**Question 1**

**#1-Part a**

Running the model while monitoring 3 parameters with 15000 iterations

Chosen sd.b, b.smok and b.ob, as the 3 parameters to monitor

> paramsM4=c("b.smok", "b.ob" , "sd.b")

Running the model with the 3 parameters with 15000 iterations

> SmokeHypeBaseM4=bugs(bugM3.dat, initM3.fun, paramsM4, model.file="SmokeHyperMod3.txt",

+ n.chains=3, n.iter=15000, n.burnin=1,

+ n.thin=1 , debug=TRUE

+ )

> print(SmokeHypeBaseM4,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 1.334 0.595 0.102 0.988 1.363 1.719 2.407 1.005 1000

b.ob 0.841 0.314 0.239 0.662 0.847 1.030 1.427 1.009 2700

sd.b 0.234 0.219 0.006 0.087 0.181 0.316 0.772 1.061 110

deviance 61.272 3.938 55.190 58.460 60.690 63.470 70.560 1.001 26000

For each parameter, n.eff is a crude measure of effective sample size,

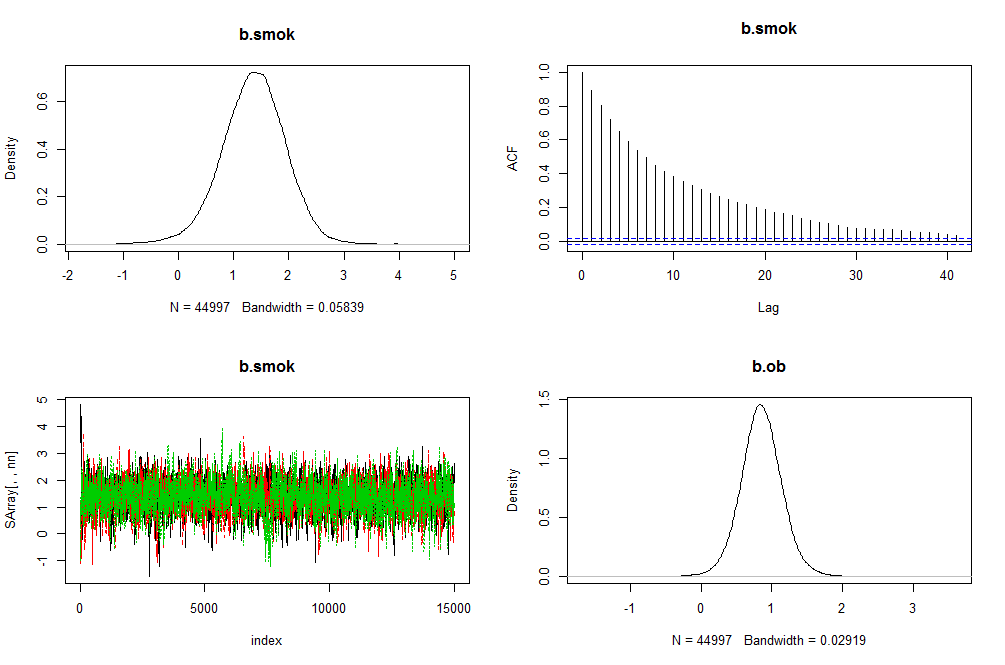
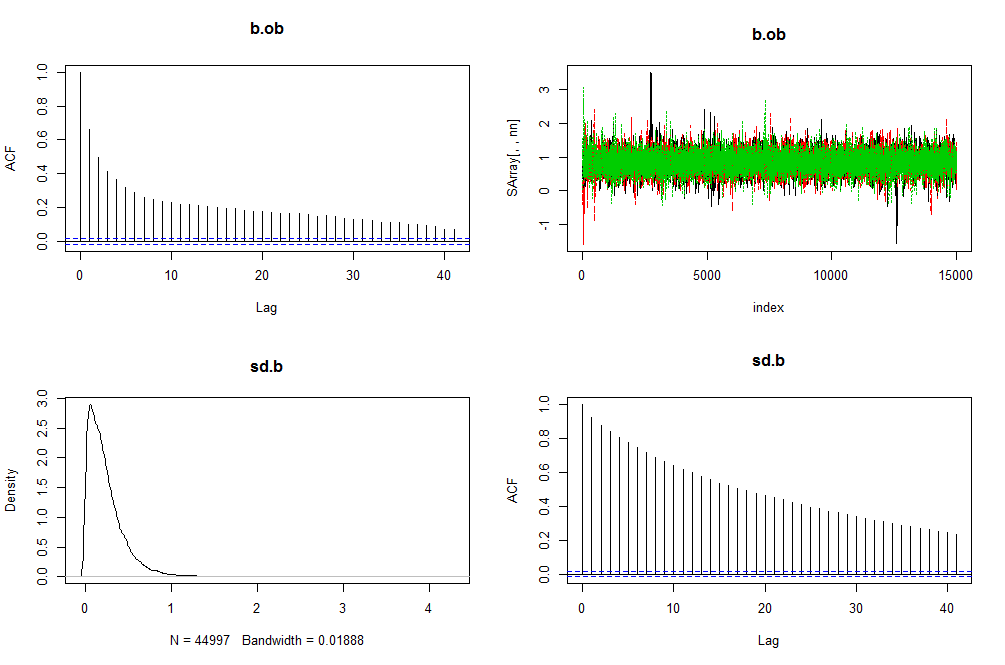
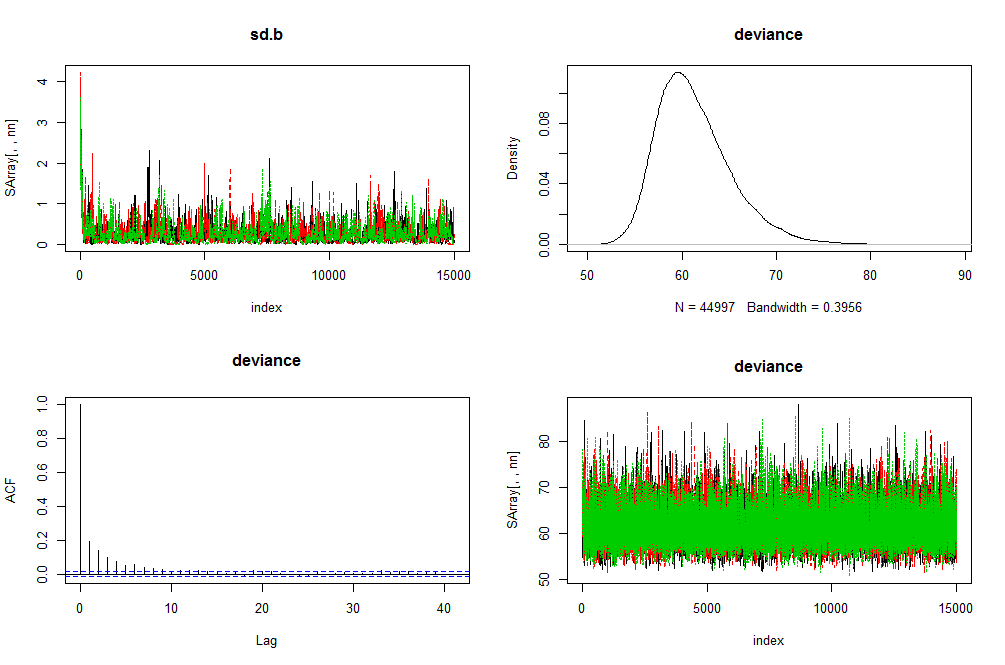
and Rhat is the potential scale reduction factor (at convergence, Rhat=1).

DIC info (using the rule, pD = Dbar-Dhat)

pD = 7.954 and DIC = 69.230

DIC is an estimate of expected predictive error (lower deviance is better).

**Trace and autocorrelation plots starting on page**

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.

