CHL5223 A3

Belina Jang

Question 1

(a) Run between 10,000 to 30,000 iterations of the MCMC. For the three parameters, provide a copy of the trace plot for a portion of the iteration, the autocorrelation plot, and the statistics.

```
library(here)
library(kableExtra)
#  SmokeHyperBaseR.txt
#  Michael Escobar
#  March 5, 2017


WorkDir<- here()
#  Note: change working directory to where everything is

setwd(WorkDir)

library(R2jags)

#
#
#data from Healy, page 90.
#  (MJR Healy, 1988, Glim: An Introduction, Clarendon Press: Oxford.)
#  Looking to see if Smoking is a risk factor for hypertension, controlling
 ↪  for obesity, snoring, and gender
#  Note 1: there was no males or females who were smokers and obese and who
 ↪  did not snore (so 1 1 0 had no exposures)
#  Note 2: here we are simply looking at the effect of smoking given the
 ↪  other factors.  We are ignoring the possibility that
```

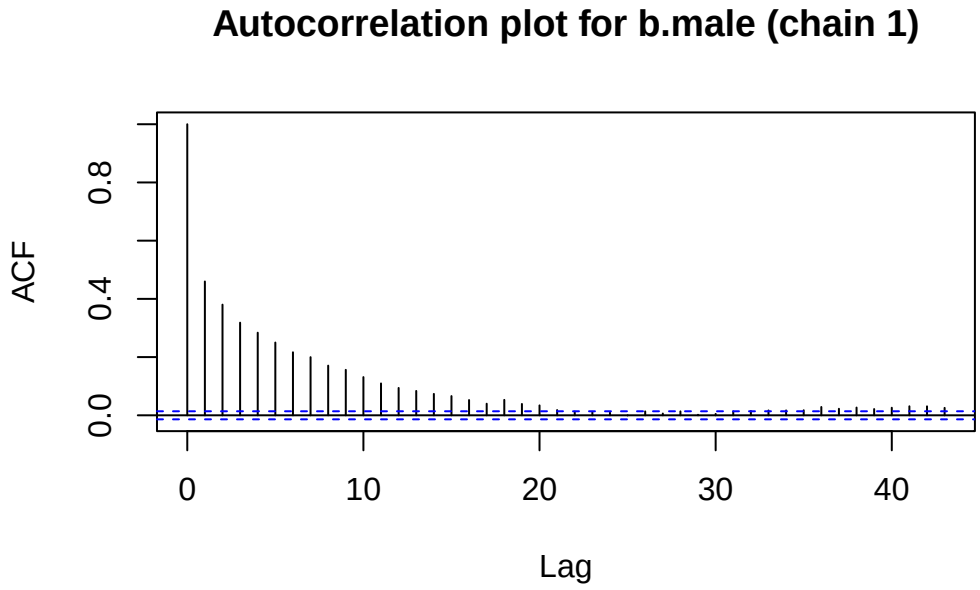
```
#  obesity might be related to smoking or that snoring might be strongly
 ↪  effected by smoking and obesity.
#  In modern epi, these factors might be consider to be in the <<causal
 ↪  path>> and perhaps you might not control for them in this way.
cat(
"smoke  obese  snore male hypoten n
0 0 0 1 5 60
0 0 0 0 10 149
1 0 0 1 2 17
1 0 0 0 6 16
0 1 0 1 1 12
0 1 0 0 2 9
0 0 1 1 36 187
0 0 1 0 28 138
1 0 1 1 13 85
1 0 1 0 4 39
0 1 1 1 15 51
0 1 1 0 11 28
1 1 1 1 8 23
1 1 1 0 4 12
", file= "SmokeHyperData.txt")


SmokeHyper=read.table("SmokeHyperData.txt",header=TRUE,sep = "")
attach(SmokeHyper)



cat("
model{
  for( i in 1:14){
   hypoten[i] ~ dbin(mu[i], n[i])
   logit(mu[i]) <- b0 + b.smok*smoke[i]+ b.ob*obese[i]+ b.sn*snore[i] +
     b.male*male[i] + b.smsn*smoke[i]*snore[i] + b[i]
    b[i] ~dnorm(0, tau.b)
   }
  b.smok ~ dnorm(0, .04) # so, sd =5.  exp(5) ~ 148 which is huge
  b.ob ~ dnorm(0, .04)
  b.sn ~ dnorm(0, .04)
  b.male ~ dnorm(0, .04)
  b0 ~ dnorm(0, .04)
  b.smsn ~dnorm(0, .04)
  sd.b ~ dunif(0, 5)
```

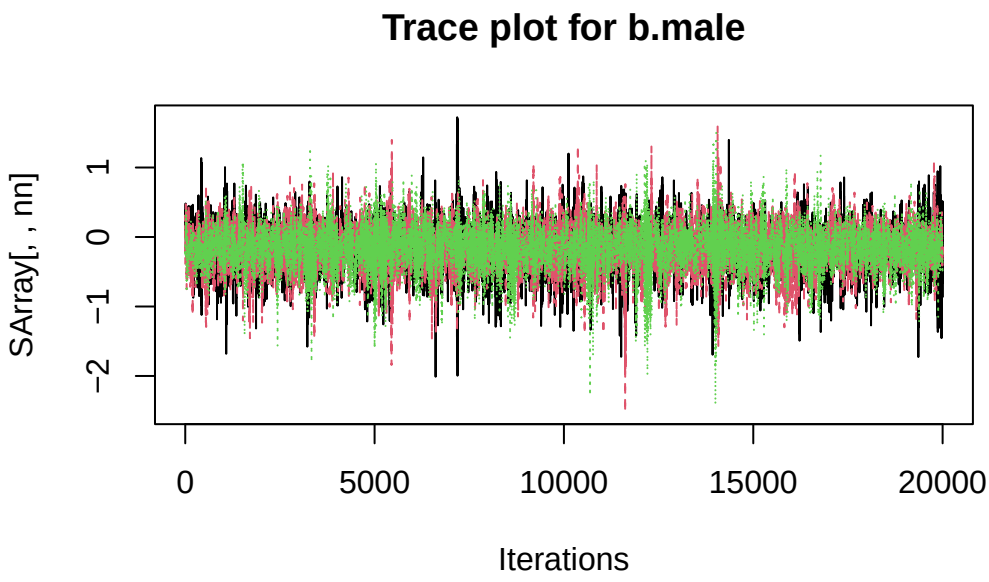
```
  tau.b <- 1/sd.b/sd.b
  }
  ", file="SmokeHyperMod3.txt")


bugM3.dat=list("hypoten", "n", "smoke", "obese", "snore", "male")  # what
 ↪  variable you need in the model

initM3.fun=function(){list(b.smok=runif(1,-.8,-.2),b.male=runif(1,-.8,-.2),
                           sd.b=runif(1,.2,.8))}

paramsM3=c("b.male","b.smok","sd.b")
     ### what variables you want to monitor


#### Could change the code below...
SmokeHypeBaseM3<-jags(bugM3.dat, initM3.fun, paramsM3,
 ↪  model.file="SmokeHyperMod3.txt",
    n.chains=3, n.iter=20000, n.burnin=0, n.thin = 1)
```

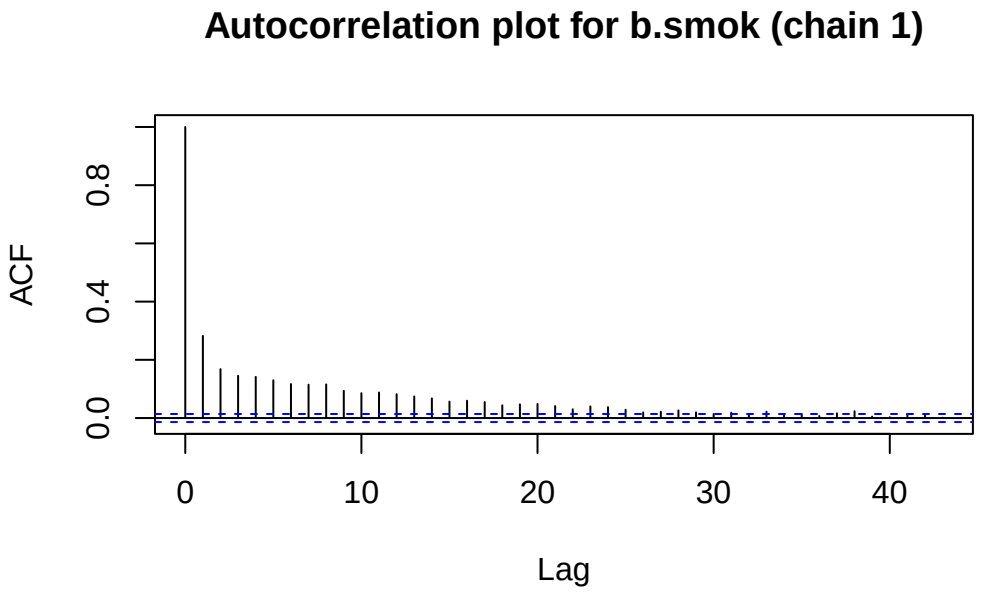
```
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 14
   Unobserved stochastic nodes: 21
   Total graph size: 151

Initializing model
```

