ST440/540 Applied Bayesian Analysis
Lab activity for 3/4/2024

A. HOMEWORK AND QUIZ SOLUTIONS

Quiz 5: The optimal acceptance rate for Metropolis sampling is 20-40%.

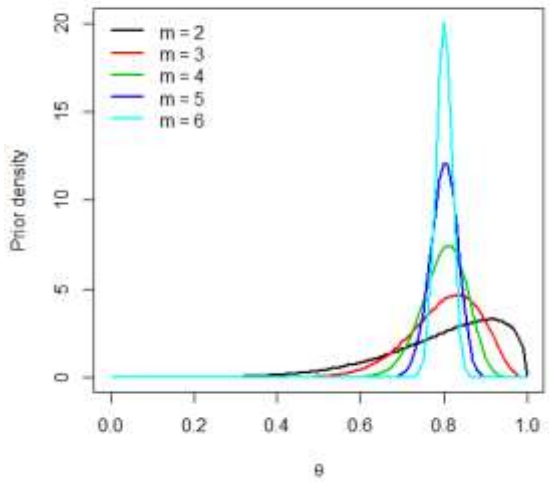
(a) Why is a lower acceptance rate suboptimal? The algorithm gets stuck on particular values for several iterations and doesn't explore the full posterior efficiently.

(b) Why is a high acceptance rate suboptimal? The algorithm makes only small steps every iteration and doesn't explore the full posterior efficiently.

Chapter 3, problem 6

(a) Because the prior mean of theta is q_i, so the clutch percentage prior is centered on the overall percentage.

(b) If prior variance is controlled by m. Below is the prior with q_i = 0.8 and various m




```
theta <- seq(0,1,0.01)
q     <- 0.8
m     <- seq(2,6,1)
plot(NA,xlim=0:1,ylim=c(0,20),xlab=expression(theta),ylab="Prior density")
for(i in 1:length(m)){
  lines(theta,dbeta(theta,exp(m[i])*q,exp(m[i])*(1-q)),lwd=2,col=i)
}
legend("topleft",paste("m =",m),lwd=2,col=1:length(m),bty="n")
```

(c) The full posterior is proportional to

$$f(Y_1|\theta_1)*\ldots*f(Y_n|\theta_n)*\pi(\theta_1|m)*\ldots*\pi(\theta_n|m)*\pi(m)$$

For the full conditional of theta1 all terms that don't involve theta1 can be absorbed into the proportionality constant, and so the full conditional distribution for theta1 is proportional to

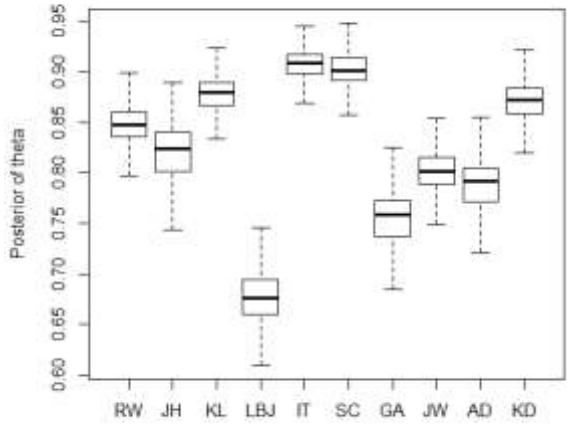
$$f(Y_1|\theta_1)\,\pi(\theta_1|m)$$

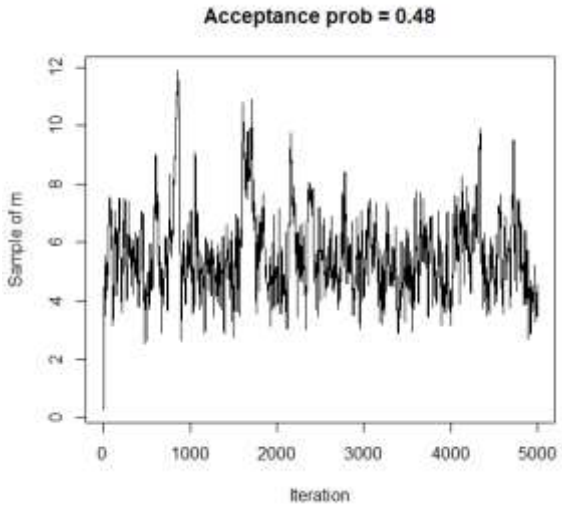
We are left with a binomial likelihood and beta prior. So, following the usual beta-binomial conjugacy results with a=exp(m)*q1 and b = exp(m)*(1-q1) we have that

$$\theta_1|Y1 \sim \text{Beta}(Y1+\exp(m)*q1, n1-Y1-\exp(1-q1)).$$

(d) Here is the code. It uses a Gibbs step for theta and Metropolis for m. Because m can be any real number (this is why we decided to use exp(m) rather than m in the beta prior) we use a Gaussian candidate distribution.

```
N      <- 10
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)
player <- c("RW","JH","KL","LBJ","IT","SC","GA","JW","AD","KD")

iters      <- 5000
burn       <- 1000
keep_theta <- matrix(0,iters,N)
keep_m     <- rep(0,N)
colnames(keep_theta) <- player

theta  <- q
m      <- 0
can_sd <- 1
for(iter in 1:iters){
   theta <- rbeta(N,Y+exp(m)*q,n-Y+exp(m)*(1-q))

   can <- rnorm(1,m,can_sd)
   R   <- sum(dbeta(theta,exp(can)*q,exp(can)*(1-q),log=TRUE))-
          sum(dbeta(theta,  exp(m)*q,  exp(m)*(1-q),log=TRUE))+
          dnorm(can,0,sqrt(10),log=TRUE)-
          dnorm(  m,0,sqrt(10),log=TRUE)
   if(log(runif(1))<R){m <- can}

   keep_theta[iter,] <- theta
   keep_m[iter]      <- m
}

acc_rate <- mean(keep_m[2:iters]!=keep_m[2:iters-1])
boxplot(keep_theta[burn:iters,],ylab="Posterior of theta",outline=FALSE)
plot(keep_m,type="l",xlab="Iteration",ylab="Sample of m",
     main=paste("Acceptance prob =",round(acc_rate,2)))
```