**#1-Part b**

Burning first 4000 observations

*R-code*

*> SmokeHypeBaseM5=bugs(bugM3.dat, initM3.fun, paramsM4, model.file="SmokeHyperMod3.txt",*

*n.chains=3, n.iter=15000, n.burnin=4000,*

*n.thin=1 , debug=TRUE)*

*> print(SmokeHypeBaseM5,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 1.381 0.554 0.281 1.022 1.389 1.747 2.456 1.002 3700

b.ob 0.854 0.285 0.296 0.671 0.852 1.034 1.425 1.001 23000

sd.b 0.222 0.192 0.008 0.080 0.174 0.308 0.723 1.011 300

deviance 61.197 3.882 55.190 58.450 60.610 63.370 70.390 1.001 32000

For each parameter, n.eff is a crude measure of effective sample size,

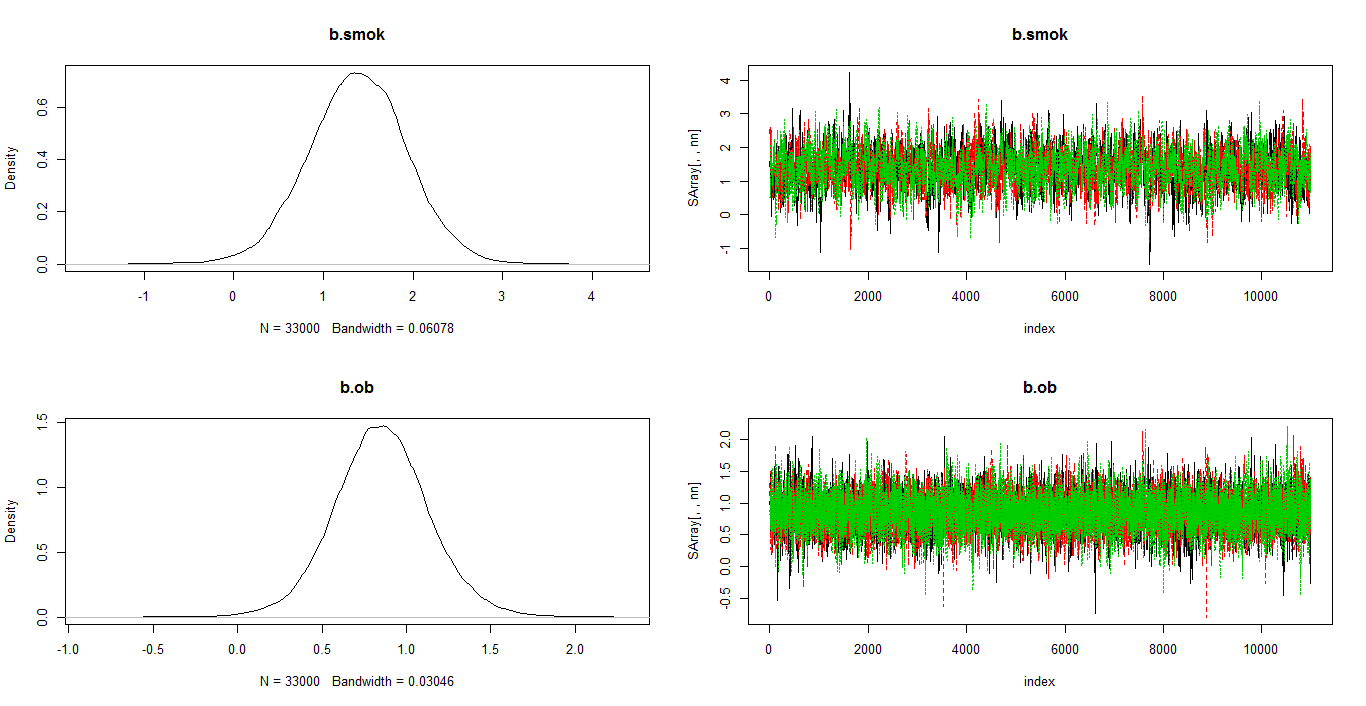
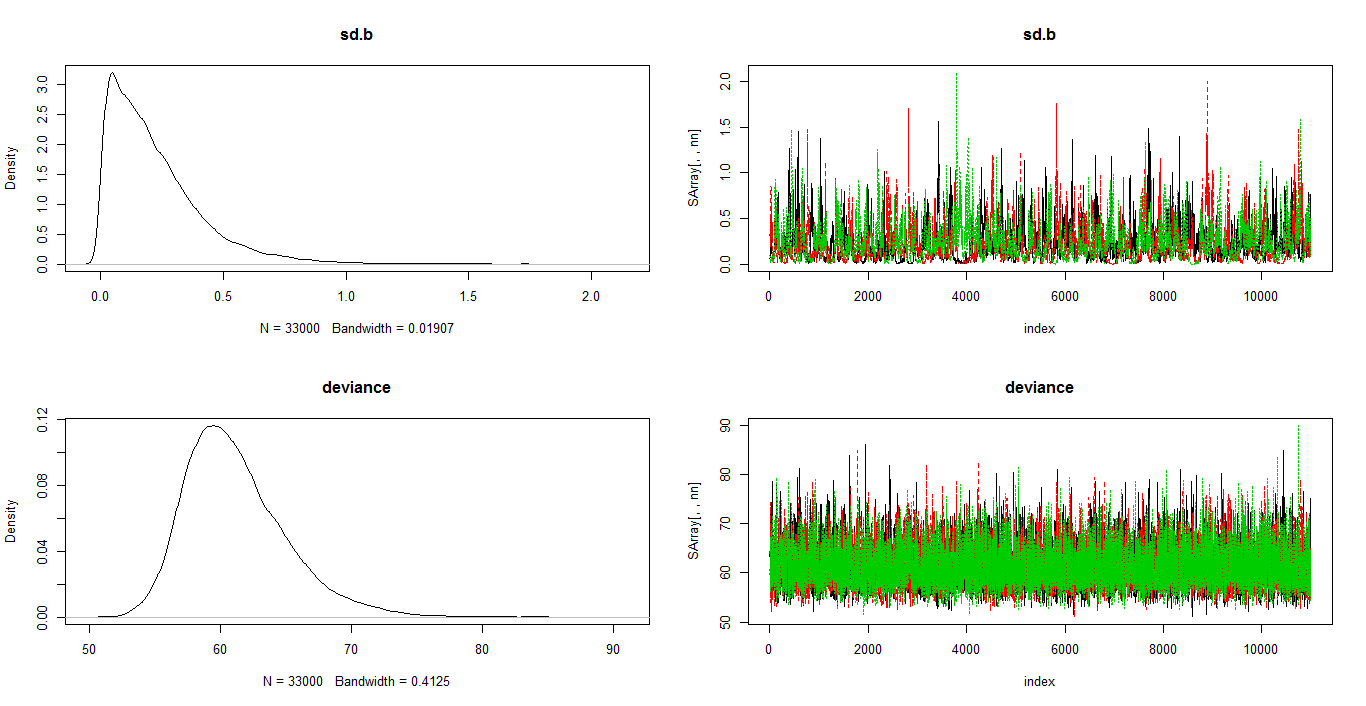
and Rhat is the potential scale reduction factor (at convergence, Rhat=1).