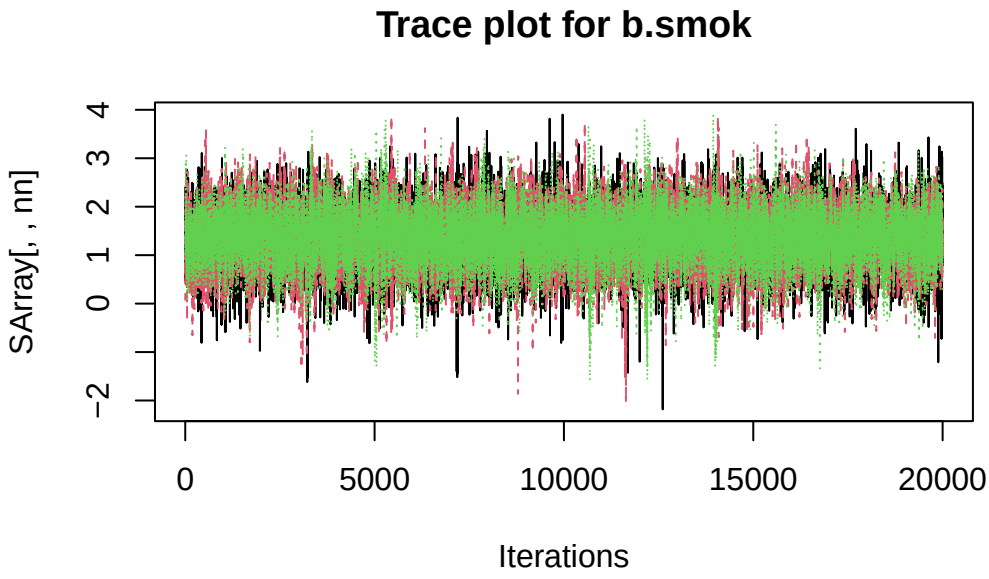
```
if(T){
SArray= SmokeHypeBaseM3$BUGSoutput$sims.array
vname=attr(SArray,"dimnames")[3][[1]]
vname <- vname[vname != "deviance"]
# drop deviance
SArray <- SArray[,,vname]

chainL=attr(SArray,"dim")[1][[1]]
for(i in 1:length(vname)){
    nn=vname[i]
    acf( SArray[,1,nn], main=paste0("Autocorrelation plot for ",nn, " (chain
     ↪  1)"))  #note: this is only for 1st chain
```

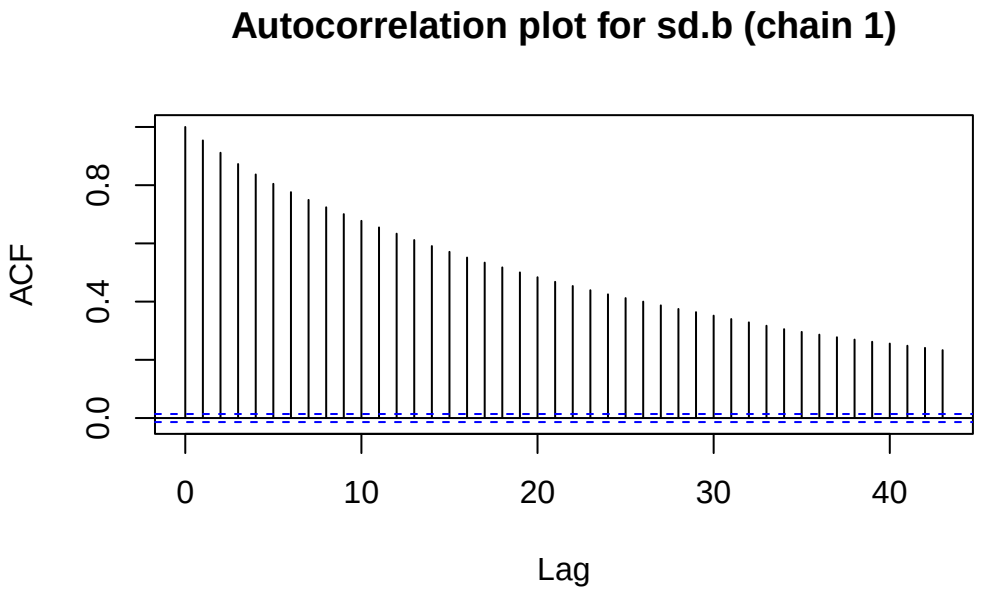
```
    xnul=locator(1)
    matplot(1:chainL,SArray[,,nn], main=paste0("Trace plot for ", nn),
     ↪  xlab="Iterations",type="l")
    xnul=locator(1)
    }
}
```

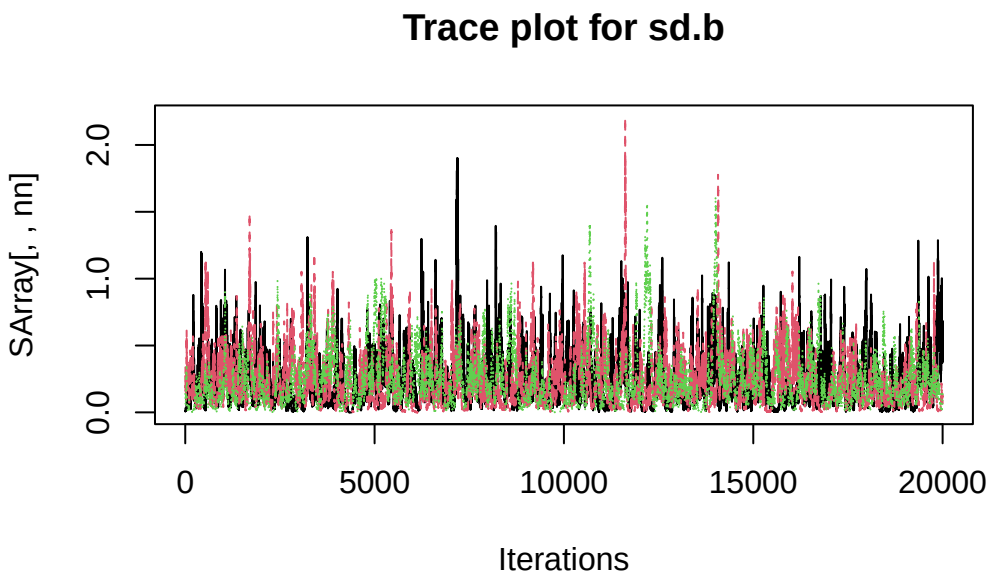



Trace plot for b.male
SArray[, , nn]
−2
−1
0
1
0
5000
10000
15000
20000
Iterations


Autocorrelation plot for b.smok (chain 1)
ACF
0.0
0.4
0.8
0
10
20
30
40
Lag


Trace plot for b.smok
SArray[, , nn]
−2
0
1
2
3
4
0
5000
10000
15000
20000
Iterations


Autocorrelation plot for sd.b (chain 1)
ACF
0.0
0.4
0.8
0
10
20
30
40
Lag




```
# Statistics
signif(SmokeHypeBaseM3$BUGSoutput$summary[paramsM3, ],4) %>% kable(format =
 ↪  "html", caption = "Statistics of sd.b, b.smok and b.male")
```

Table 1: Statistics of sd.b, b.smok and b.male

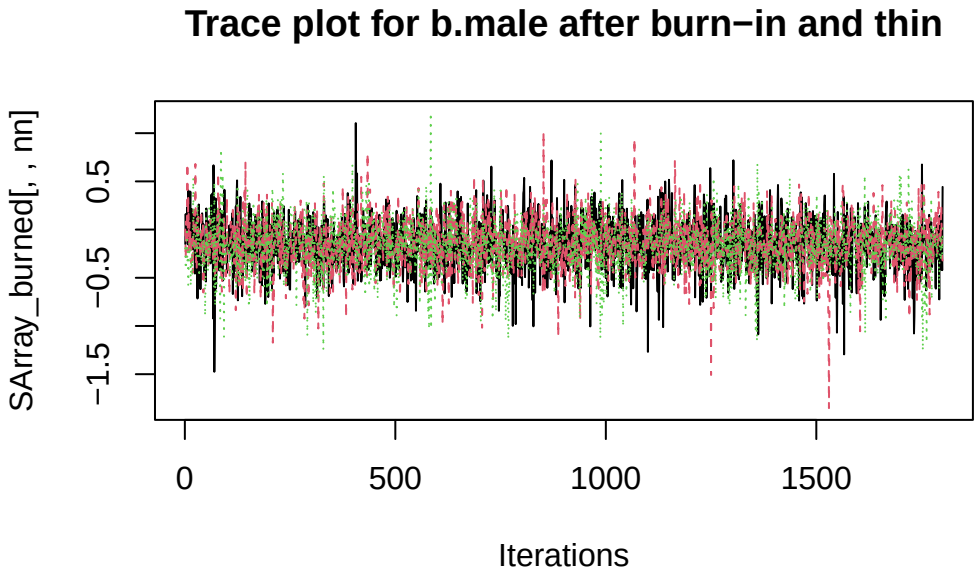
	mean	sd	2.5%	25%	50%	75%	97.5%	Rhat	n.eff
b.male	-0.1605	0.2707	-0.720500	-0.31830	-0.1537	0.006522	0.3535	1.001	6700
b.smok	1.3820	0.5571	0.264600	1.02500	1.3910	1.751000	2.4450	1.001	22000
sd.b	0.2259	0.2030	0.008511	0.07831	0.1725	0.314500	0.7497	1.003	1000

