Applied Bayesian Analysis Assignment 3

Faizan Khalid Mohsin

March 14, 2019

Table of Contents

[Question 1 1](#_Toc5255801)

[Question 1 a 1](#_Toc5255802)

[Question 1 b 10](#_Toc5255803)

[Question 1 c 17](#_Toc5255804)

[Question 1 d 24](#_Toc5255805)

[Question 1 e 24](#_Toc5255806)

[Question 1 f 25](#_Toc5255807)

[Question 2 26](#_Toc5255808)

[Questin 2 a 26](#_Toc5255809)

[Question 2 b 49](#_Toc5255810)

[Question 3 50](#_Toc5255811)

[Question 3 a 50](#_Toc5255812)

[Question 3 b 50](#_Toc5255813)

[Question 3 c 50](#_Toc5255814)

[Question 3 d 51](#_Toc5255815)

# Question 1

## Question 1 a

library(R2OpenBUGS)

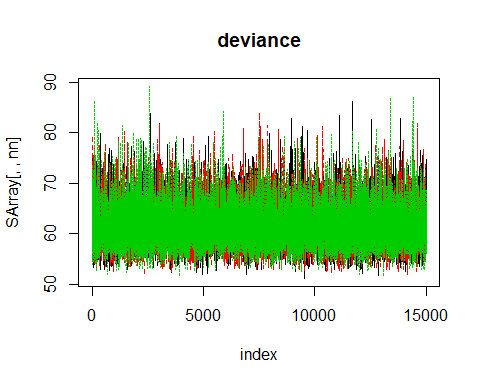
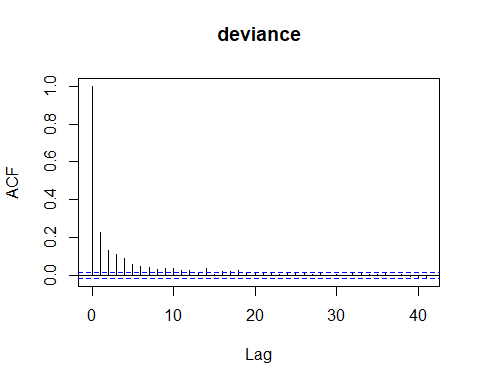
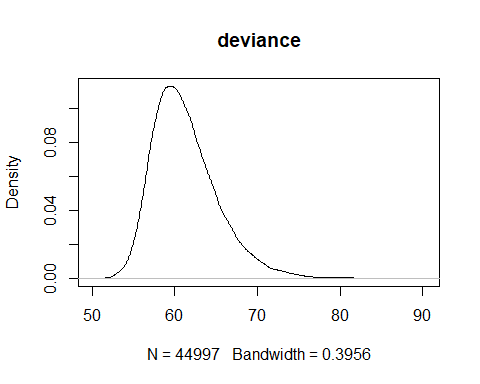
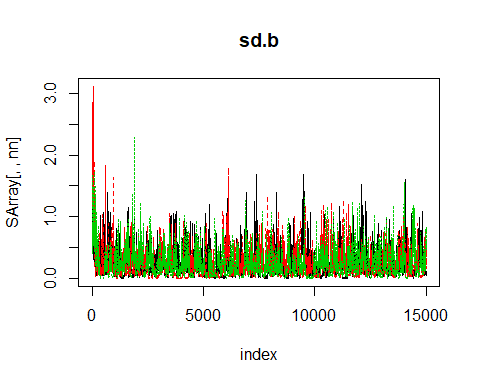
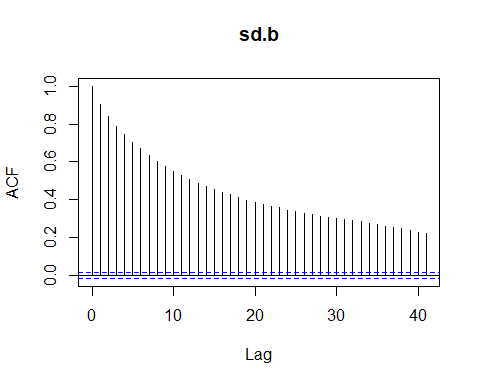
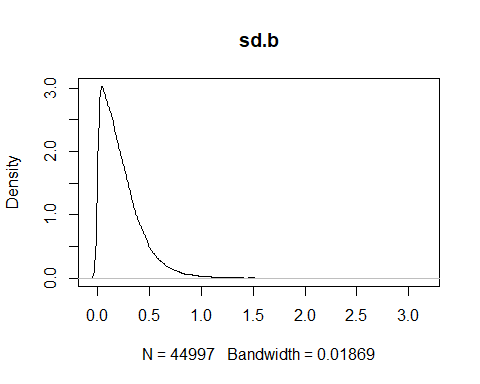
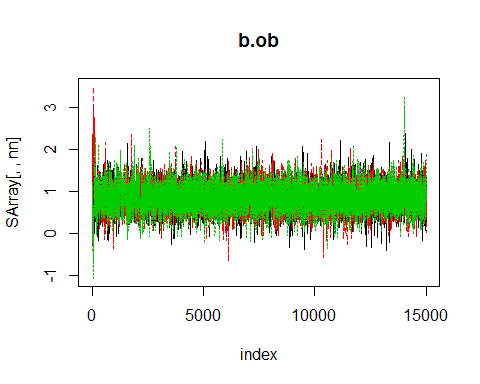
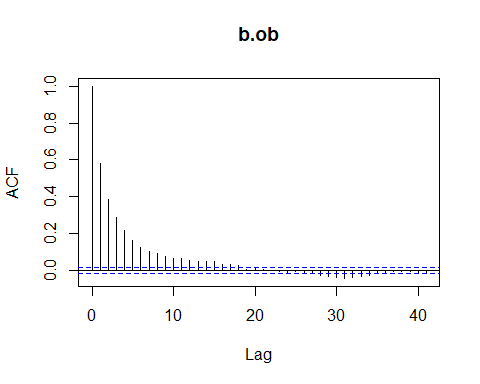
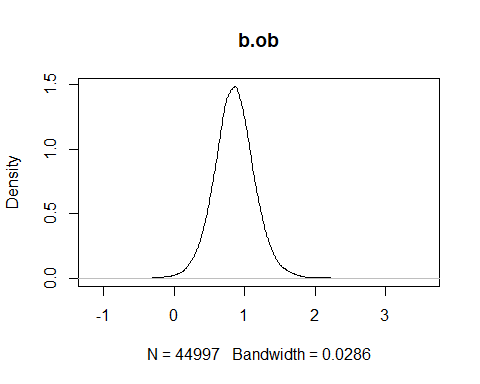
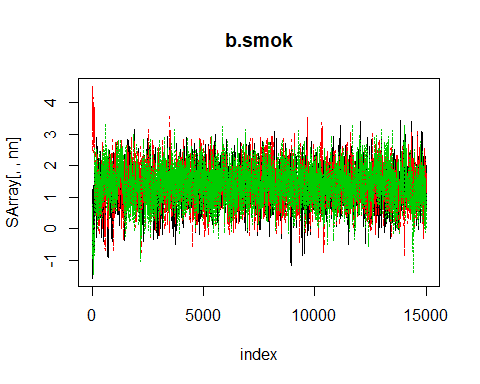
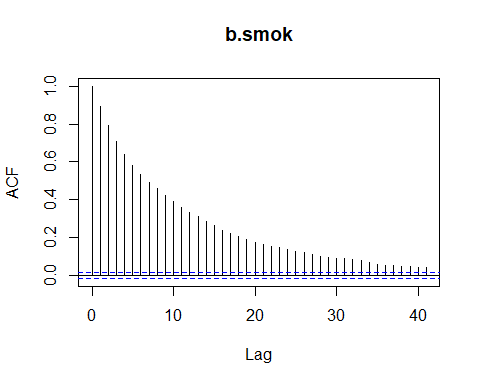
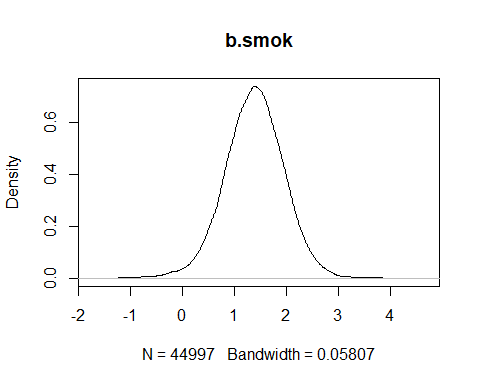
## Warning: package 'R2OpenBUGS' was built under R version 3.4.4