DIC info (using the rule, pD = Dbar-Dhat)

pD = 7.810 and DIC = 69.010

DIC is an estimate of expected predictive error (lower deviance is better).

Plotting the trace plots and density estimates

   
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.

**#1-Part c**

R-code

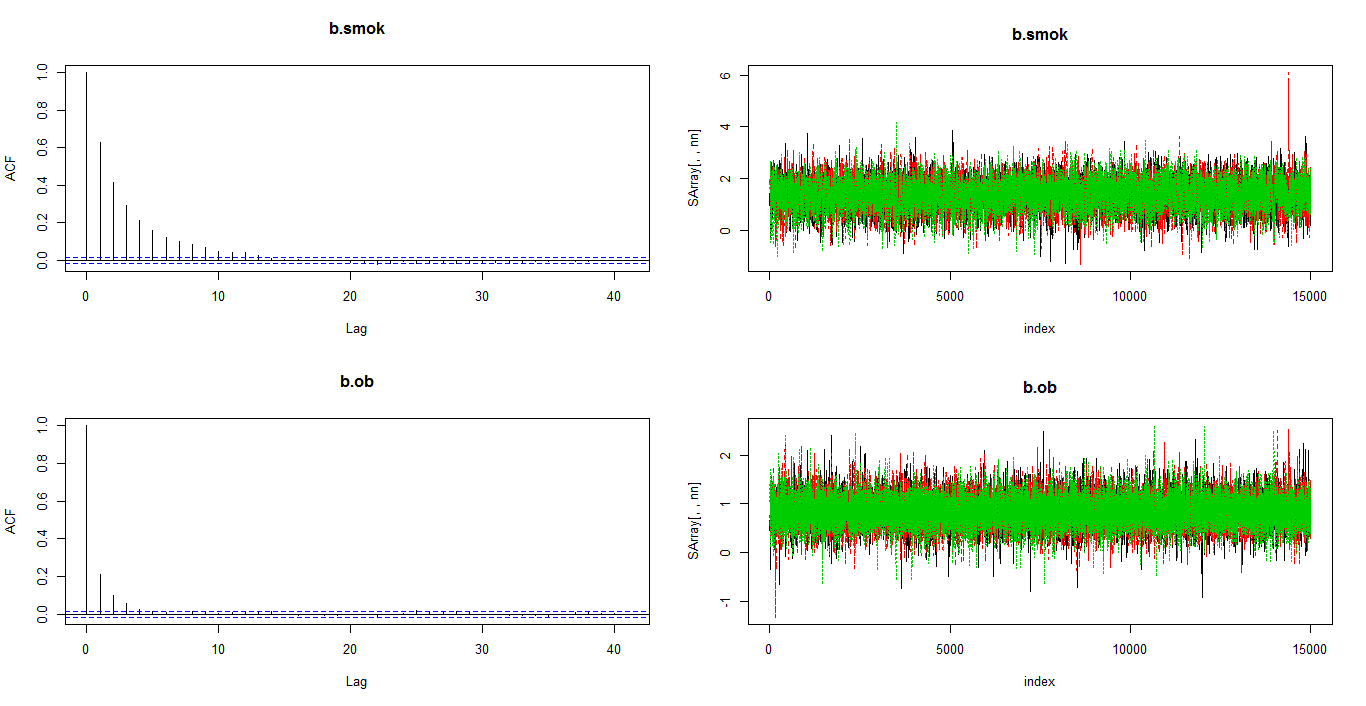
#thinning the chain by a factor of 4

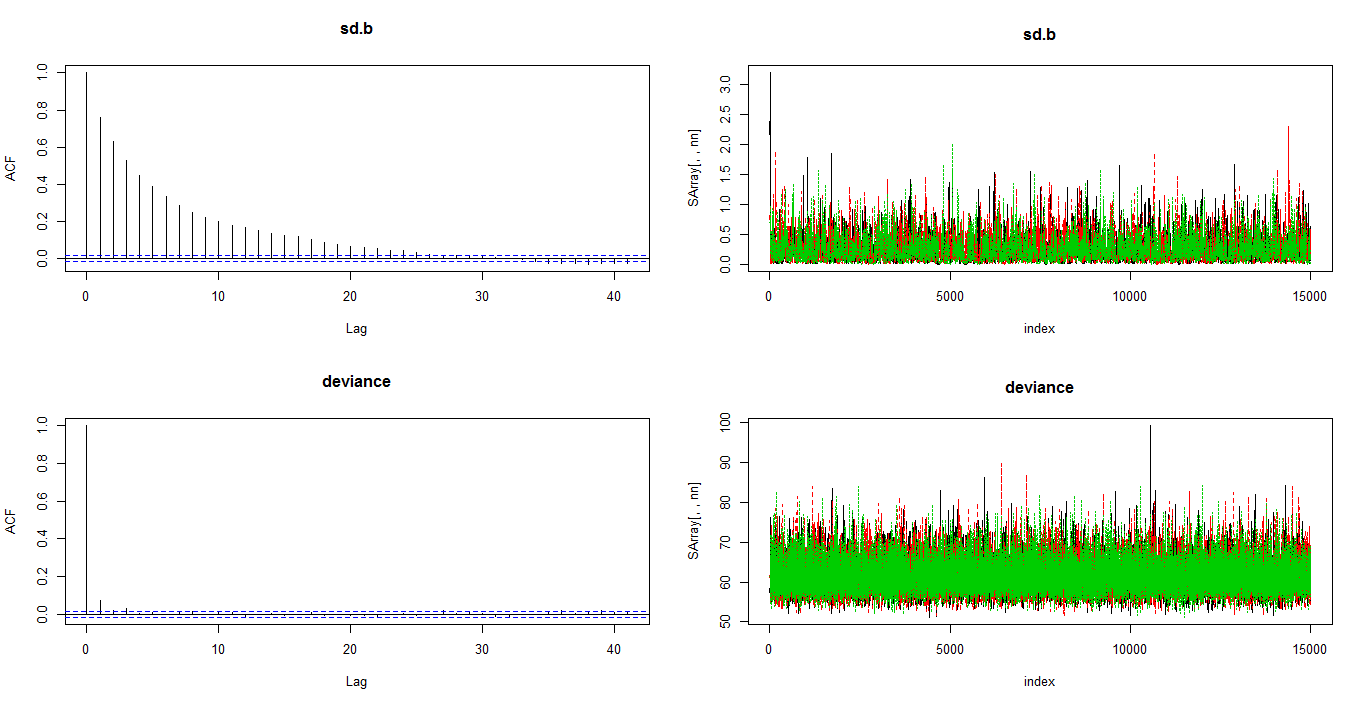
|  |
| --- |
| > SmokeHypeBaseM6=bugs(bugM3.dat, initM3.fun, paramsM4, model.file="SmokeHyperMod3.txt",  + n.chains=3, n.iter=15000, n.burnin=1,  + n.thin=4 , debug=TRUE  + ) |

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.

Trace and autocorrelation plots





**#1-Part d**

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

> 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(SmokeHypeBaseM6)