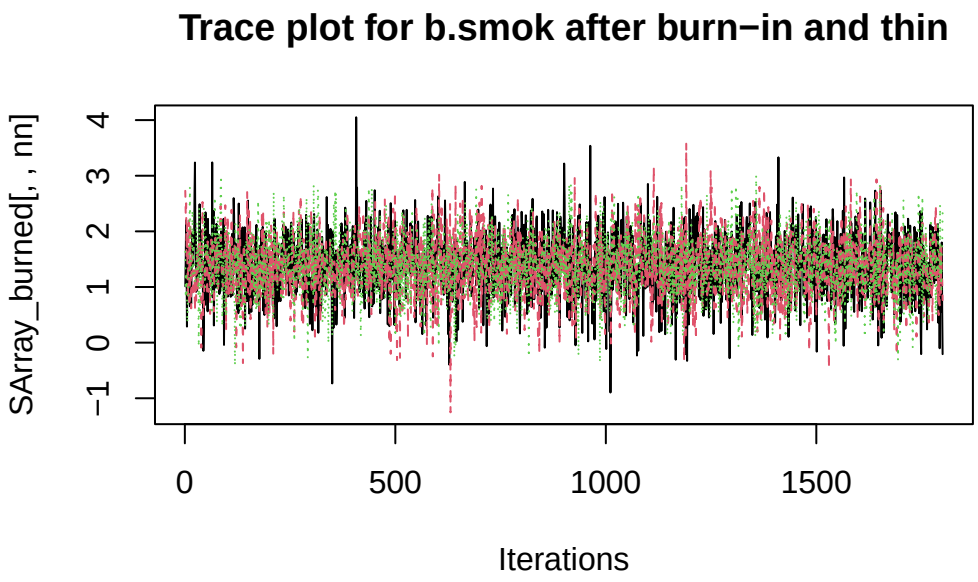
(b) Input the MCMC values into R (but not yet into coda). Plot the trace plot in R. Remove some of the early values of the chain (throwing away a part that is "burned-in") and then plot the estimate of the densities for the parameters with a different density estimate for each of the three chains.

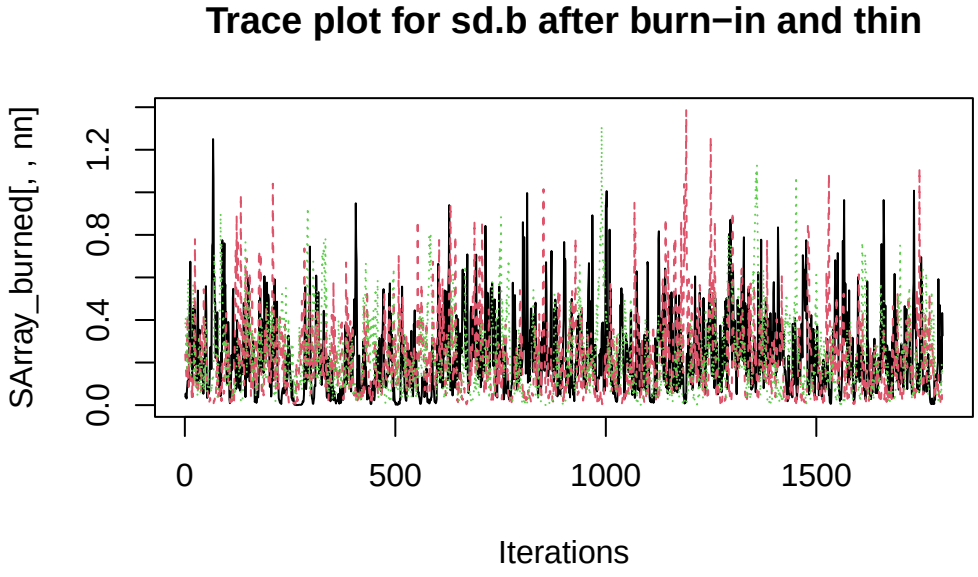
```
# burn(discard) in first 2000 iterations with thinning of 10
SmokeHypeBaseM3<-jags(bugM3.dat, initM3.fun, paramsM3,
 ↪  model.file="SmokeHyperMod3.txt",
    n.chains=3, n.iter=20000, n.burnin=2000, n.thin = 10)
```

```
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 14
   Unobserved stochastic nodes: 21
   Total graph size: 151

Initializing model
```

```
SArray_burned= SmokeHypeBaseM3$BUGSoutput$sims.array
vname=attr(SArray_burned,"dimnames")[3][[1]]

vname <- vname[vname != "deviance"]
SArray_burned <- SArray_burned[,,vname]

chainL=attr(SArray_burned,"dim")[1][[1]]
for(i in 1:length(vname)){
    nn=vname[i]
    matplot(1:chainL,SArray_burned[,,nn], main=paste0("Trace plot for ", nn,"
     ↪  after burn-in and thin"),xlab="Iterations",type="l")
    xnul=locator(1)
    }
```

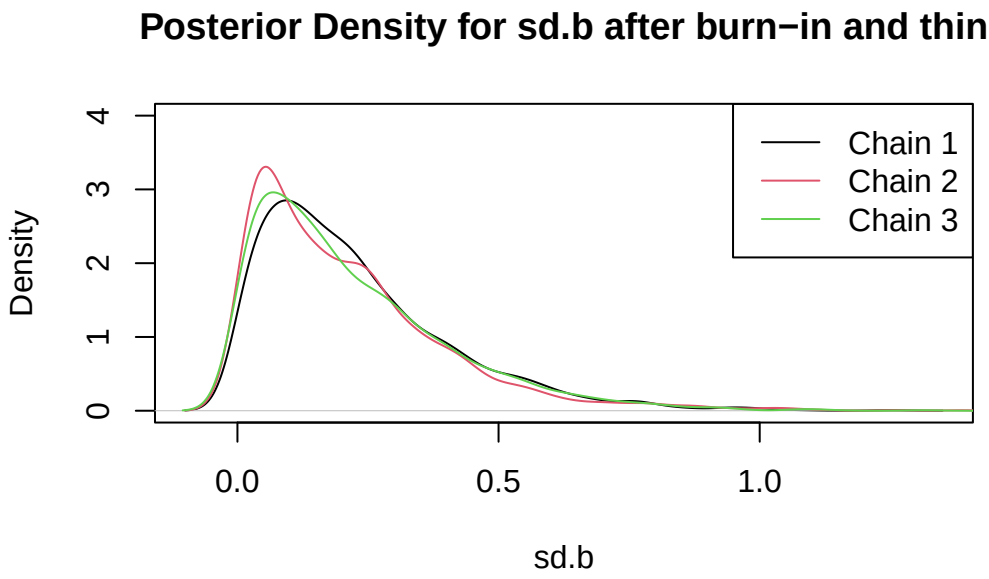




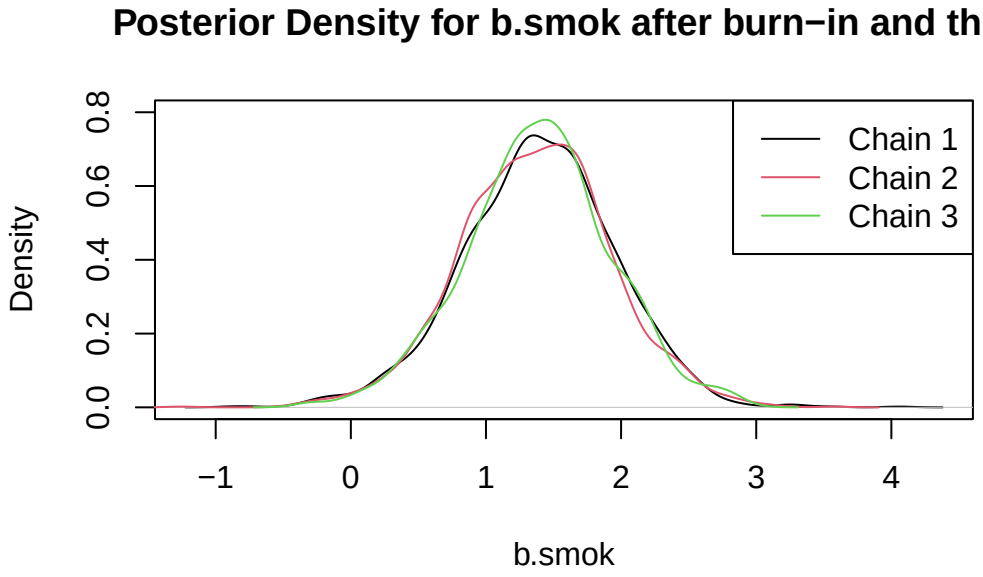



```
# Plot: sd.b
plot(density(SArray_burned[,1,"sd.b"]), col=1, xlab="sd.b",
     main="Posterior Density for sd.b after burn-in and thin",
```