#  
#   
#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=runif(14,-.8,-.2),   
 b0=runif(1,-.8,-.2),  
 b.smok=runif(1,-.8,-.2),b.ob=runif(1,-.8,-.2), b.sn=runif(1,-.8,-.2),  
 b.male=runif(1,-.8,-.2), b.smsn=runif(1, -8,-.2), sd.b=runif(1,.2,.8)   
 ) }

paramsM3=c("b.smok", "b.ob" , "sd.b")  
  
SmokeHypeBaseM3=bugs(bugM3.dat, initM3.fun, paramsM3, model.file="SmokeHyperMod3.txt",  
 n.chains=3, n.iter=15000, n.burnin=1,  
 n.thin=1 , debug=TRUE  
)  
  
print(SmokeHypeBaseM3,dig=3) # Summary statistics

## Inference for Bugs model at "SmokeHyperMod3.txt",   
## Current: 3 chains, each with 15000 iterations (first 1 discarded)  
## Cumulative: n.sims = 44997 iterations saved  
## mean sd 2.5% 25% 50% 75% 97.5% Rhat n.eff  
## b.smok 1.380 0.579 0.215 1.017 1.388 1.754 2.487 1.003 1000  
## b.ob 0.859 0.300 0.286 0.674 0.854 1.037 1.461 1.001 20000  
## sd.b 0.228 0.210 0.006 0.079 0.177 0.316 0.750 1.004 1500  
## deviance 61.284 3.960 55.230 58.460 60.680 63.480 70.630 1.001 30000  
##   
## For each parameter, n.eff is a crude measure of effective sample size,  
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).  
##   
## DIC info (using the rule, pD = Dbar-Dhat)  
## pD = 7.951 and DIC = 69.230  
## DIC is an estimate of expected predictive error (lower deviance is better).

SArray= SmokeHypeBaseM3$sims.array  
vname=attr(SArray,"dimnames")[3][[1]]  
chainL=attr(SArray,"dim")[1][[1]]  
for(i in 1:length(vname)){   
 nn=vname[i]  
 plot(density(SArray[,,nn]), main=nn)  
 xnul=locator(1)   
 acf( SArray[,1,nn], main=nn) #note: this is only for 1st chain  
 xnul=locator(1)  
 matplot(1:chainL,SArray[,,nn], main=nn,xlab="index",type="l")  
 xnul=locator(1)  
}



From the trace plots, we can see that the chains are traversing the sample space in the same way, therefore they seem to have converged. We can see from the density plots that they have converged to the high probability region for all parameters being monitored. Also, the autocorrelation plots show that the chains are independently converging.

## Question 1 b

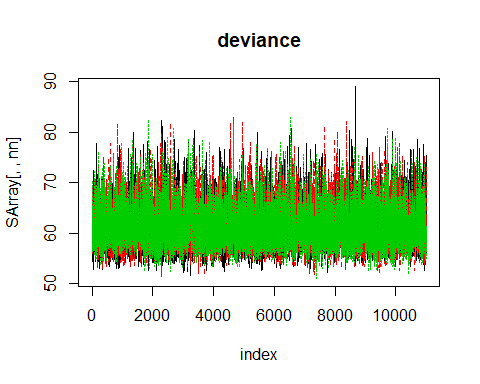
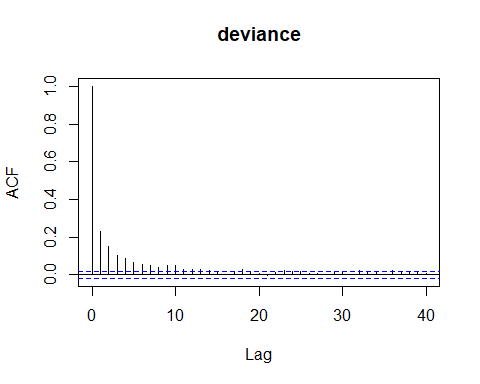
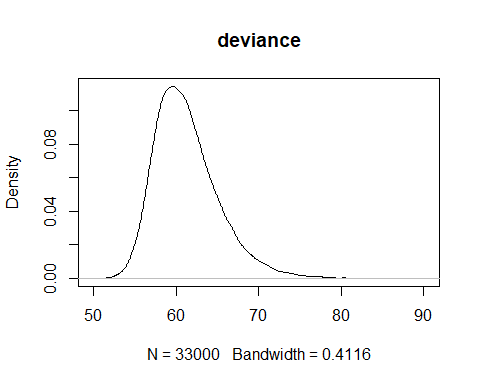
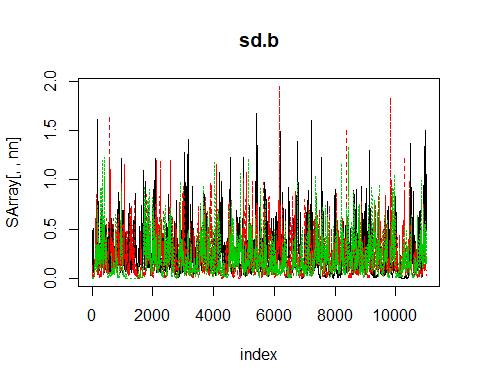
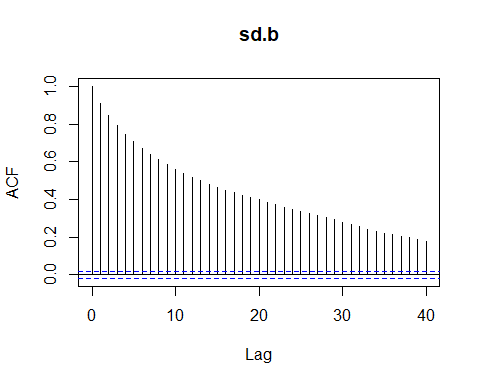
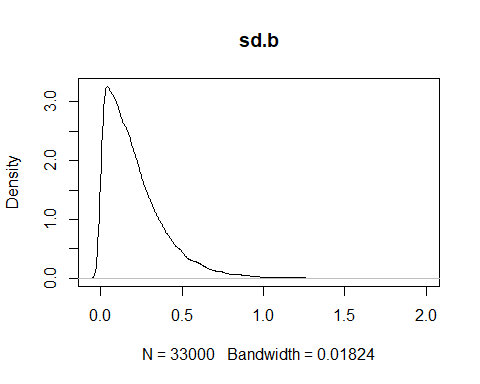
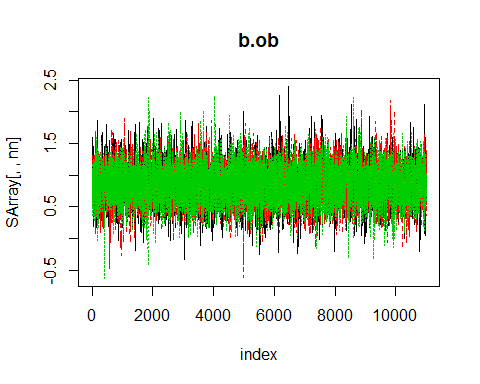
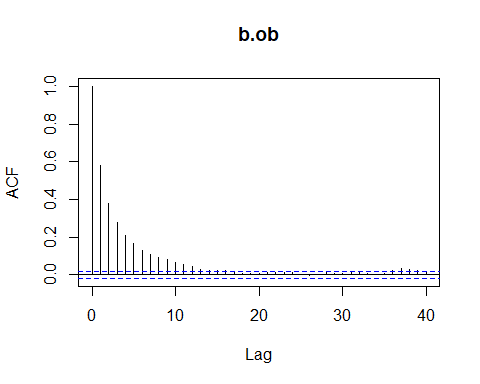
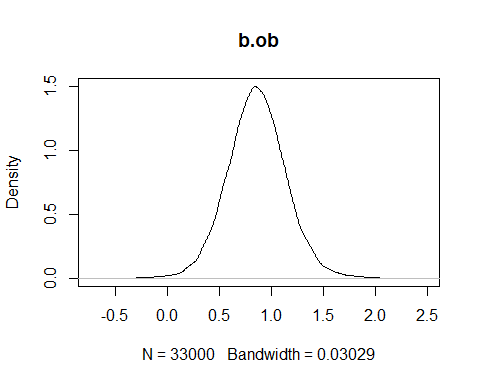
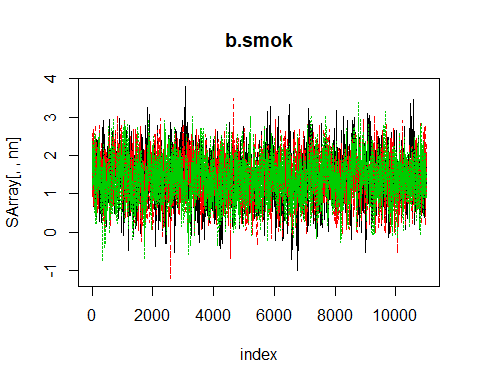
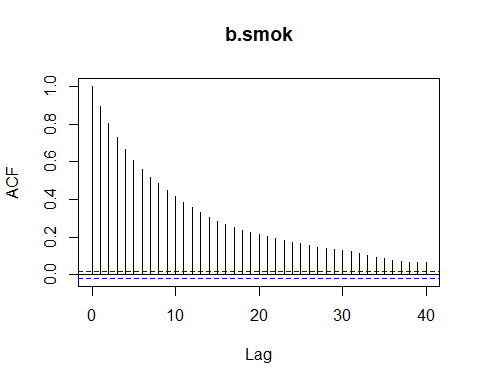
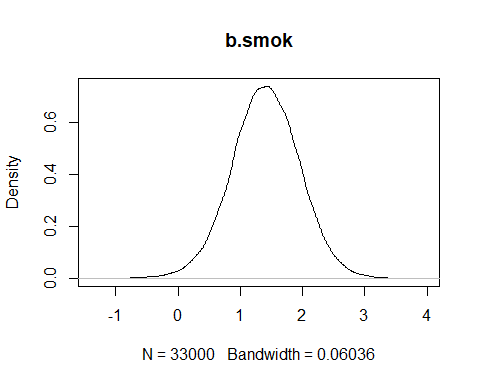
Burning first 4000 observations

