Lab 5 Hints

TASK 1:

Functional skeleton:
Make a coin-die bbox

```
# k = number of faces in the event set E for acceptance of proposal (we have set of 2); K=1...6
# lik = likelihood for 2 states
# theta = two states = 0.4 and 0.8
# h1 = small relative to h2
cdbbox<-function(k=1,lik,theta, h1="s")
{
 # rename the first and second components of the likelihood
 lik1<-lik[1]
 lik2<-lik[2]
 # We will now make a prior that has the desired characteristics
 # if h1 small "s" then ... else ...
 ifelse(h1=="s",pi1<-k/6*lik2/(lik1+k/6*lik2), pi1<-lik2/(lik2+k/6*lik1))
 # sum of probs is 1
 prior=c(pi1,1-pi1)
 lik<-c(lik1,lik2)
 h<-prior*lik
 # Bayes
 post=h/sum(h)
}
```

The prior in the coin die is set so that the ratio of h will be k/6 where k is the number of faces in a fair dice needed to match the probability of k/6.

EX: If k=2 then the acceptance set can be any number of two faces of the die ({1,2}, or {4,6}, etc). If any of the acceptance set values is thrown by a dice the proposal would be accepted with probability 2/6=1/3. However, if a number outside of the acceptance set is thrown the proposal would be rejected with probability 4/6=2/3.

1.a.iii) A larger acceptance set would mean the lower state (from proposed values) will be accepted more frequently which may not necessarily be desired.

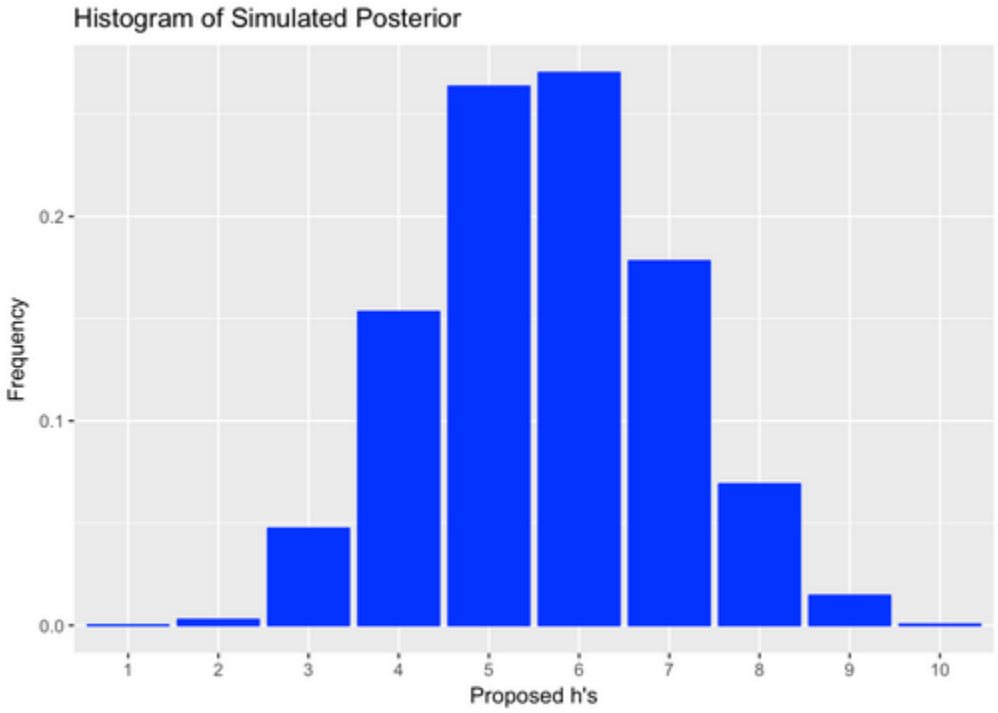
```
ifelse(h1=="s",pi1<-k/6*lik2/(lik1+k/6*lik2), pi1<-lik2/(lik2+k/6*lik1))
```

2.)
The above piece of code is calculating prior probabilities (in a 2 state MCMC context) from likelihood values in such a way that underlying h ratios conform to desired probabilities generated from throwing an unbiased die.

In other words, the prior will be calculated based on idea that hj/hi=k/6 where the sampler moves from state i to state j and k is the cardinality of acceptance event space Ej that is tied to throwing of a fair die.

TASK 4

Using simRQ(), generate all the inputs into the simulation. This proposal will have eleven values and will peak in the middle. H, as before, will have a uniform prior and likelihood with x=4 and n=10.



TASK 4/5

Alpha, within the for loop, must be adjusted to take non-uniform proposals.

```
alpha[i]=min(1, h[prop[i]]*q(post[i-1])/(h[post[i-1]]*q(prop[i])))
```

TASK 6:

OpenBUGS:

Note that OpenBUGS uses a mean and precision for classifying a distribution. Because of this a link between tau (the precision) and sigma (the standard deviation) must be expressed. We do this as a logical node with where tau=pow(sigma,-2). Sigma is set with a non-informative prior as a stochastic node using a uniform distribution.

Classical method confidence intervals and point estimates:

```
library(s20x)   #invoke s20x package
y.lm=lm(y~x)    #creates linear model of data
quad.lm=lm(y~x + I(x^2))     #creates a quadratic model of data
ci=ciReg(y.lm)    #creates confidence intervals for betas of linear
     model
ciq=ciReg(quad.lm)      #creates confidence intervals for betas of
     quadratic model
sum=summary(y.lm)       #creates a summary of the linear model - this
     summary includes information on the residuals, coefficient point
```

```
      estimates and classical t-test info. For our purposes in this lab
      we will focus on the coefficient estimates which can be pulled by
      calling the object sum$coefficients and reading the Estimate
      column
sumq=summary(quad.lm)   #creates a summary of the quadratic model
      (same format as the linear model summary)
```

***NOTE: If you are having trouble running jags make sure you have installed the latest update of the rjags package.