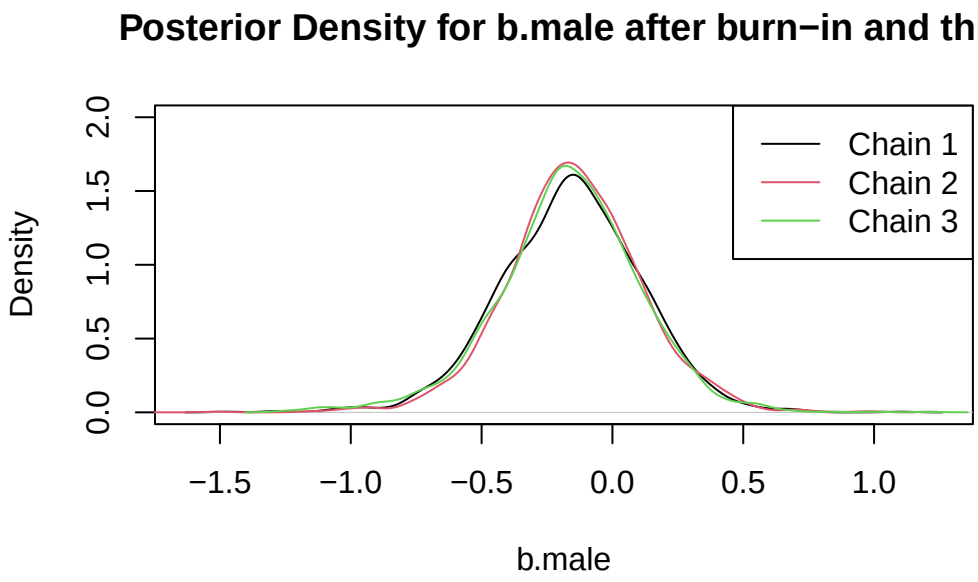
```
     ylim=c(0, 4))
lines(density(SArray_burned[,2,"sd.b"]), col=2)
lines(density(SArray_burned[,3,"sd.b"]), col=3)
legend("topright", legend=paste("Chain", c(1:3)), col=c(1,2,3), lwd=1)
```




```
# Plot: b.smok
plot(density(SArray_burned[,1,"b.smok"]), col=1, xlab="b.smok",
     main="Posterior Density for b.smok after burn-in and thin",
     ylim=c(0, 0.8))
lines(density(SArray_burned[,2,"b.smok"]), col=2)
lines(density(SArray_burned[,3,"b.smok"]), col=3)
legend("topright", legend=paste("Chain", c(1:3)), col=c(1,2,3), lwd=1)
```



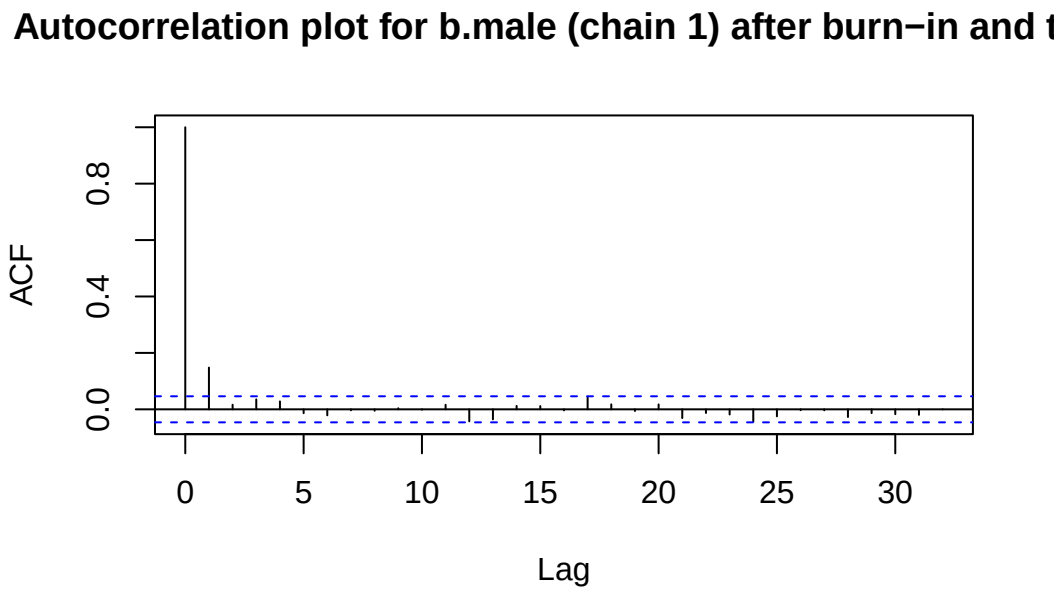

```
# Plot: b.male
plot(density(SArray_burned[,1,"b.male"]), col=1, xlab="b.male",
     main="Posterior Density for b.male after burn-in and thin",
     ylim=c(0, 2))
lines(density(SArray_burned[,2,"b.male"]), col=2)
lines(density(SArray_burned[,3,"b.male"]), col=3)
legend("topright", legend=paste("Chain", c(1:3)), col=c(1,2,3), lwd=1)
```