SmokeHypeBaseM4=bugs(bugM3.dat, initM3.fun, paramsM3, model.file="SmokeHyperMod3.txt",  
 n.chains=3, n.iter=15000, n.burnin=4000,  
 n.thin=1 , debug=TRUE)  
print(SmokeHypeBaseM4,dig=3)

## Inference for Bugs model at "SmokeHyperMod3.txt",   
## Current: 3 chains, each with 15000 iterations (first 4000 discarded)  
## Cumulative: n.sims = 33000 iterations saved  
## mean sd 2.5% 25% 50% 75% 97.5% Rhat n.eff  
## b.smok 1.406 0.546 0.310 1.051 1.409 1.771 2.464 1.003 1400  
## b.ob 0.861 0.285 0.308 0.679 0.858 1.040 1.431 1.001 15000  
## sd.b 0.210 0.186 0.007 0.075 0.164 0.292 0.689 1.050 100  
## deviance 61.286 3.912 55.270 58.510 60.720 63.420 70.640 1.001 4500  
##   
## For each parameter, n.eff is a crude measure of effective sample size,  
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).  
##   
## DIC info (using the rule, pD = Dbar-Dhat)  
## pD = 7.805 and DIC = 69.090  
## DIC is an estimate of expected predictive error (lower deviance is better).

Autocorrelation and trace plots.

SArray= SmokeHypeBaseM4$sims.array  
vname=attr(SArray,"dimnames")[3][[1]]  
chainL=attr(SArray,"dim")[1][[1]]  
for(i in 1:length(vname)){   
 nn=vname[i]  
 plot(density(SArray[,,nn]), main=nn)  
 xnul=locator(1)   
 acf( SArray[,1,nn], main=nn) #note: this is only for 1st chain  
 xnul=locator(1)  
 matplot(1:chainL,SArray[,,nn], main=nn,xlab="index",type="l")  
 xnul=locator(1)  
}



After a burn-in of 4000 iterations, looking at the density and trace plots of all the parameters, the chains seem to have converged since they move similarly from the start over the sample space. Also, they are in the high probability region, depicted by the density plots.

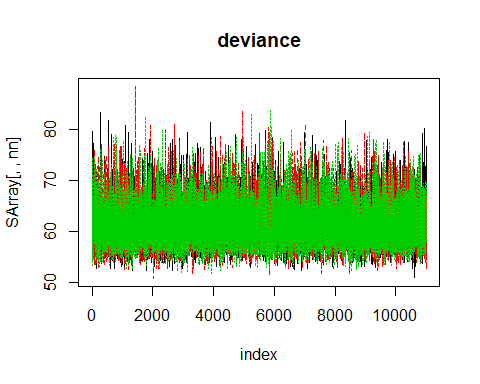
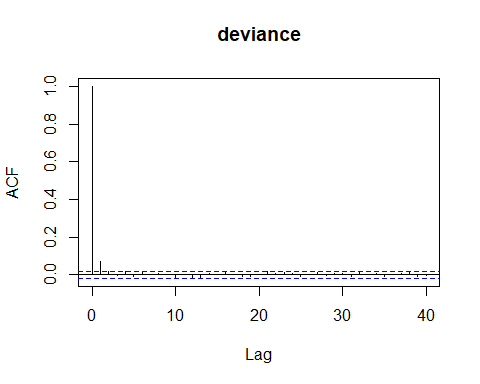
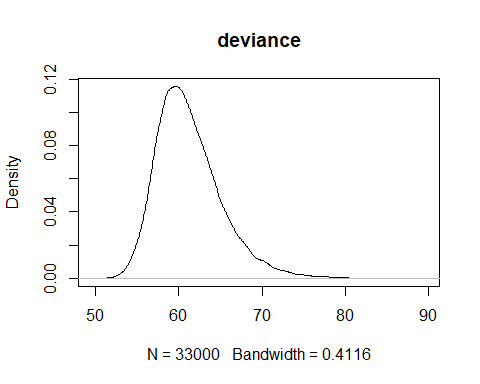
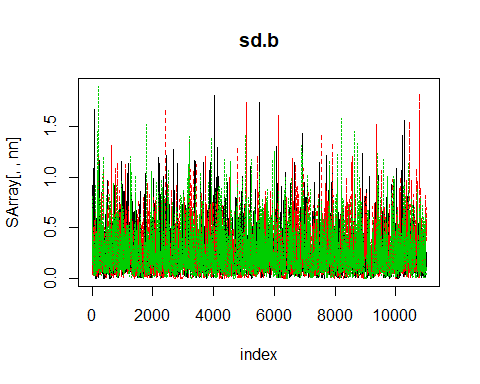
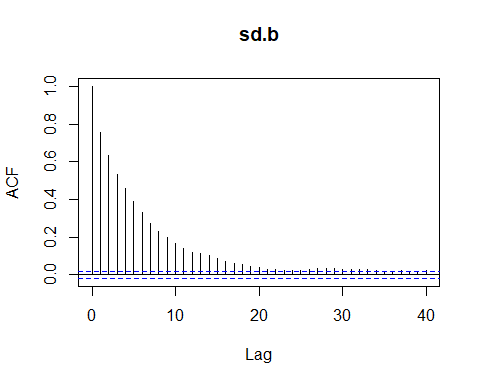
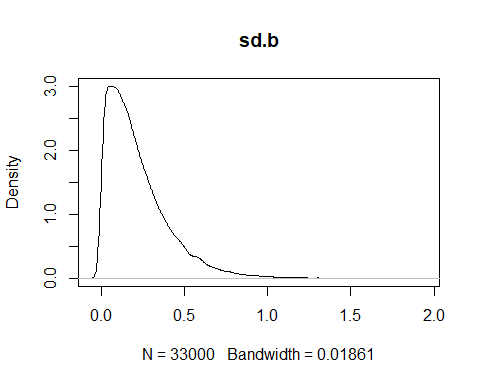
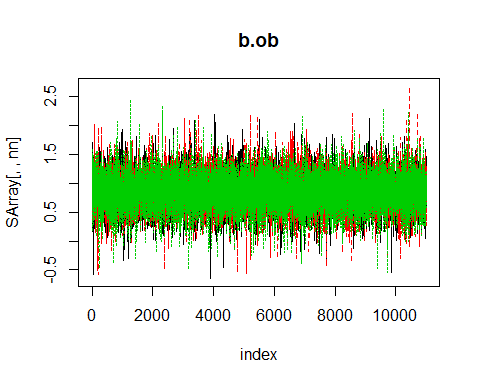
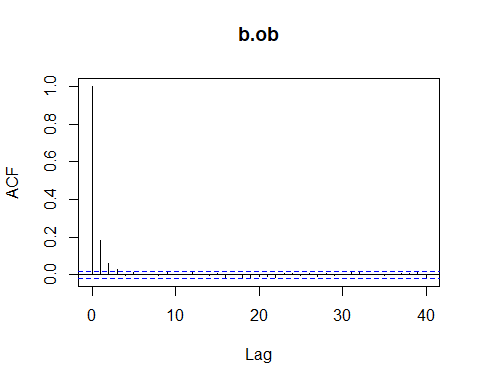
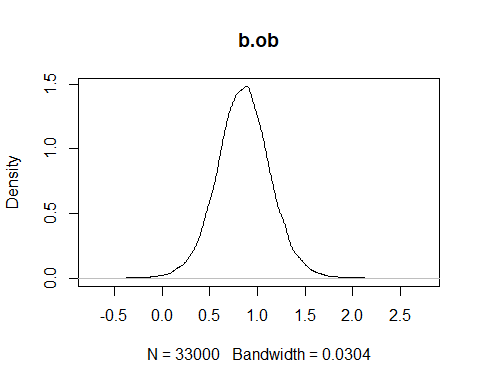
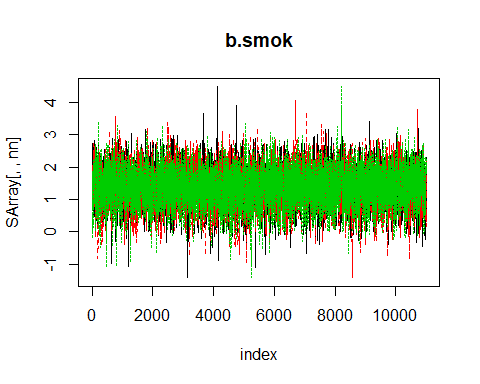
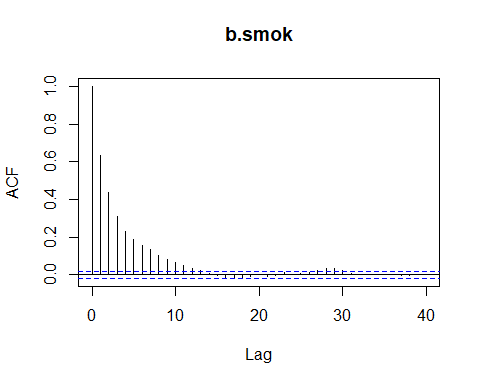
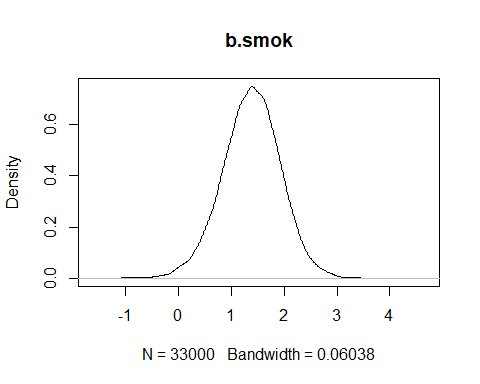
## Question 1 c

