00	190194	0.80	0.02	0.75	0.80	0.85
θ_9	1.00	52310	0.79	0.03	0.71	0.79	0.84
θ_{10}	1.00	96232	0.87	0.03	0.80	0.87	0.92

Table 4: Posterior summary statistics from JAGS(1000000 iterations, 800000 burn-ins)

The Gelman-Ruban statistics are well below 1.01 and the ESS for m is significantly larger than that in (c).

(f) For an applied Bayesian project, it's not a good idea to write your own MCMC algorithm. Because it can be very error-prone and time-consuming. Black-box MCMC samplers, such as JAGS and Stan, are highly optimized for efficiency with advanced features like auto-tuning. They also come with handy functions to let you analyze the results conveniently.

However, if you are focusing on a specific class of models, it might be beneficial to write your own MCMC algorithms. There is a good chance that your algorithms are faster or more robust than JAGS by specializing for a given model. But this can be time-consuming too.

```
##################################################################
## 4(d) JAGS part
##################################################################
library(coda)
library(rjags)
Y = seq(1, 10)
# JAGS model as a string
model_string_p4 = textConnection("model{
                                   # Likelihood
                                   for (i in 1:10){
                                     Y[i] ~ dnorm(0, precision[i])
                                   }
                                   # Priors
                                   for(i in 1:10){
                                     precision[i] ~ dgamma(1,b)
                                     sigmasq[i] <- 1 / (precision[i])
                                   }
                                   b ~ dgamma(1,1)
                                 }")
# same initialization as in (c)
inits = list(precision = 1 / rep(0.1, 10), b=1)
# create JAGS model
model_p4 = jags.model(model_string_p4, data=list(Y=Y), inits=inits,
                       n.chains=4, n.adapt=0)
# 150000 burnin + 50000 samples = 200000
# coda.samples returns a mcmc.list object
params = c("sigmasq", "b")
JAGS_samples_p4 = coda.samples(model_p4, variable.names=params, n.iter=200000)

# get the summary statistics from JAGS model
# ignore the burnins
summary(window(JAGS_samples_p4, start=150001))

# get the summary statistics from hand-crafted model
# first convert the sample matrix in (c) to mcmc.list object
samples_p4c = as.mcmc.list(mcmc(samples_p4c))
summary(samples_p4c)


##################################################################
## 6(d)
##################################################################
# 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)

# intial values for the parameters
m = 0
theta = q

# conditional posterior of m
log_post_m = function(m, q, theta)
{
  part1 = sum(dbeta(theta, exp(m) * q, exp(m) * (1 - q), log=TRUE))
  part2 = dnorm(m, 0, 10^0.5, log=TRUE)
  return(part1 + part2)
}

# number of iterations
```

```
N = 1000000
# number of burn-ins
burnin = 800000
# matrix to save the posterior samples
samples_6d = matrix(NA, N - burnin, 10 + 1)
colnames(samples_6d) = c("theta1","theta2","theta3","theta4","theta5","theta6",
                         "theta7","theta8","theta9","theta10","m")

cnt_accept = 0
# MCMC
for(t in 1:N){
  # gibbs sampler
  theta = rbeta(10, Y + exp(m) * q, n - Y + exp(m) * (1 - q))
  # MH
  can_m = rnorm(1, m, 1.3)
  # probability of acceptance(log scale)
  log_P = log_post_m(can_m, q, theta) - log_post_m(m, q, theta)
  if(log(runif(1)) < log_P)
  {
    m = can_m
    if(t > burnin)
      cnt_accept = cnt_accept + 1
  }

  # save samples
  if(t > burnin)
    samples_6d[t-burnin,] = c(theta, m)
}

samples_6d = as.mcmc.list(mcmc(samples_6d))
# posterior mean and credible interval
summary(samples_6d)
# check the effective sample size
effectiveSize(samples_6d)
# acceptance rate
cnt_accept/(N - burnin)


####################################################
## 6(e)
####################################################
model_string_p6 = textConnection("model{
                          # likelihood
                          for(i in 1:10){
                            Y[i] ~ dbin(theta[i], n[i])
                          }
                          # prior
                          for(i in 1:10){
                            theta[i] ~ dbeta(q[i]*exp(m), (1-q[i])*exp(m))
                          }
                          m ~ dnorm(0,1/10)
                          }")
model_p6 = jags.model(model_string_p6, data = list(Y=Y, n=n, q=q),
                      n.chains = 4, n.adapt=0)
# 800000 burnin + 200000 samples = 1000000
# coda.samples returns a mcmc.list object
params = c("m", "theta")
JAGS_samples_p6 = coda.samples(model_p6, variable.names=params, n.iter=1000000)

# get the summary statistics from JAGS model
# ignore the burnins
```

```
summary(window(JAGS_samples_p6, start=800001))
```