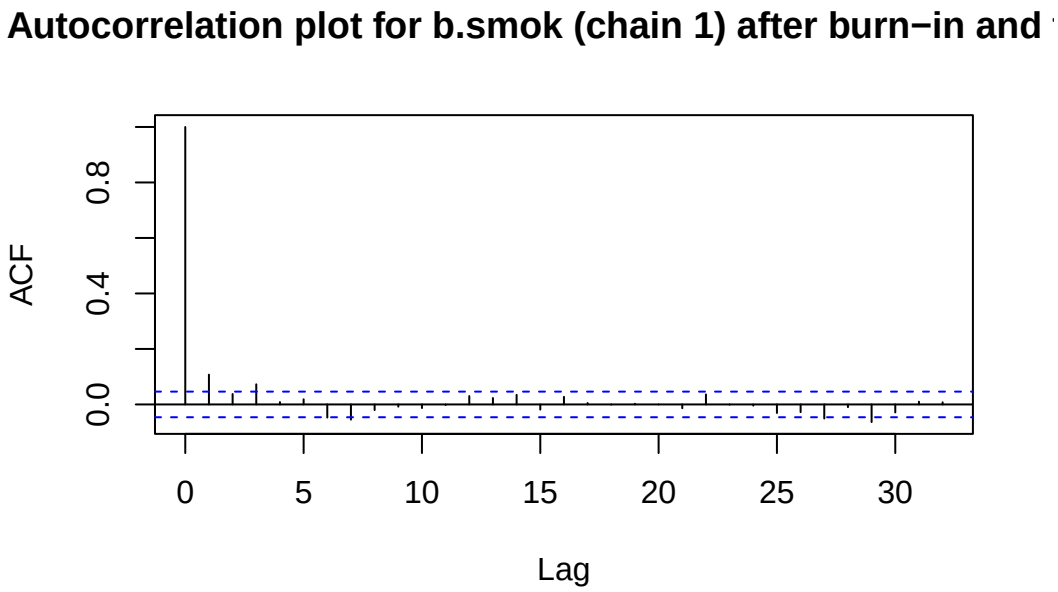


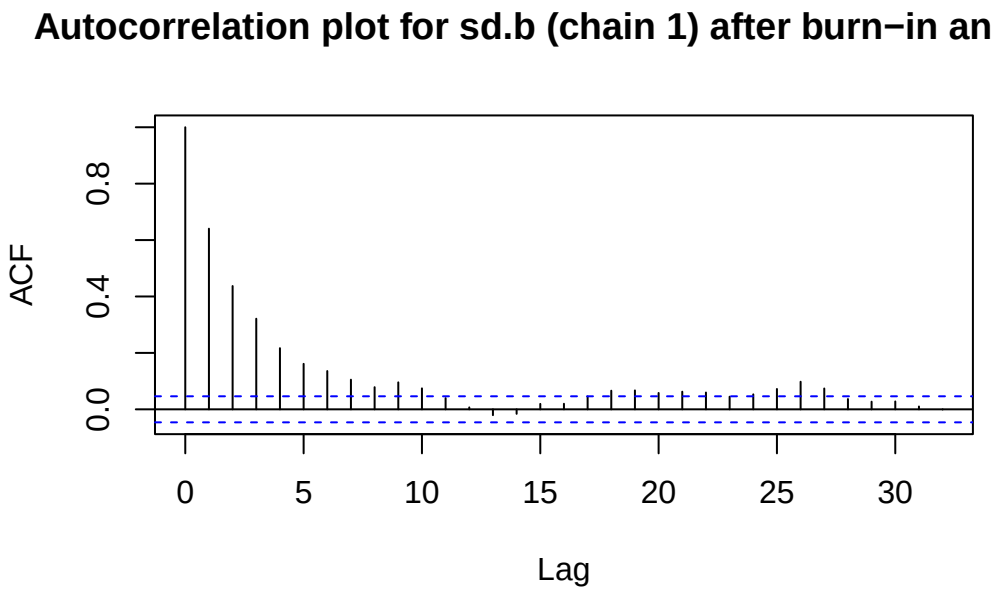
Burning in a chain means discarding chosen number of initial iterations (samples) from your MCMC chain results. This is helpful because the early samples depend heavily on the starting (initial) values and may not reflect the true posterior distribution. By burning in, you discard the early samples and prevents them from influencing the final results. This helps getting more accurate estimates of the posterior distribution.

(c) If you thinned the chain, what would be the advantages? Is it necessary to thin a chain?

```
for(i in 1:length(vname)){
    nn=vname[i]
    acf( SArray_burned[,1,nn], main=paste0("Autocorrelation plot for ",nn, "
     ↪  (chain 1) after burn-in and thin"))  #note: this is only for 1st
     ↪  chain
    xnul=locator(1)
    }
```


Autocorrelation plot for b.male (chain 1) after burn−in and t
ACF
0.0
0.4
0.8
0
5
10
15
20
25
30
Lag


Autocorrelation plot for b.smok (chain 1) after burn−in and
ACF
0.0
0.4
0.8
0
5
10
15
20
25
30
Lag



Thinning the chain means that we only keep every kth sample from the result. For example, if we set `n.thin=10` in question 1 part b with no burning in, we keep only 2000 samples per variable per chain. Thinning reduces the autocorrelation between the samples and make the retained samples less correlated. The advantage of thinning is that it reduces the autocorrelation significantly and can make the chains appear to mix faster. However, thinning may reduce the precision of the result, since it discards some samples.

The "Autocorrelation plot for sd.b (chain 1)" shows that autocorrelation did not converge to 0 when `n.iter=20000` with no thinning. This means that many samples are highly correlated. So here, thinning the chain would be necessary to reduce autocorrelation. As you can see from the "Autocorrelation plot for sd.b (chain 1) after burn-in and thin" plot, the autocorrelation is reduced after burning in and thinning.

(d) Provide the estimate of the posterior mean of the three parameters for each chain and also give the Monte Carlo accuracy of your estimate. For the Monte Carlo accuracy, compute by batch means and by using the autocorrelation function.

```
# Posterior mean

means_per_chain <- sapply(paramsM3, function(param){
  sapply(1:3, function(chain){signif(mean(SArray_burned[, chain, param]),4)})
```

```
})

rownames(means_per_chain) <- paste("Chain", 1:3)
means_per_chain %>% kable(format = "html", caption = "Posterior mean of the
 ↪  three parameters for each chain")
```

Table 2: Posterior mean of the three parameters for each chain

	b.male	b.smok	sd.b
Chain 1	-0.1578	1.392	0.2261
Chain 2	-0.1467	1.368	0.2097
Chain 3	-0.1644	1.390	0.2178

```
# Monte Carlo accuracy

# Standard error: via Batch Means
calcBM=function(x,Batn=50){
BigN=length(x)
BatInc=ceiling( (1:BigN)/(BigN/Batn) )
BM=tapply(x,BatInc,mean)
list(MCE=(sd(BM)/sqrt(length(BM))), BM=BM)}

BatchSE <- sapply(paramsM3, function(param){

   ↪  sapply(1:3,function(chain)signif(calcBM(SArray_burned[,chain,param])$MCE,4))
})

rownames(BatchSE) <- paste("Chain", 1:3)

BatchSE%>%kable(format="html",caption="Monte Carlo accuracy: SE via Batch
 ↪  Means")
```

Table 3: Monte Carlo accuracy: SE via Batch Means

	b.male	b.smok	sd.b
Chain 1	0.006858	0.01502	0.009821
Chain 2	0.007660	0.01338	0.011200
Chain 3	0.008127	0.01574	0.012040

```
# Note: the above can also be run via a coda command:
mcmc_chain1 <- SArray_burned[, 1, ]
r1 <- batchSE(mcmc(mcmc_chain1))

mcmc_chain2 <- SArray_burned[, 2, ]
r2 <- batchSE(mcmc(mcmc_chain2))

mcmc_chain3 <- SArray_burned[, 3, ]
r3 <-batchSE(mcmc(mcmc_chain3))

BatchSE_coda <- signif(rbind(r1, r2, r3),4)
rownames(BatchSE_coda) <- paste("Chain", 1:3)
BatchSE_coda %>% kable(format = "html", caption = "Monte Carlo accuracy: SE
 ↪  via Batch Means (coda)")
```

Table 4: Monte Carlo accuracy: SE via Batch Means (coda)

	b.male	b.smok	sd.b
Chain 1	0.007249	0.01606	0.01299
Chain 2	0.006531	0.01598	0.01311
Chain 3	0.007658	0.01466	0.01321

```
# Autocorrelation function
CalcAcSe=function(x,lag.max=50){
autoc=(acf(x,lag.max=lag.max,plot=FALSE))$acf
sd(x)/sqrt(length(x))*sqrt(-1+2*sum(autoc))}

AutocSE=sapply(paramsM3, function(param){
  sapply(1:3, function(chain) signif(CalcAcSe(SArray_burned[, chain, param]),
   ↪  4))
})

rownames(AutocSE) <- paste("Chain", 1:3)

AutocSE %>% kable(format = "html", caption = "Monte Carlo accuracy: SE via
 ↪  Autocorrelation function")
```

Table 5: Monte Carlo accuracy: SE via Autocorrelation function

	b.male	b.smok	sd.b
Chain 1	0.007257	0.01283	0.01114
Chain 2	0.008315	0.01186	0.01261
Chain 3	0.008258	0.01452	0.01226

The Monte Carlo standard errors (MCSE), computed via both batch means and autocorrelation function, were very close to 0, which indicates that we ran the chains long enough to get a stable, reliable estimate.

(e) Using the coda (or boa) package, use the Geweke and Brooks-Gelman-Rubin diagnostic procedures to assess how well the MCMC algorithm has converged.

```
library(coda)

# mcmc list for each chain
mcmc_all <- mcmc.list(mcmc(mcmc_chain1), mcmc(mcmc_chain2),
 ↪  mcmc(mcmc_chain3))

# Geweke diagnostic
geweke.diag(mcmc_all)
```

```
[[1]]

Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

b.male b.smok   sd.b
1.6166 0.7713 0.4069


[[2]]

Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

  b.male   b.smok     sd.b
-0.04328 -0.21016  0.15060
```

```
[[3]]

Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

  b.male   b.smok     sd.b
-0.44656 -0.09489  0.46613
```

The geweke's Z scores for the three parameters across all three chains are all within the range of -2 to 2, which does not raise any warning flags regarding convergence.

```
# Brooks-Gelman-Rubin diagnostic
gelman.diag(mcmc_all, autoburnin = FALSE)
```

```
Potential scale reduction factors:

       Point est. Upper C.I.
b.male          1          1
b.smok          1          1
sd.b            1          1

Multivariate psrf

1
```

The Brooks–Gelman–Rubin diagnostic was used to assess convergence across the three MCMC chains. For each parameter, the potential scale reduction factor (PSRF) was exactly 1.00 or lower than 1.2 (critical value) for both the point estimate and the upper confidence limit. Additionally, the multivariate PSRF was 1.00. These results indicate excellent convergence (under the influence of the limiting distribution), with no evidence of between-chain variation.

(f) Using the information from this question, state if you feel that the MCMC algorithm has converged. Justify your answer.

From 1(a), trace plots showed that all three chains for each parameter were well mixed, with no trends across iterations evem before burning in.

From 1(b), autocorrelation plots didn't not decay to 0 for sd.b, but the autocorrelation decayed reasonably. From 1(c), introduction of thinning reduced the autocorrelation significantly.