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

We thin the chain by a factor of 4 (n.thin=4).

SmokeHypeBaseM5=bugs(bugM3.dat, initM3.fun, paramsM3, model.file="SmokeHyperMod3.txt",  
 n.chains=3, n.iter=15000, n.burnin=4000,  
 n.thin=4 , debug=TRUE  
)

SArray= SmokeHypeBaseM5$sims.array  
vname=attr(SArray,"dimnames")[3][[1]]  
chainL=attr(SArray,"dim")[1][[1]]  
for(i in 1:length(vname)){   
 nn=vname[i]  
 plot(density(SArray[,,nn]), main=nn)  
 xnul=locator(1)   
 acf( SArray[,1,nn], main=nn) #note: this is only for 1st chain  
 xnul=locator(1)  
 matplot(1:chainL,SArray[,,nn], main=nn,xlab="index",type="l")  
 xnul=locator(1)  
}



Thinning helps in this case, where we have 15000 iterations. It is efficient here in reducing the autocorrelation.

From trace and autocorrelation plots, which are given below, we can see that thinning helps us get rid of the autocorrelation, especially in the case of sd.b parameter. These chains mix better, and will produce a more precise estimate of the sample.

## Question 1 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.

Model used here has been thinned by a factor of 4, with 15000 iterations

library(coda)

## Warning: package 'coda' was built under R version 3.4.4

make.mcmc.list=function(x){  
 aa=x$sims.array  
 zz=list(list())  
 for(i in 1:(dim(aa)[2]) ){  
 tmp=mcmc(aa[,i,])  
 zz=c(zz,list(tmp)) }  
 res=mcmc.list(zz[-1])  
 res  
}  
  
wave0=make.mcmc.list(SmokeHypeBaseM5)

# Provide the estimate of the posterior mean of the three parameters for each chain  
  
for (i in 1:3){  
 summary(wave0[i]); batchSE(wave0[i]); effectiveSize(wave0[i])# chain i  
}

Observing the distribution of the same parameters in each chain, we can see that they seem to converge to the high probability region for each chain from the summary statistics, and the small batch standard error calculated/shown point towards estimates with high precision.

## Question 1 e

gelman.diag(wave0[-3])

## Potential scale reduction factors:  
##   
## Point est. Upper C.I.  
## b.smok 1.00 1.00  
## b.ob 1.00 1.00  
## sd.b 1.01 1.04  
## deviance 1.00 1.00  
##   
## Multivariate psrf  
##   
## 1.01

Gelman diagnostic gives a value of 1 for the PSRF that shows that the mcmc algorithm has converged well.

geweke.diag(wave0)

## [[1]]  
##   
## Fraction in 1st window = 0.1  
## Fraction in 2nd window = 0.5   
##   
## b.smok b.ob sd.b deviance   
## 0.7909 2.0277 0.8393 -0.7188   
##   
##   
## [[2]]  
##   
## Fraction in 1st window = 0.1  
## Fraction in 2nd window = 0.5   
##   
## b.smok b.ob sd.b deviance   
## 0.3253 -1.4107 0.1656 1.4317   
##   
##   
## [[3]]  
##   
## Fraction in 1st window = 0.1  
## Fraction in 2nd window = 0.5   
##   
## b.smok b.ob sd.b deviance   
## -0.7699 0.6494 1.8713 -0.6904

Using the Geweke diagnostic to check algorithm convergence, we see that the values generated are not that high for z-scores, therefore the algorithm seems to converge well.

## Question 1 f

The mcmc algorithm seems to have converged well after thinning as the autocorrelation is reduced, trace plots seem to cross over in the high probability regions, and the diagnostic measures such as the gelman or geweke show no alarming measure sizes.

# Question 2

## Questin 2 a

For this question we first show the code, then some raw output of the code and lastly, we summarize the output at the end to answer question 2 a.

cropdata = data.frame(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))  
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)  
attach(cropdata)

## The following objects are masked \_by\_ .GlobalEnv:  
##   
## x, y

mean(x); sd(x)

## [1] 31

## [1] 9.521905

mean(y); sd(y)

## [1] 3283.125

## [1] 418.3434

mean(x^2);sd(x^2)

## [1] 1046

## [1] 595.496

oy <- y  
  
  
cat("  
 ### model1  
 model{  
   
 for( i in 1:16){  
   
 sx1[i]<- (x[i]-31)/9.52  
 y[i]<- (oy[i]-3283.125)/418.3434  
   
   
 y[i]~dnorm(mu[i],tau)  
 mu[i]<- beta[1]+beta[2]\*(sx1[i])   
   
   
 ######################################  
 # model checking steps are here.........  
   
 # getting the residuals for the observed values...  
 # note: I am deviating from the bugs manual... not getting the moments.  
   