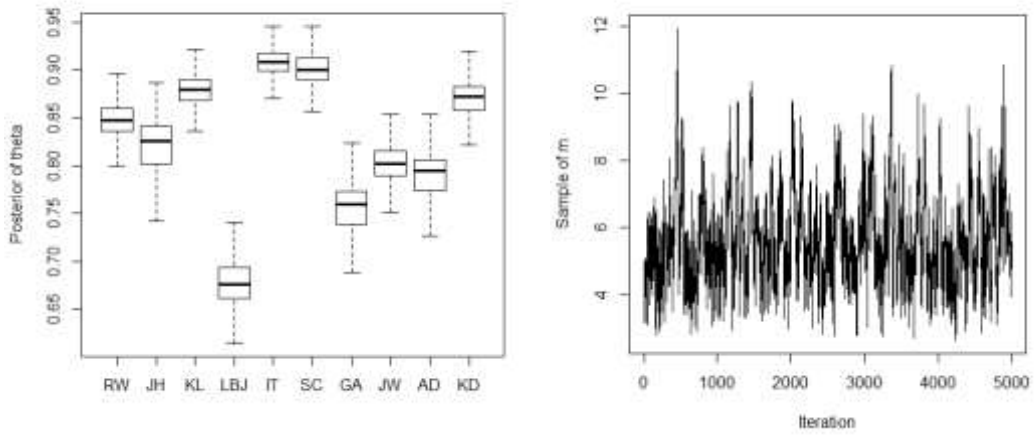




Convergence looks pretty good for m, theta is even better.

(e) Here is the JAGS code

```
library(rjags)

model_string <- textConnection("model{
   for(i in 1:10){
     Y[i]     ~ dbin(theta[i],n[i])
     theta[i] ~ dbeta(q[i]*exp(m),(1-q[i])*exp(m))
   }
   m  ~  dnorm(0, 0.1)
}")

data  <- list(n=n,Y=Y,q=q)
inits <- list(theta=q,m=0)
model <- jags.model(model_string,data = data,
         inits=inits, n.chains=1,quiet=TRUE)
update(model, 1000, progress.bar="none")
params  <- c("theta","m")
samples <- coda.samples(model,
           variable.names=params,
           n.iter=5000, progress.bar="none")
samples <- as.matrix(samples[[1]])
m       <- samples[,1]
theta   <- samples[,-1]
colnames(theta) <- player
boxplot(theta,ylab="Posterior of theta",outline=FALSE)
plot(m,type="l",xlab="Iteration",ylab="Sample of m")
```



The results are the same of the previous question.

(f) The advantage of coding it yourself is you have more control over the algorithm, disadvantages are that it generally takes longer to code and is more prone to error.

B. DISCUSSION QUESTIONS

(1) Describe to the class the error messages on

https://www4.stat.ncsu.edu/~bjreich/BSMdata/errors.html

(i) Forget to pass a variable as data

(ii) Missing values

(iii) Defining a variables twice

(iv) Priors with invalid range or initial values

(v) Passing variables that are not used

(vi) Sending an invalid list of parameters to keep

(2) Write the following model in JAGS code:

https://www4.stat.ncsu.edu/~bjreich/BSMdata/MH_concussion.html

The model is $Y_i \sim \text{Poisson}(N\lambda_i)$ where $\lambda_i = \exp(\beta_1 + i\beta_2)$ and the priors are $\beta_1, \beta_2 \sim \text{Normal}(0, \tau)$.

```
model{
 for(i in 1:4){
   Y[i] ~ dpois(lambda[i])
   log(lambda[i]) <- beta1+beta2*i
 }
 beta1 ~ dnorm(0,0.1) # setting prior variance to tau=10.
 beta2 ~ dnorm(0,0.1)
}
```

(3) The analyses linked below compare four different Bayesian software packages

https://www4.stat.ncsu.edu/~bjreich/BSMdata/software.html

https://www4.stat.ncsu.edu/~bjreich/BSMdata/software2.html

Rank (keeping in mind that this a very small example) the four packages in terms of

(a) Speed per iteration – JAGS was fastest (STAN is faster for harder problems)

(b) Convergence – JAGS was easier to tell it converged, OpenBUGS plots were confusing, ESS was highest for STAN

(c) User interface – STAN is complicated, others are about the same

(4) Examine the output in the analysis posted at

https://www4.stat.ncsu.edu/~bjreich/logistic_bball.html

and summarize MCMC converge for each model

(a) Linear: Convergence is pretty good, could be run a bit longer to get ESS > 100

(b) Quadratic: It seems to have converged (Geweke and R are OK) but ESS is low so it needs to run longer

(c) Cubic: All metrics indicate the chains have not converged.

(5) List 10 things you might try if MCMC doesn't converge:

1. Run it longer
2. Better initial values
3. Different candidate distribution (M-H v Metropolis)
4. Use STAN or NIMBLE
5. Change priors (beta instead for normal prior, make them tighter/reduce variance)
6. Pick prior based on data (empirical bayes, get MLEs) or fix some parameters
7. Get more data
8. **Simplify the model**!
9. Check identifiability
10. Double check code
11. Chose a fancier algorithm, like adaptive MH