From 1(d), the Monte Carlo standard errors (MCSE), computed via both batch means and autocorrelation function, were close to 0, which indicates that we ran the chains long enough to get a stable, reliable estimate.

From 1(e), the geweke diagnostic z-scores for all parameters for all three chains were within the acceptable range (–2 to 2), which indicates excellent convergence.

Finally, the Brooks–Gelman–Rubin diagnostic (PSRF) were exactly 1.00 or below the critical value (1.2) for all parameters, with a multivariate PSRF of 1.00. This procides a strong evidence that the chains have fully converged to the target distribution.

These results strongly suggest that the MCMC algorithm has converged successfully.

Question 2

```
# Given data and models

q2data <- list( x=c(16,18,20,22,24,26,28,30,32,34,36,38,40,42,44,46),
y=c(2508,2518,3304,3423,3057,3190,3500,3883,3823,3646,3708,
3333,3517,3241,3103,2776))

cat("model{
for(i in 1:16){
  y[i]~dnorm(mu[i],tau)
  mu[i]<- b[1] + b[2]*(x[i]-31)
}
b[1]~dnorm(0,.000001)
b[2]~dnorm(0,.000001)
tau~dgamma(.0001,.0001)
}", file = "model1.txt")

cat("model{
for(i in 1:16){
  y[i]~dnorm(mu[i],tau)
  mu[i]<- b[1] + b[2]*(x[i]-31)+ b[3]*pow((x[i]-31),2)
}
b[1]~dnorm(0,.000001)
b[2]~dnorm(0,.000001)
b[3]~dnorm(0,.01)
tau~dgamma(.0001,.0001)
}",file = "model2.txt")
```

Do the following two parts:

(a)

Compare these two models. First, compare these two models by looking at the "deviance" measures and the DIC. Calculate these values for each model and comment on them.

Then, compare these two models by calculating the Bayes Factor. To calculate the Bayes factor, run an MCMC algorithm which switches between the two models using a method similar (in which you might have to somewhat change the model code as well as the model) to the method proposed by Kuo and Mallick. Comment on your belief between the two models. (That is, which model do you prefer and justify your answer.)

```
# Model 1
init1 <- function(){ list(b=rnorm(2, 0, 1), tau=runif(1,1,2))}

fit1 <- jags(data = q2data, inits = init1, parameters.to.save = c("b",
 ↪  "tau"),
             model.file = "model1.txt", n.chains = 3, n.iter = 10000,
             n.burnin = 2000, n.thin = 1)
```

```
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 16
   Unobserved stochastic nodes: 3
   Total graph size: 87

Initializing model
```

```
x1 <- cbind(rbind(quantile(fit1$BUGSoutput$sims.list$deviance,
                           probs=c(0.025,.25,.5,.75,.975))),
            mean=mean(fit1$BUGSoutput$sims.list$deviance),
            DIC=fit1$BUGSoutput$DIC)

# Model 2
init2 <- function(){ list(b=rnorm(3, 0, 1), tau=runif(1,1,2))}
fit2 <- jags(data = q2data, inits = init2, parameters.to.save = c("b",
 ↪  "tau"),
             model.file = "model2.txt", n.chains = 3, n.iter = 10000,
             n.burnin = 2000, n.thin = 1)
```

```
Compiling model graph
   Resolving undeclared variables
```

```
   Allocating nodes
Graph information:
   Observed stochastic nodes: 16
   Unobserved stochastic nodes: 4
   Total graph size: 114

Initializing model
```

```
x2 <- cbind(rbind(quantile(fit2$BUGSoutput$sims.list$deviance,
                           probs=c(0.025,.25,.5,.75,.975))),
            mean=mean(fit2$BUGSoutput$sims.list$deviance),
            DIC=fit2$BUGSoutput$DIC)

temp2 <- rbind(x1,x2)
rownames(temp2)=c("Model 1","Model 2")

temp2 %>% kable(format = "html", caption = "Quantiles of deviance and DIC for
 ↪  each model")
```

Table 6: Quantiles of deviance and DIC for each model

	2.5%	25%	50%	75%	97.5%	mean	DIC
Model 1	236.4911	237.6193	238.9343	240.9234	247.0786	239.6798	243.7710
Model 2	212.8164	214.4866	216.1521	218.5185	225.3524	216.9211	222.4457

From the quantiles of deviance and DIC for each model table, we can observe that Model 2 has a lower mean deviance and a lower DIC compared to Model 1. Deviance measures the model fit (lower the better) and DIC measures model fit while accounting for parsimony (also lower the better), therefore Model 2 with both lower mean deviance and DIC is a better fit for the data than Model 1.

Using Kuo and Mallick approach, the Bayes Factor is

$$BF_{21} = \frac{P(Model\ 2 \mid data)}{P(Model\ 1 \mid data)} = \frac{P(del[1] = 1 \mid data)}{P(del[1] = 0 \mid data)} = \frac{mean(del[1])}{1 - mean(del[1])}$$

```
# calculate the Bayes factor
cat("
model{
  for(i in 1:16){
    y[i]~dnorm(mu[i],tau)
```

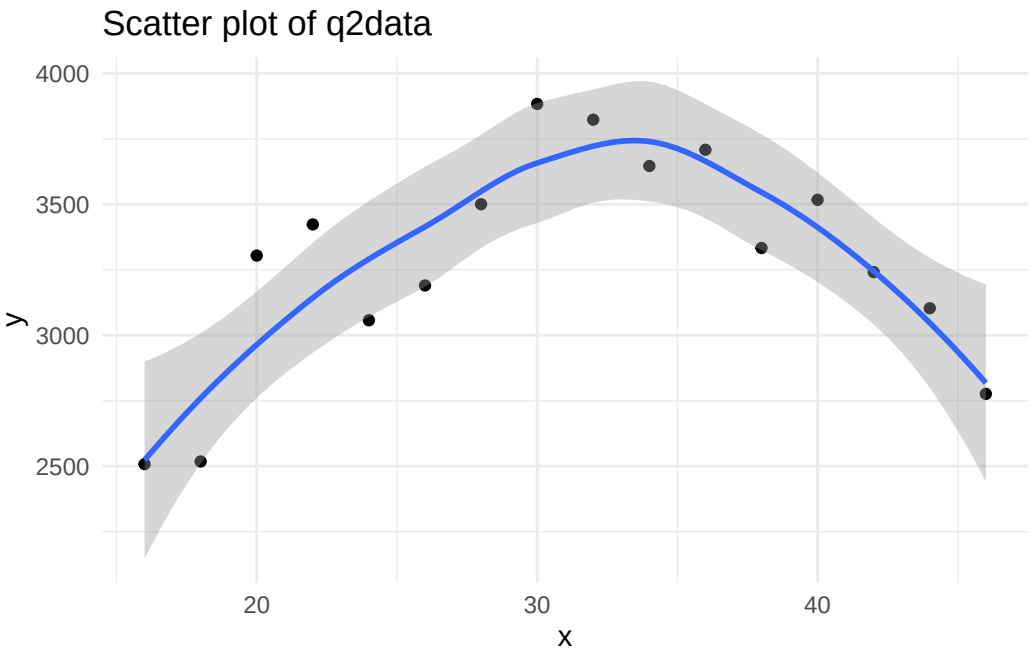
```
    # del switch between two models
    mu[i]<- b[1] + b[2]*(x[i]-31)+ del[1]*b[3]*pow((x[i]-31),2)
}

b[1]~dnorm(0,.000001)
b[2]~dnorm(0,.000001)
b[3]~dnorm(0,.01)
tau~dgamma(.0001,.0001)

del[1]~dbern(0.5)
}

", file="model_BF.txt")


q2data <- list(
  x = c(16,18,20,22,24,26,28,30,32,34,36,38,40,42,44,46),
  y = c(2508,2518,3304,3423,3057,3190,3500,3883,3823,3646,
        3708,3333,3517,3241,3103,2776)
)

parameters<-c("b","tau","del[1]")

model_BF<-jags(data=q2data, inits=init2, parameters.to.save=parameters,
               model.file="model_BF.txt", n.chains=3, n.iter=10000,
               ↪  n.burnin=2000, n.thin=10)
```

```
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 16
   Unobserved stochastic nodes: 5
   Total graph size: 116

Initializing model
```

```
p2 <- mean(model_BF$BUGSoutput$sims.list$del==1) # model 2
p1 <- 1-p2 #model 1

BF <- p2/p1; cat("Bayes Factor (Model 2 vs Model 1): ", BF, "\n")
```

```
Bayes Factor (Model 2 vs Model 1):  799
```

```
# plot q2data
library(ggplot2)
data.frame(q2data) %>% ggplot(aes(x=x,y=y))+geom_point()+geom_smooth()+
  theme_minimal()+labs(title="Scatter plot of q2data", x="x", y="y")
```



The Bayes Factor is very large, which indicates that Model 2 (Quadratic) is strongly preferred over Model 1 (linear). This is expected since the scatter plot of `q2data` shows a strong quadratic relationship between `x` and `y`.