 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)  
   
 # likelihood for each observed and replicated data....  
 # note: need to know the density function of the probability model  
 loglike[i]<- (0.5)\*log(tau/6.283) + (-0.5)\*tau\*pow((y[i]-mu[i]),2)  
 loglike.rep[i]<- (0.5)\*log(tau/6.283) + (-0.5)\*tau\*pow((y.rep[i]-mu[i]),2)  
   
 p.inv[i]<- 1/exp(loglike[i]) # this is to find the predictive ordinate of the observations   
   
 }  
   
 beta[1]~dnorm(0,0.0625)  
 beta[2]~dnorm(0,0.0625)  
   
   
   
 ##############################  
 # 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)  
   
 # Deviance statistic  
 dev<- -2\*sum(loglike[])  
 dev.rep <- -2\*sum(loglike.rep[])  
 dev.pval<-step(dev-dev.rep)  
   
   
 tau~dgamma(.5,.01)  
   
 #abeta[1]<-beta[1]\*9.52/418.34  
 abeta[2]<-beta[2]\*9.52/418.34  
 #abeta[3]<-beta[3]\*12.549/79.976  
 #abeta[4]<-beta[4]\*4.658/79.976  
   
 }  
   
 ", file="cropMod1.txt")  
data<-list("x", "oy")  
  
inits<-function(){ list(beta=rnorm(2), tau=runif(.5,1),y.rep=rnorm(16))}   
  
parameters<-c("beta", "tau",# the rest are for the model checking  
 "mu","res", "stdres", "res.rep", "stdres.rep", "p.smaller",  
 "p.inv", "chidev1.pval", "chidev2.pval", "chidev1.obs", "chidev2.obs",  
 "chidev1.rep", "chidev2.rep", "dev", "dev.rep", "dev.pval")  
  
  
cropMod1.sim<-bugs(data,inits, parameters,model.file="cropMod1.txt",  
 n.chains=3, n.iter=10000, n.burnin=700,  
 n.thin=1 #,debug=TRUE  
)   
  
##MODEL 2  
mean(x^2)

## [1] 1046

sd(x^2)

## [1] 595.496

cat("  
 ### model2  
 model{  
   
 for( i in 1:16){  
   
 sx1[i]<- (x[i]-31)/9.52  
 sx2[i]<- (pow(x[i],2)-1046)/595.496  
 y[i]<- (oy[i]-3283.125)/418.3434  
   
   
 y[i]~dnorm(mu[i],tau)  
 mu[i]<- beta[1]+beta[2]\*(sx1[i]) + beta[3]\*(sx2[i])   
   
   
 ######################################  
 # model checking steps are here.........  
   
 # getting the residuals for the observed values...  
 # note: I am deviating from the bugs manual... not getting the moments.  
   
 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)  
   
 # likelihood for each observed and replicated data....  
 # note: need to know the density function of the probability model  
 loglike[i]<- (0.5)\*log(tau/6.283) + (-0.5)\*tau\*pow((y[i]-mu[i]),2)  
 loglike.rep[i]<- (0.5)\*log(tau/6.283) + (-0.5)\*tau\*pow((y.rep[i]-mu[i]),2)  
   
 p.inv[i]<- 1/exp(loglike[i]) # this is to find the predictive ordinate of the observations   
   
 }  
   
 beta[1]~dnorm(0,0.0625)  
 beta[2]~dnorm(0,0.0625)  
 beta[3]~dnorm(0,0.0625)  
   
   
 ##############################  
 # 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)  
   
 # Deviance statistic  
 dev<- -2\*sum(loglike[])  
 dev.rep <- -2\*sum(loglike.rep[])  
 dev.pval<-step(dev-dev.rep)  
   
   
 tau~dgamma(.5,.01)  
   
 #abeta[1]<-beta[1]\*9.52/418.34  
 abeta[2]<-beta[2]\*9.52/418.34  
 abeta[3]<-beta[3]\*595.496/418.34  
 #abeta[4]<-beta[4]\*4.658/79.976  
   
 }  
   
 ", file="cropMod2.txt")  
  
cropMod2.sim<-bugs(data,inits, parameters,model.file="cropMod2.txt",  
 n.chains=3, n.iter=10000, n.burnin=700,  
 n.thin=1 #,debug=TRUE  
)   
  
#BOTH MODELS are writen correctly   
print(cropMod1.sim,dig=3)

# ONLY PARTIAL OUTPUT IS PRESENTED BELOW IN THE INTEREST OF BREVITY

## Inference for Bugs model at "cropMod1.txt",   
## Current: 3 chains, each with 10000 iterations (first 700 discarded)  
## Cumulative: n.sims = 27900 iterations saved  
## mean sd 2.5% 25% 50% 75% 97.5% Rhat  
## beta[1] 0.000 0.258 -0.512 -0.165 0.000 0.167 0.514 1.001  
## beta[2] 0.274 0.266 -0.254 0.103 0.274 0.446 0.804 1.001  
## tau 1.082 0.396 0.453 0.797 1.032 1.315 1.986 1.001  
## mu[1] -0.431 0.494 -1.413 -0.748 -0.431 -0.115 0.556 1.001  
## mu[16] 0.432 0.490 -0.533 0.116 0.432 0.749 1.397 1.001  
## res[1] -1.421 0.494 -2.409 -1.738 -1.422 -1.105 -0.440 1.001  
## res[16] -1.644 0.490 -2.610 -1.961 -1.644 -1.328 -0.679 1.001  
## stdres[1] -1.454 0.547 -2.526 -1.826 -1.452 -1.081 -0.385 1.001  
## stdres[16] -1.683 0.570 -2.812 -2.064 -1.680 -1.294 -0.583 1.001  
## res.rep[1] 0.006 1.038 -2.077 -0.658 0.010 0.665 2.055 1.001  
## res.rep[16] -0.006 1.033 -2.058 -0.666 -0.003 0.655 2.043 1.001  
## stdres.rep[1] 0.006 1.005 -1.986 -0.666 0.010 0.681 1.952 1.001  
## stdres.rep[16] -0.006 0.997 -1.977 -0.678 -0.003 0.663 1.961 1.001  
## p.smaller[1] 0.103 0.304 0.000 0.000 0.000 0.000 1.000 1.001  
## p.smaller[16] 0.071 0.256 0.000 0.000 0.000 0.000 1.000 1.001  
## p.inv[1] 12.402 23.528 2.931 4.776 7.225 12.570 52.440 1.001  
## p.inv[16] 23.036 94.831 3.377 6.172 10.290 19.742 107.252 1.001  
## chidev1.pval 0.545 0.498 0.000 0.000 1.000 1.000 1.000 1.001  
## chidev2.pval 0.545 0.498 0.000 0.000 1.000 1.000 1.000 1.001  
## chidev1.obs 15.955 2.461 13.880 14.370 15.170 16.660 22.565 1.001  
## chidev2.obs 16.963 5.838 7.592 12.780 16.290 20.430 30.190 1.001  
## chidev1.rep 17.125 9.953 5.260 10.490 14.840 21.110 41.895 1.001  
## chidev2.rep 16.047 5.663 6.884 11.960 15.350 19.440 28.825 1.001  
## dev 46.194 2.543 43.300 44.330 45.545 47.350 52.740 1.001  
## dev.rep 45.279 8.302 30.385 39.470 44.750 50.550 62.870 1.001  
## dev.pval 0.545 0.498 0.000 0.000 1.000 1.000 1.000 1.001  
## deviance 46.195 2.543 43.300 44.330 45.550 47.350 52.745 1.001  
## n.eff  
## beta[1] 28000  
## beta[2] 28000  
## tau 28000  
## mu[1] 28000  
## mu[16] 24000  
## res[1] 28000  
## res[16] 24000  
## stdres[1] 28000  
## stdres[16] 20000  
## res.rep[1] 8700  
## res.rep[16] 28000  
## stdres.rep[1] 8600  
## stdres.rep[16] 28000  
## p.smaller[1] 19000  
## p.smaller[16] 28000  
## p.inv[1] 28000  
## p.inv[16] 10000  
## chidev1.pval 28000  
## chidev2.pval 28000  
## chidev1.obs 28000  
## chidev2.obs 28000  
## chidev1.rep 28000  
## chidev2.rep 28000  
## dev 25000  
## dev.rep 28000  
## dev.pval 28000  
## deviance 25000  
##   
## For each parameter, n.eff is a crude measure of effective sample size,  
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).  
##   
## DIC info (using the rule, pD = Dbar-Dhat)  
## pD = 3.084 and DIC = 49.280  
## DIC is an estimate of expected predictive error (lower deviance is better).

print(cropMod2.sim,dig=3)