(b) For model 1, look at the residuals for the model using functions 1, 2, and 3. That is, calculate 1) the residuals, 2) the standardized residuals, and 3) the chance of getting a more extreme observation. For the residual and the standardize residual, please calculate the distribution of these statistics under the predictive distribution. Comment on the results of these statistics for the observation. Also, comment on how well you think the model fits the data.

```
cat("model{
for(i in 1:16){
  y[i]~dnorm(mu[i],tau)
```

```
  mu[i]<- b[1] + b[2]*(x[i]-31)
  res[i]<-(y[i]-mu[i])                 #  estimate of the residuals for this
 ↪  model
  stdres[i]<-res[i]*sqrt(tau)        #  for the standardized residuals

  dev1.obs[i]<-pow(res[i],2)
  dev2.obs[i]<-pow(stdres[i],2)

  #  getting a replicated sample..... This is a sample of the predictive
 ↪  distribution

  y.rep[i]~dnorm(mu[i],tau)
  p.smaller[i] <-step(y[i]-y.rep[i])       # check to see the probability of
 ↪  getting a more extreme value

  #  residual and moments of replicated data....   this gives the predicted
 ↪  distribution for these values.
  res.rep[i]<- y.rep[i] - mu[i]
  stdres.rep[i]<- res.rep[i]*sqrt(tau)

  dev1.rep[i]<-pow(res.rep[i],2)
  dev2.rep[i]<-pow(stdres.rep[i],2)
}
b[1]~dnorm(0,.000001)
b[2]~dnorm(0,.000001)
tau~dgamma(.0001,.0001)


#     summing the diagnostic values

chidev1.obs <- sum(dev1.obs[])
chidev2.obs <- sum(dev2.obs[])

chidev1.rep <- sum( dev1.rep[] )
chidev2.rep <- sum( dev2.rep[] )

chidev1.pval<-step(chidev1.obs-chidev1.rep)
chidev2.pval<-step(chidev2.obs-chidev2.rep)
}", file = "model1b.txt")

## getting the residuals and the calibrations:
parameters<-c("b","res", "stdres", "res.rep", "stdres.rep", "p.smaller",
 ↪  "chidev1.pval", "chidev2.pval", "chidev1.obs", "chidev2.obs",
```

```
  "chidev1.rep", "chidev2.rep", "dev", "dev.rep", "dev.pval")


AnscombeLin.sim<-jags(data=q2data, inits=init1,
 ↪  parameters.to.save=parameters,
  model.file="model1b.txt", n.chains=3, n.iter=10000, n.burnin=2000,
   ↪  n.thin=10)
```

```
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 16
   Unobserved stochastic nodes: 19
   Total graph size: 277

Initializing model
```

```
xxx<-AnscombeLin.sim$BUGSoutput

temp<-cbind(xxx$mean$res,t(apply(xxx$sims.list$res.rep,2,function(x)
 ↪  {c(quantile(x,probs=c(0.025,.975)),mean(x),sd(x))})))
colnames(temp)=c("res","2.5%","97.5%","mean","SD");temp %>% kable(format =
 ↪  "html", caption = "Residuals")
```

Table 7: Residuals

res	2.5%	97.5%	mean	SD
-551.255767	-888.8042	934.5745	8.732000	464.2886
-565.719186	-885.9555	866.9548	-15.015027	446.9756
195.817396	-839.3966	900.2413	-6.800166	448.0965
290.353977	-925.2128	885.7221	-11.626821	457.9899
-100.109442	-877.4350	894.6170	8.691453	452.3818
8.427139	-915.4977	859.1558	7.086164	449.1526
293.963721	-904.2005	907.5989	15.043110	461.7396
652.500302	-906.6085	891.6851	-4.121572	452.4815
568.036883	-911.0672	896.7475	-4.376902	448.9843
366.573464	-874.2753	907.9423	4.360466	455.6137
404.110046	-907.9685	847.7244	-21.091011	447.4680
4.646627	-890.4939	888.0864	6.623796	456.8105
164.183208	-893.0598	920.3858	11.138429	451.4364

res	2.5%	97.5%	mean	SD
-136.280211	-883.0536	856.6848	-13.333462	445.6921
-298.743630	-924.9426	875.4092	-15.133098	451.5812
-650.207048	-909.8091	891.6592	-9.635183	455.2491

```
temp<-cbind(xxx$mean$stdres,t(apply(xxx$sims.list$stdres.rep,2,function(x)
 ↪  {c(quantile(x,probs=c(0.025,.975)),mean(x),sd(x))})))
colnames(temp)=c("stdres","2.5%","97.5%","mean","SD");temp %>% kable(format =
 ↪  "html", caption = "Standardized residuals")
```

Table 8: Standardized residuals

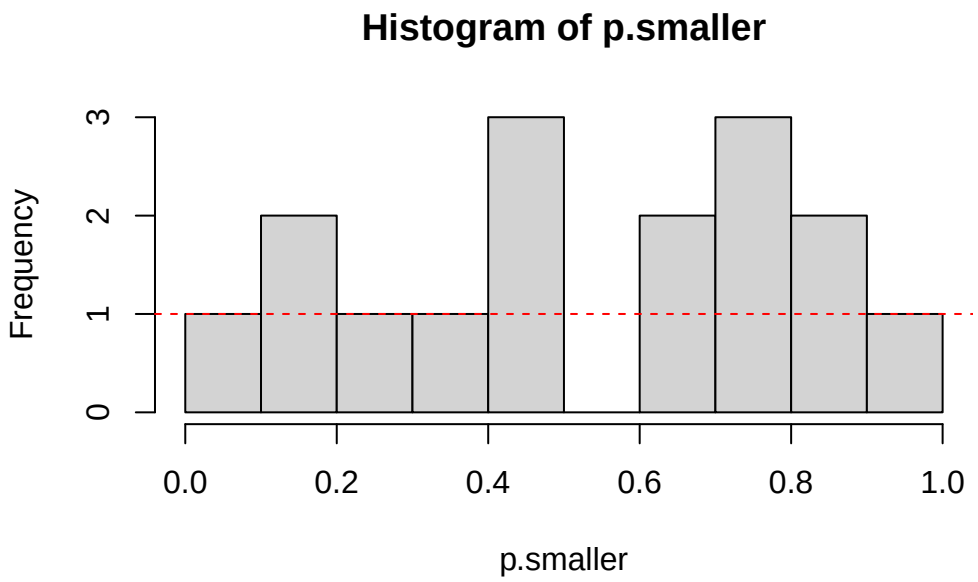
stdres	2.5%	97.5%	mean	SD
-1.3076083	-1.935113	2.038523	0.0243122	1.0133386
-1.3412153	-1.982130	1.888268	-0.0399713	0.9891102
0.4513580	-1.922645	2.065325	-0.0229734	1.0093851
0.6742634	-1.968664	1.973292	-0.0208429	0.9991840
-0.2441938	-1.899190	1.929135	0.0168837	0.9874030
0.0116582	-2.008293	1.947323	0.0162553	1.0002777
0.6840487	-2.005493	1.998652	0.0378445	1.0216006
1.5282319	-1.979244	1.993929	-0.0052849	1.0112545
1.3298922	-1.969301	1.928359	-0.0115160	0.9985643
0.8562134	-1.936171	1.978391	0.0095596	1.0068413
0.9449793	-2.008079	1.863606	-0.0410007	0.9931721
0.0053421	-1.978284	1.963677	0.0156371	1.0051218
0.3812137	-1.990201	1.967423	0.0168075	0.9911890
-0.3254443	-1.937181	1.914395	-0.0312388	0.9855964
-0.7073434	-2.030110	1.953982	-0.0283354	1.0119760
-1.5340209	-2.018347	2.003375	-0.0265288	1.0104413

```
# chance of getting a more extreme observation

temp<-cbind(xxx$mean$p.smaller,t(apply(xxx$sims.list$p.smaller,2,function(x)
 ↪  {c(quantile(x,probs=c(0.025,0.5,.975)),sd(x))})))
colnames(temp)=c("mean","2.5%","50%","97.5%","SD");temp %>% kable(format =
 ↪  "html", caption = "Chance of getting a more extreme observation")
```

Table 9: Chance of getting a more extreme observation

mean	2.5%	50%	97.5%	SD
0.1254167	0	0	1	0.3312598
0.1200000	0	0	1	0.3250293
0.6679167	0	1	1	0.4710589
0.7433333	0	1	1	0.4368849
0.4008333	0	0	1	0.4901694
0.4879167	0	0	1	0.4999581
0.7287500	0	1	1	0.4446975
0.9241667	0	1	1	0.2647864
0.8937500	0	1	1	0.3082215
0.7812500	0	1	1	0.4134848
0.8270833	0	1	1	0.3782540
0.4991667	0	0	1	0.5001035
0.6358333	0	1	1	0.4812960
0.3866667	0	0	1	0.4870877
0.2679167	0	0	1	0.4429662
0.0958333	0	0	1	0.2944239

```
# Histogram of p.smaller
hist(colMeans(xxx$sims.list$p.smaller), breaks = 10, main = "Histogram of
 ↪  p.smaller", xlab = "p.smaller")
abline(h = 1, col = "red", lty = 2)
```



From the residuals and standardized residuals tables, the observed residuals are well within the 95% credible interval of the replicated residuals. And the standardized residuals are also well within the 95% credible interval of the replicated standardized residuals. The replicated residuals have means around 0 with standard deviation around 1, which is expected for a well fitted standard normal model.