## Inference for Bugs model at "cropMod2.txt",   
## Current: 3 chains, each with 10000 iterations (first 700 discarded)  
## Cumulative: n.sims = 27900 iterations saved  
## mean sd 2.5% 25% 50% 75% 97.5% Rhat  
## beta[1] -0.001 0.131 -0.264 -0.084 0.000 0.083 0.261 1.001  
## beta[2] 5.891 1.039 3.680 5.252 5.927 6.593 7.798 1.012  
## beta[3] -5.670 1.039 -7.589 -6.374 -5.705 -5.031 -3.454 1.007  
## tau 4.329 1.678 1.665 3.111 4.121 5.320 8.210 1.001  
## mu[1] -1.761 0.349 -2.414 -1.993 -1.773 -1.544 -1.033 1.003  
## mu[16] -0.906 0.349 -1.566 -1.133 -0.918 -0.690 -0.184 1.003  
## res[1] -0.092 0.349 -0.820 -0.309 -0.080 0.140 0.561 1.003  
## res[16] -0.306 0.349 -1.029 -0.522 -0.295 -0.079 0.353 1.003  
## stdres[1] -0.157 0.663 -1.433 -0.614 -0.163 0.290 1.160 1.003  
## stdres[16] -0.602 0.659 -1.883 -1.051 -0.599 -0.162 0.695 1.003  
## res.rep[1] -0.001 0.520 -1.033 -0.332 -0.001 0.329 1.035 1.001  
## res.rep[16] -0.004 0.523 -1.054 -0.334 -0.003 0.331 1.035 1.001  
## stdres.rep[1] 0.000 0.997 -1.959 -0.670 -0.002 0.671 1.954 1.001  
## stdres.rep[16] -0.007 1.003 -1.985 -0.677 -0.006 0.667 1.950 1.001  
## p.smaller[1] 0.447 0.497 0.000 0.000 0.000 1.000 1.000 1.001  
## p.smaller[16] 0.308 0.462 0.000 0.000 0.000 1.000 1.000 1.001  
## p.inv[1] 1.756 2.014 0.945 1.226 1.459 1.849 4.367 1.003  
## p.inv[16] 2.331 4.136 0.976 1.301 1.628 2.300 7.614 1.003  
## chidev1.pval 0.545 0.498 0.000 0.000 1.000 1.000 1.000 1.001  
## chidev2.pval 0.545 0.498 0.000 0.000 1.000 1.000 1.000 1.001  
## chidev1.obs 4.072 1.009 3.144 3.428 3.772 4.366 6.781 1.006  
## chidev2.obs 16.905 5.764 7.493 12.750 16.245 20.370 30.010 1.001  
## chidev1.rep 4.361 2.635 1.308 2.619 3.728 5.358 11.115 1.001  
## chidev2.rep 16.028 5.650 6.951 11.940 15.380 19.350 28.970 1.001  
## dev 24.109 3.550 19.670 21.530 23.310 25.830 33.170 1.004  
## dev.rep 23.233 8.572 7.947 17.250 22.660 28.670 41.525 1.001  
## dev.pval 0.545 0.498 0.000 0.000 1.000 1.000 1.000 1.001  
## deviance 24.110 3.550 19.670 21.530 23.310 25.830 33.170 1.004  
## n.eff  
## beta[1] 28000  
## beta[2] 410  
## beta[3] 420  
## tau 4100  
## mu[1] 820  
## mu[16] 1100  
## res[1] 820  
## res[16] 1100  
## stdres[1] 870  
## stdres[16] 1200  
## res.rep[1] 27000  
## res.rep[16] 26000  
## stdres.rep[1] 26000  
## stdres.rep[16] 16000  
## p.smaller[1] 9000  
## p.smaller[16] 4900  
## p.inv[1] 2200  
## p.inv[16] 1500  
## chidev1.pval 28000  
## chidev2.pval 28000  
## chidev1.obs 1000  
## chidev2.obs 28000  
## chidev1.rep 7000  
## chidev2.rep 28000  
## dev 1100  
## dev.rep 8100  
## dev.pval 28000  
## deviance 1100  
##   
## For each parameter, n.eff is a crude measure of effective sample size,  
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).  
##   
## DIC info (using the rule, pD = Dbar-Dhat)  
## pD = 4.088 and DIC = 28.200  
## DIC is an estimate of expected predictive error (lower deviance is better).

#2a  
#Dic info for model 1  
#DIC info (using the rule, pD = Dbar-Dhat)  
#pD = 3.084 and DIC = 49.280  
#DIC is an estimate of expected predictive error (lower deviance is better).  
  
#Dic info for model 2  
#DIC info (using the rule, pD = Dbar-Dhat)  
#pD = 4.088 and DIC = 28.200  
#DIC is an estimate of expected predictive error (lower deviance is better).  
  
#Model 2 has a lower DIC and deviance based on the openbugs output.  
  
#calculating deviance  
# deviance, from model/not Openbugs intrinsic  
xxx<-cropMod1.sim  
xxx1<-xxx$sims.list$dev.rep  
temp<-c(xxx$mean$dev,quantile(xxx1,probs=c(0.025,.975)),mean(xxx1),sd(xxx1))  
names(temp)=c("Deviance","2.5%","97.5%","mean","SD");temp

## Deviance 2.5% 97.5% mean SD   
## 46.194343 30.384750 62.870000 45.278596 8.302437

xxx1<-xxx$sims.list$chidev2.rep  
temp<-c(xxx$mean$chidev2.obs,quantile(xxx1,probs=c(0.025,.975)),mean(xxx1),sd(xxx1))  
names(temp)=c("ChiDev2","2.5%","97.5%","mean","SD");temp

## ChiDev2 2.5% 97.5% mean SD   
## 16.963040 6.884000 28.825250 16.047275 5.662972

##### non calibrated ("pval-stats")  
  
apply(xxx$sims.list$p.smaller,2,mean)

## [1] 0.10268817 0.09215054 0.63849462 0.71315412 0.36799283 0.47086022  
## [7] 0.72390681 0.91795699 0.89032258 0.77784946 0.80232975 0.46698925  
## [13] 0.61415771 0.34688172 0.23057348 0.07064516

xxx$mean$chidev2.pval

## [1] 0.5450896

xxx$mean$dev.pval

## [1] 0.5450896

###########################  
###########################  
  
# comparing intrinsic and self calculated value for Deviance:  
  
xxx1<-xxx$sims.list$dev  
xxx2<-xxx$sims.list$deviance  
temp<-rbind(  
 quantile(xxx1,probs=c(0.025,.25,.5,.75,.975)),  
 quantile(xxx2,probs=c(0.025,.25,.5,.75,.975)) )  
rownames(temp)=c("SelfProgramed:","Openbugs Made:");temp

## 2.5% 25% 50% 75% 97.5%  
## SelfProgramed: 43.3 44.33 45.545 47.35 52.74000  
## Openbugs Made: 43.3 44.33 45.550 47.35 52.74525

c(xxx$mean$deviance,xxx$mean$dev)

## [1] 46.19482 46.19434

Dbar<-mean(xxx1);Dbar

## [1] 46.19434

pd2<-0.5\*var(xxx1);pd2

## [1] 3.234312

devNormFunc <- function(beta0, beta1, beta2, tau, x, y){  
 mu<- beta0+beta1\*x + beta2\*x^2  
 return(-2\*sum(log(dnorm(y,mu,1/sqrt(tau)))))}  
beta0Bar<- xxx$mean$beta0  
beta1Bar<- xxx$mean$beta1  
beta2Bar<- 0   
tauBar <- xxx$mean$tau  
Dhat <- devNormFunc(beta0Bar, beta1Bar, beta2Bar, tauBar, x,y);Dhat

## [1] 0

pd1<-Dbar-Dhat;pd1

## [1] 46.19434

DIC1<-Dbar+pd1; DIC1

## [1] 92.38869

DIC2<-Dbar+pd2; DIC2

## [1] 49.42866

#bayes factor  
xxx<-cropMod1.sim  
xxx$mean$p.inv

## [1] 12.401625 12.000888 2.986302 3.326517 2.853119 2.657246 3.216499  
## [8] 9.251179 6.548777 3.690001 4.068193 2.682927 2.850924 3.058971  
## [15] 4.266920 23.036100

# Getting predictive Ordinates and Pseudo m2LogL (aka: )  
# Note: difference of Pseudo-m2LogL is the PsuedoBayes Factor btw models.  
#  
xxx1<-xxx$mean$p.inv  
PLogLI=-1\*log(xxx1)  
temp<-cbind(xxx1,1/xxx1,PLogLI);colnames(temp)=c("p.inv","p(x)","PLogLI");temp

## p.inv p(x) PLogLI  
## [1,] 12.401625 0.08063459 -2.5178275  
## [2,] 12.000888 0.08332717 -2.4849806  
## [3,] 2.986302 0.33486233 -1.0940358  
## [4,] 3.326517 0.30061468 -1.2019260  
## [5,] 2.853119 0.35049363 -1.0484128  
## [6,] 2.657246 0.37632942 -0.9772904  
## [7,] 3.216499 0.31089705 -1.1682934  
## [8,] 9.251179 0.10809433 -2.2247510  
## [9,] 6.548777 0.15270026 -1.8792784  
## [10,] 3.690001 0.27100262 -1.3056268  
## [11,] 4.068193 0.24580940 -1.4031988  
## [12,] 2.682927 0.37272727 -0.9869083  
## [13,] 2.850924 0.35076348 -1.0476431  
## [14,] 3.058971 0.32690729 -1.1180787  
## [15,] 4.266920 0.23436108 -1.4508923  
## [16,] 23.036100 0.04341013 -3.1370626

Pseudom2logL= -2\*sum(PLogLI);Pseudom2logL # Bayes factor

## [1] 50.09241

#DEVIANCE for model 2  
xxx<-cropMod2.sim  
xxx1<-xxx$sims.list$dev.rep  
temp<-c(xxx$mean$dev,quantile(xxx1,probs=c(0.025,.975)),mean(xxx1),sd(xxx1))  
names(temp)=c("Deviance","2.5%","97.5%","mean","SD");temp

## Deviance 2.5% 97.5% mean SD   
## 24.109448 7.946900 41.525250 23.232516 8.572244

xxx1<-xxx$sims.list$chidev2.rep  
temp<-c(xxx$mean$chidev2.obs,quantile(xxx1,probs=c(0.025,.975)),mean(xxx1),sd(xxx1))  
names(temp)=c("ChiDev2","2.5%","97.5%","mean","SD");temp

## ChiDev2 2.5% 97.5% mean SD   
## 16.905456 6.950850 28.970000 16.028472 5.649806

xxx1<-xxx$sims.list$dev  
xxx2<-xxx$sims.list$deviance  
temp<-rbind(  
 quantile(xxx1,probs=c(0.025,.25,.5,.75,.975)),  
 quantile(xxx2,probs=c(0.025,.25,.5,.75,.975)) )  
rownames(temp)=c("SelfProgramed:","Openbugs Made:");temp

## 2.5% 25% 50% 75% 97.5%  
## SelfProgramed: 19.67 21.53 23.31 25.83 33.17  
## Openbugs Made: 19.67 21.53 23.31 25.83 33.17

c(xxx$mean$deviance,xxx$mean$dev)

## [1] 24.10992 24.10945

Dbar<-mean(xxx1);Dbar

## [1] 24.10945

pd2<-0.5\*var(xxx1);pd2

## [1] 6.300758

devNormFunc <- function(beta0, beta1, beta2, tau, x, y){  
 mu<- beta0+beta1\*x + beta2\*x^2  
 return(-2\*sum(log(dnorm(y,mu,1/sqrt(tau)))))}  
beta0Bar<- xxx$mean$beta0  
beta1Bar<- xxx$mean$beta1  
beta2Bar<- 0   
tauBar <- xxx$mean$tau  
Dhat <- devNormFunc(beta0Bar, beta1Bar, beta2Bar, tauBar, x,y);Dhat

## [1] 0

pd1<-Dbar-Dhat;pd1

## [1] 24.10945

DIC1<-Dbar+pd1; DIC1

## [1] 48.2189

DIC2<-Dbar+pd2; DIC2

## [1] 30.41021

#bayes factor  
xxx<-cropMod2.sim  
xxx$mean$p.inv

## [1] 1.756100 4.921280 5.638659 2.869548 4.461320 4.122641 1.426486  
## [8] 3.904659 2.243141 1.382970 1.666547 2.094945 1.603159 1.417642  
## [15] 1.494116 2.331283

# Getting predictive Ordinates and Pseudo m2LogL (aka: )  
# Note: difference of Pseudo-m2LogL is the PsuedoBayes Factor btw models.  
#  
xxx1<-xxx$mean$p.inv  
PLogLI=-1\*log(xxx1)  
temp<-cbind(xxx1,1/xxx1,PLogLI);colnames(temp)=c("p.inv","p(x)","PLogLI");temp

## p.inv p(x) PLogLI  
## [1,] 1.756100 0.5694436 -0.5630955  
## [2,] 4.921280 0.2031992 -1.5935687  
## [3,] 5.638659 0.1773471 -1.7296463  
## [4,] 2.869548 0.3484870 -1.0541545  
## [5,] 4.461320 0.2241489 -1.4954446  
## [6,] 4.122641 0.2425630 -1.4164939  
## [7,] 1.426486 0.7010231 -0.3552144  
## [8,] 3.904659 0.2561043 -1.3621704  
## [9,] 2.243141 0.4458034 -0.8078773  
## [10,] 1.382970 0.7230815 -0.3242333  
## [11,] 1.666547 0.6000431 -0.5107537  
## [12,] 2.094945 0.4773395 -0.7395273  
## [13,] 1.603159 0.6237684 -0.4719761  
## [14,] 1.417642 0.7053966 -0.3489950  
## [15,] 1.494116 0.6692921 -0.4015347  
## [16,] 2.331283 0.4289484 -0.8464186

Pseudom2logL= -2\*sum(PLogLI);Pseudom2logL # Bayes factor

## [1] 28.04221

From Openbugs output:

Dic info for model 1 pD = 3.084 and DIC = 49.280

Dic info for model 2 pD = 4.088 and DIC = 28.200

**Notice that DIC, deviance and bayes factor measures are lower for model 2 which is the quadratic model, and therefore model 2 is preferred.**

### 

### MODEL 1

From the above output we obtain the following:

The Deviance:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Deviance | 2.5% | 97.5% | mean | SD |
| 46.204683 | 30.400000 | 62.880000 | 45.288958 | 8.305494 |

The ChiDev2:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ChiDev2 | 2.5% | 97.5% | mean | SD |
| 16.963055 | 6.884000 | 28.825250 | 16.047275 | 5.662972 |

And lastlly:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X | 2.5% | 25% | 50% | 75% | 97.5% |
| SelfProgramed: | 43.3 | 44.34 | 45.55 | 47.36 | 52.78 |
| Openbugs Made: | 43.3 | 44.34 | 45.55 | 47.36 | 52.78 |

SELF programmed and OPENBUGS MEANS for Deviance are 46.20517 46.20468

DIC 49.4639

Pseudo-Bayes factor 50.11571

### MODEL 2

From the above output we obtain the following:

The Deviance:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Deviance | 2.5% | 97.5% | mean | SD |
| 24.109448 | 7.946900 | 41.525250 | 23.232516 | 8.572244 |

The ChiDev2:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ChiDev2 | 2.5% | 97.5% | mean | SD |
| 16.905456 | 6.950850 | 28.970000 | 16.028472 | 5.649806 |

And lastlly:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X | 2.5% | 25% | 50% | 75% | 97.5% |
| SelfProgramed: | 19.67 | 21.53 | 23.31 | 25.83 | 33.17 |
| Openbugs Made: | 19.67 | 21.53 | 23.31 | 25.83 | 33.17 |

SELF programmed and OPENBUGS MEANS for Deviance are 24.10992 24.10945

DIC 30.41021

Pseudo-Bayes factor 28.04221

BAYES-FACTOR CALCULATION

# MODEL for Bayes factor calculation  
  
cat("  
 ### model3  
 model{  
 for( i in 1:16){  
 sx1[i]<- (x[i]-31)/9.52  
 sx2[i]<- (pow(x[i],2)-1046)/595.496  
 y[i]<- (oy[i]-3283.125)/418.3434  
 y[i]~dnorm(mu[i],tau)  
 mu[i]<- del[1]\*beta[1]\*(sx1[i]) + del[2]\*beta[2]\*(sx2[i])  
 }  
 beta[1]~dnorm(0,0.0625)  
 beta[2]~dnorm(0,0.0625)  
 tau~dgamma(.5,.01)  
 for(k in 1:2){del[k]~dbern(0.5)}  
 for(i1 in 1:2){  
 for(i2 in 1:2){  
 mod[i1,i2]<-equals((2-i1),del[1])\*equals( (2-i2),del[2])  
 }}  
 }   
 ", file="cropMod3.txt")  
  
inits1<-function(){ list(beta=rnorm(3), del=c(1,2) ,tau=runif(.5,1),y.rep=rnorm(16))}   
  
parameters1<-c("mod","del","beta")  
  
cropMod3.sim<-bugs(data,inits1, parameters1,model.file="cropMod3.txt",  
 n.chains=3, n.iter=10000, n.burnin=700,  
 n.thin=1 #,debug=TRUE  
 )  
  