From the chance of getting a more extreme observation table, mean values for rows 5,6 and 12 are close to 0.5 which is ideal. The mean values for rows 8-11 are close to 1, which indicates that these observed values are extreme. The mean values for rows 1,2 and 16 are close to 0, which indicates that observed values are lower than predicted. In an ideal situation where the data is well fitted by the model (i.e. no major discrepancies between observed and replicated values), the distribution of `p.smaller[i]` values should be fairly uniform and centered around 0.5. But as shown in the histogram of `p.smaller`, the distribution is left skewed with a small clustering near 0.15 and bigger clustering near 0.75, which indicates that the model may not be a perfect fit for the data.

Question 3

(a)

$Z = \frac{U_1+U_2}{2}$ where $U_1 \sim uniform(0,1)$ and $U_2 \sim uniform(0,1)$. Then Z is a random sample from the triangle distribution and has density function g.

```r
set.seed(0)

N=500 # 500 each for each Ui
U1=runif(N)
U2=runif(N)
Z=(U1+U2)/2

ex <- mean(Z)
var <- var(Z)

cat("Estimated E[X]:", ex, "\nEstimated Var[X]:", var, "\n")
```

```
Estimated E[X]: 0.5006092
Estimated Var[X]: 0.04340092
```

(b)

Just using sample from a uniform distribution on [0, 1], use the importance sampler method. That is, do the following:

i. Provide the weight function for the importance sampler when using sampled values from the uniform distribution.

$$\text{when } 0 \le x \le 1,\ w(x) = \frac{g(x)}{f(x)} = g(x) = \begin{cases} 4x, & \text{if } 0 \le x \le 0.5, \\ 4(1-x), & \text{if } 0.5 < x \le 1. \end{cases}$$

ii. Give the estimates for E(X) and Var(X). (Note: $Var(X) = (E(X^2) - [E(X)]^2)$)

For some function of X, h(X), the expectation of h(X) under the distribution of g is:

$$\begin{aligned} E_g[h(X)] &= \int h(x)g(x)dx = \int h(x)\frac{g(x)}{f(x)}f(x)dx \\ &= E_f[h(X)\frac{g(X)}{f(X)}] = E_f[h(X)w(X)] \\ &\approx \frac{1}{N}\sum_i (h(x_i)w(x_i)) \end{aligned}$$

Then, estimate of $E(X)$ (when $h(X) = X$) is $\frac{1}{N}\sum_i (X_i w(X_i))$ and the estimate for $E(X^2)$ (when $h(X) = X^2$) is $\frac{1}{N}\sum_i (X_i^2 w(X_i))$.

Therefore, the estimate for Var(X) is $E(\hat{X^2}) - (E(\hat{X}))^2 = \frac{1}{N}\sum_i (X_i^2 w(X_i)) - (\frac{1}{N}\sum_i (X_i w(X_i)))^2$.

```r
set.seed(0)

N <- 1000
X <- runif(N)

# weight function
g=function(x){(x>0)*(x<1)*((x<=0.5)*4*x+ (x>0.5)*(4-4*x))}

# estimates
ex <- mean(X * g(X)) # E(X)
ex2 <- mean((X^2) * g(X)) # E(X^2)
var <- ex2 - (ex)^2 # Var(X)

cat("Using the importance sampler method:\n Estimated E(X):", ex,
    "\n Estimated Var(X):", var, "\n")
```

```
Using the importance sampler method:
 Estimated E(X): 0.4965714
 Estimated Var(X): 0.04021657
```

(c)

Using just samples from the uniform distribution, use the acceptance-rejection method to estimate E(X) and Var(X). To do this, do the following:

i. Generate a random variable X using the acceptance-rejection method. State how your algorithm works and that the acceptance test function is.

We know $2f(x) \geq g(x)$, which follows:

There exist a constant M (namely 2) such that $g(x) \leq Mf(x)$ for all x on the support of g.

Algorithm:

1. Generate $X \sim f(x)$, $U \sim uniform(0, 1)$
2. Accept Y=X if $U \leq \frac{g(X)}{Mf(X)} = \frac{g(X)}{2}$ since $f(X) = 1$
3. If not accept, go back to step 1 unitl you finally accept.

$\therefore$ The **acceptance test function** is $U \leq \frac{g(X)}{Mf(X)} = \frac{g(X)}{2}$.

ii. Give the rate of acceptance. That is, what percentage of proposed values is accepted.

iii. Provide your estimates of E(X) and Var(X) for this method.

```r
set.seed(0)
N <- 500 # 500 loops (1000 samples)
accepted <- c()

for (i in 1:N) {
  X <- runif(1)
  u <- runif(1)

  # acceptance test
  if (u <= g(X)/2) {accepted <- c(accepted, X)} # save accepted
}

n_acc <- length(accepted)
rate <- n_acc / N

# estimates
ex <- mean(accepted)
var <- var(accepted)

cat("Using acceptance-rejection method:\n Acceptance rate:", round(rate *
 ↪  100, 1), "%\n Estimated E(X):", ex, "\n Estimated Var(X):", var, "\n")
```

```
Using acceptance-rejection method:
 Acceptance rate: 49 %
 Estimated E(X): 0.480262
 Estimated Var(X): 0.03884125
```

(d)

Use the Metropolis-Hasting algorithm to generate an MCMC sequence of Xi's which have the triangle distribution as the invariant and the limiting distribution. Do this by doing the following:

- Let the transition function, q(x, y), be uniform density and have it not depend on the value of x. (That is, the new proposed move is always a sample from the uniform distribution on [0, 1] and this does not depend on the previous location. Therefore, q(x, y) = 1 for all values of x and for $y \in [0, 1]$.
- The invariant distribution is the triangle distribution. So, u(x) = g(x).
- Don't burn in the chain and don't thin the chain.

To answer this question provide the following:

i. What is the test function, $\alpha(x, y)$, for this chain given the above information?
ii. Provide the R code which samples the chain.
iii. What is the acceptance rate for the proposed moves in this chain?
iv. What are your estimates of E(X) and Var(X)?

We know that $q(x, y) \sim uniform(0, 1)$ and it's independent of the current state x. So every proposal is independent of the current x.

Since invariant distribution is the triangle distribution, u(x) = g(x).

Using metropolis-hasting algorithm, the acceptance probability (test function) is

$$\alpha(x, y) = \begin{cases} min[\frac{g(y)q(y,x)}{g(x)q(x,y)}, 1], & \text{if } g(x)q(x, y) > 0, \\ 1, & \text{otherwise} \end{cases}$$

Since $y \sim uniform(0, 1)$, $q(x, y) = 1$ for all values of x and for $y \in [0, 1]$. Since $x \sim uniform(0, 1)$, $q(y, x) = 1$ for all values of y and for $x \in [0, 1]$ as well.

Since $q(x, y) = q(y, x) = 1$, the **acceptance probability (test function)** simplifies to

$$\alpha(x, y) = \begin{cases} min[\frac{g(y)}{g(x)}, 1], & \text{if } g(x) > 0, \\ 1, & \text{otherwise} \end{cases}$$

```r
set.seed(0)

N <- 500 # 500 MH loops (1000 samples)
X_vals <- numeric(N)
X_vals[1] <- runif(1) # first X value
n_acc <- 0

for (i in 2:N) {
  y <- runif(1)
  u <- runif(1)

  # acceptance prob (test function)
  a <- min(1, g(y)/g(X_vals[i - 1]))

  if (u <= a) {
    X_vals[i] <- y # accept: save new value
    n_acc <- n_acc + 1
  } else {
    X_vals[i] <- X_vals[i - 1] # fail: stay at previous value
  }
}
```

```r
rate <- n_acc / (N - 1)

# estimates
ex <- mean(X_vals)
var <- var(X_vals)

cat("Using Metropolis-Hastings:\n Acceptance rate:", round(rate * 100, 1),
 ↪  "%\n Estimated E(X):", ex, "\n Estimated Var(X):", var, "\n")
```

```
Using Metropolis-Hastings:
 Acceptance rate: 68.5 %
 Estimated E(X): 0.5009705
 Estimated Var(X): 0.04142388
```