print(cropMod3.sim, dig=5) # printing model probabilities

## Inference for Bugs model at "cropMod3.txt",   
## Current: 3 chains, each with 10000 iterations (first 700 discarded)  
## Cumulative: n.sims = 27900 iterations saved  
## mean sd 2.5% 25% 50% 75% 97.5% Rhat  
## mod[1,1] 0.94136 0.23495 0.00000 1.000 1.000 1.000 1.00000 1.41373  
## mod[1,2] 0.00509 0.07116 0.00000 0.000 0.000 0.000 0.00000 1.29955  
## mod[2,1] 0.00351 0.05916 0.00000 0.000 0.000 0.000 0.00000 1.29684  
## mod[2,2] 0.05004 0.21802 0.00000 0.000 0.000 0.000 1.00000 1.39201  
## del[1] 0.94645 0.22513 0.00000 1.000 1.000 1.000 1.00000 1.40069  
## del[2] 0.94487 0.22823 0.00000 1.000 1.000 1.000 1.00000 1.40468  
## beta[1] 5.56211 1.93939 -0.37327 5.163 5.935 6.564 7.67200 1.24923  
## beta[2] -5.36073 1.89902 -7.46200 -6.345 -5.719 -4.943 0.43983 1.24044  
## deviance 24.25350 6.14401 19.36000 20.660 22.300 25.000 44.99000 1.21817  
## n.eff  
## mod[1,1] 17  
## mod[1,2] 200  
## mod[2,1] 280  
## mod[2,2] 20  
## del[1] 19  
## del[2] 18  
## beta[1] 32  
## beta[2] 34  
## deviance 27  
##   
## For each parameter, n.eff is a crude measure of effective sample size,  
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).  
##   
## DIC info (using the rule, pD = var(deviance)/2)  
## pD = 17.12945 and DIC = 41.38296  
## DIC is an estimate of expected predictive error (lower deviance is better).

BAYES FACTOR

We take Model(1,1) and Model (1,2) probabilities from the output above to get the Bayes factor.

2\*log(.94/.005) #BAYES FACTOR

## [1] 10.47288

The above calculation 2log(BF)=10.47, shows that the quadratic model is an improvement over the linear model.

## Question 2 b

For model 1, calculate 1) the residuals, 2) the standardized residuals, and 3) the chance of getting a more extreme observation. Comment on how well you think the model fits the data.

Below we calculate the residuals and distribution of the statistics under the predicted distribution.

xxx<-cropMod1.sim  
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

## res 2.5% 97.5% mean SD  
## [1,] -1.42144105 -2.076525 2.055050 0.0062601396 1.038220  
## [2,] -1.45510978 -2.032000 2.032000 -0.0020030019 1.031310  
## [3,] 0.36615507 -2.037050 2.026000 -0.0070145278 1.030376  
## [4,] 0.59303699 -2.034000 2.045575 0.0092588728 1.031934  
## [5,] -0.33941827 -2.046000 2.046000 0.0076637922 1.027122  
## [6,] -0.07907171 -2.075000 2.055000 -0.0058125469 1.039830  
## [7,] 0.60437220 -2.081525 2.053050 -0.0029415863 1.038352  
## [8,] 1.46231510 -2.092525 2.058525 -0.0075467085 1.039572  
## [9,] 1.26131412 -2.074000 2.042000 -0.0048087624 1.037108  
## [10,] 0.78064590 -2.050000 2.035050 0.0008537005 1.030170  
## [11,] 0.87127287 -2.047050 2.057525 0.0007529547 1.031227  
## [12,] -0.08269294 -2.043050 2.030525 -0.0036801887 1.030697  
## [13,] 0.29956299 -2.062000 2.045525 0.0013273343 1.039885  
## [14,] -0.41775738 -2.073000 2.061525 -0.0063400294 1.036641  
## [15,] -0.80520228 -2.105050 2.031575 -0.0166401295 1.037538  
## [16,] -1.64443360 -2.058050 2.043000 -0.0058182587 1.032675

Below we calculate the STANDARD RESIDUALS and distribution of the statistics under the predicted distribution.

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

## stdres 2.5% 97.5% mean SD  
## [1,] -1.45395464 -1.986000 1.952000 0.0055035023 1.0046636  
## [2,] -1.48846714 -1.961525 1.942000 -0.0014370468 0.9980588  
## [3,] 0.37473454 -1.983000 1.939000 -0.0070272732 0.9987307  
## [4,] 0.60678269 -1.953525 1.974525 0.0098946483 0.9985751  
## [5,] -0.34724131 -1.940050 1.965000 0.0093059947 0.9923961  
## [6,] -0.08095662 -2.005050 1.939000 -0.0057495479 1.0057053  
## [7,] 0.61818329 -1.990525 1.970050 -0.0040182117 1.0048078  
## [8,] 1.49584539 -1.980050 1.961000 -0.0063808651 1.0102707  
## [9,] 1.29014867 -1.970000 1.946000 -0.0042359123 1.0024274  
## [10,] 0.79832526 -1.958525 1.961525 0.0011302946 0.9989464  
## [11,] 0.89098036 -1.946000 1.967000 -0.0002209465 0.9955796  
## [12,] -0.08505441 -1.980000 1.944000 -0.0035628062 0.9997934  
## [13,] 0.30595192 -1.978525 1.963000 0.0006958645 1.0058822  
## [14,] -0.42797662 -1.983000 1.970525 -0.0053971334 1.0032588  
## [15,] -0.82442536 -2.032050 1.934000 -0.0176827528 1.0069579  
## [16,] -1.68307328 -1.977000 1.961000 -0.0055120920 0.9973165

Below we calculate the PROBABILITY of extreme values.

apply(xxx$sims.list$p.smaller,2,mean)

## [1] 0.10268817 0.09215054 0.63849462 0.71315412 0.36799283 0.47086022  
## [7] 0.72390681 0.91795699 0.89032258 0.77784946 0.80232975 0.46698925  
## [13] 0.61415771 0.34688172 0.23057348 0.07064516

The mean of the residuals is within the 95% Cis of the residuals, which shows that the model fits the data well.

# Question 3

## Question 3 a

u1<-runif(1000 ,0,1) # Sampling from a uniform  
u2<-runif(1000 ,0,1) # Sampling from a uniform  
x<- (u1+u2)/2 # Adding the 2 uniform distributions  
c(mean(x),var(x)) # Mean and Variance

## [1] 0.49804775 0.04124784

## Question 3 b

The importance sampler method

u1<-runif(1000 ,0,1) # Sampling from a uniform  
g=function(x){(x>0)\*(x<1)\*((x<=0.5)\*4\*x+ (x>0.5)\*(4-4\*x))}  
gu1<- g(u1)  
fu1<- 1\*(u1>=0)\*(u1<=1)  
# weight function is gu1/fu1   
w<-(gu1/fu1) # Weight function  
c(mean(u1\*w),var(u1\*w)) # Mean and Variance based on importance sampling

## [1] 0.5129388 0.1191696

## Question 3 c

Acceptance-rejection method

x <- runif(1000)  
Y <- rep(0,1000)  
accept = c()  
for(i in 1:length(x)){  
 U = runif(1)  
 if(U <= ( g(x[i]) / (2\*dunif(x[i])) ) ){  
 Y[i] = x[i]  
 accept[i] ="Yes"  
 }  
 else {  
 accept[i] = "No"   
 Y[i]<- 1000  
 }  
}  
  
mean(accept == "Yes") #Acceptance rate

## [1] 0.507

c(mean(Y[Y<1000]),var(Y[Y<1000])) #Mean and variance of y's that were accepted

## [1] 0.5096796 0.0377997

The above algorithm is sampling from g(x) and generating sample using a uniform distribution, with a constant.

If U<= g/(some constant)\*f(x), we sample the observation. So, the algorithm above picks out the (X, U) points which are sampled by X ∼ f(x)=uniform and U ∼ Unif(0, 1) which are “under” the g(x) curve.

## Question 3 d

Metropolis Hasting Algorithm

q1 <- function(x,y){(y>=0)\*(y<=1)\*1} # THIS IS q(x,y)  
  
g=function(x){(x>0)\*(x<1)\*((x<=0.5)\*4\*x+ (x>0.5)\*(4-4\*x))} #this is our u(x)  
  
alph = function (x,y){min(g(y)/g(x),1)} #TEST FUNCTION  
  
#R-code to sample the chain  
x < - rep (0,1000)  
x[1]<-0.5  
accepted<- c()  
for (j in 2:1000){  
 ystar <- runif(1)  
 T <- runif(1)  
 if (T<= alph(x[j-1],ystar)){  
 x[j]<- ystar  
 accepted[j]<- "Yes"  
 }  
 else{  
 x[j]<- x[j-1]  
 accepted[j]<- "No"  
 }  
}

The mean, variance and acceptance rate.

mean(x) # Mean

## [1] 0.494352

var(x) # Variance

## [1] 0.04488678

mean(accepted=="Yes",na.rm=TRUE) # Acceptance rate

## [1] 0.6886887