**CHAIN 1 - Mean, standard error and effective size**

|  |
| --- |
| > summary(wave0[1]); batchSE(wave0[1]); effectiveSize(wave0[1])# chain 1  Iterations = 1:14999  Thinning interval = 1  Number of chains = 1  Sample size per chain = 14999  1. Empirical mean and standard deviation for each variable,  plus standard error of the mean:  Mean SD Naive SE Time-series SE  b.smok 1.3916 0.5499 0.004490 0.010411  b.ob 0.8580 0.2968 0.002423 0.003268  sd.b 0.2238 0.1977 0.001615 0.005640  deviance 61.2977 3.9249 0.032048 0.036953  2. Quantiles for each variable:  2.5% 25% 50% 75% 97.5%  b.smok 0.28847 1.03400 1.4050 1.7540 2.4460  b.ob 0.28090 0.67110 0.8561 1.0380 1.4590  sd.b 0.01038 0.07739 0.1750 0.3117 0.7309  deviance 55.19000 58.53000 60.6900 63.5400 70.4205  b.smok b.ob sd.b deviance  0.010047037 0.002970501 0.005623678 0.037290477  b.smok b.ob sd.b deviance  2789.848 8247.975 1229.384 11281.306  **CHAIN 2 - Mean, standard error and effective size**  > summary(wave0[2]); batchSE(wave0[2]); effectiveSize(wave0[2])# chain 2  Iterations = 1:14999  Thinning interval = 1  Number of chains = 1  Sample size per chain = 14999  1. Empirical mean and standard deviation for each variable,  plus standard error of the mean:  Mean SD Naive SE Time-series SE  b.smok 1.3726 0.5604 0.004576 0.010593  b.ob 0.8547 0.2895 0.002364 0.003431  sd.b 0.2112 0.1997 0.001631 0.006351  deviance 61.2007 3.8452 0.031397 0.035091  2. Quantiles for each variable:  2.5% 25% 50% 75% 97.5%  b.smok 0.233875 1.01450 1.3820 1.7385 2.4450  b.ob 0.289285 0.67615 0.8545 1.0300 1.4310  sd.b 0.003823 0.07239 0.1602 0.2897 0.7198  deviance 55.270000 58.43000 60.6500 63.3300 70.2710  b.smok b.ob sd.b deviance  0.010248632 0.003158746 0.006627827 0.034051325  b.smok b.ob sd.b deviance  2799.3393 7120.6916 989.2614 12007.3373  **CHAIN 3 - Mean, standard error and effective size**  > summary(wave0[3]); batchSE(wave0[3]); effectiveSize(wave0[3]) # chain 3  Iterations = 1:14999  Thinning interval = 1  Number of chains = 1  Sample size per chain = 14999  1. Empirical mean and standard deviation for each variable,  plus standard error of the mean:  Mean SD Naive SE Time-series SE  b.smok 1.3840 0.5606 0.004577 0.010978  b.ob 0.8494 0.3038 0.002480 0.003851  sd.b 0.2269 0.2049 0.001673 0.007064  deviance 61.2877 3.9117 0.031940 0.036154  2. Quantiles for each variable:  2.5% 25% 50% 75% 97.5%  b.smok 0.246835 1.02800 1.3990 1.7545 2.4450  b.ob 0.266885 0.66410 0.8489 1.0340 1.4400  sd.b 0.006272 0.07896 0.1756 0.3184 0.7397  deviance 55.250000 58.52000 60.6800 63.4700 70.6005  b.smok b.ob sd.b deviance  0.010226270 0.003103558 0.005990399 0.039013715  b.smok b.ob sd.b deviance  2607.550 6222.560 841.174 11706.360  **ALL CHAINS - Mean, standard error and effective size**  > summary(wave0); batchSE(wave0);effectiveSize(wave0) # all chains together  Iterations = 1:14999  Thinning interval = 1  Number of chains = 3  Sample size per chain = 14999  1. Empirical mean and standard deviation for each variable,  plus standard error of the mean:  Mean SD Naive SE Time-series SE  b.smok 1.3827 0.5570 0.0026260 0.006156  b.ob 0.8541 0.2968 0.0013990 0.002035  sd.b 0.2206 0.2009 0.0009472 0.003682  deviance 61.2620 3.8943 0.0183584 0.020827  2. Quantiles for each variable:  2.5% 25% 50% 75% 97.5%  b.smok 0.25843 1.02500 1.3960 1.7490 2.4451  b.ob 0.27997 0.67060 0.8533 1.0340 1.4440  sd.b 0.00639 0.07625 0.1696 0.3066 0.7315  deviance 55.24000 58.50000 60.6700 63.4500 70.4300  b.smok b.ob sd.b deviance  0.005873206 0.001777361 0.003522138 0.021325378  b.smok b.ob sd.b deviance  8196.738 21591.227 3059.820 34995.004 |

**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.**

**#1-Part e**

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

R-code and output below

> gelman.diag(wave0[-3])

Potential scale reduction factors:

Point est. Upper C.I.

b.smok 1 1

b.ob 1 1

sd.b 1 1

deviance 1 1

Multivariate psrf

1

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. Output is shown below

> geweke.diag(wave0)

[[1]]

Fraction in 1st window = 0.1

Fraction in 2nd window = 0.5

b.smok b.ob sd.b deviance

0.8305 -1.6573 -0.8655 -0.2524

[[2]]

Fraction in 1st window = 0.1

Fraction in 2nd window = 0.5

b.smok b.ob sd.b deviance

1.88209 0.14357 -0.43422 0.02341

[[3]]

Fraction in 1st window = 0.1

Fraction in 2nd window = 0.5

b.smok b.ob sd.b deviance

-1.3311 -0.5170 -0.9872 0.1013

**#1-Part 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**

**#2-Part a**

Calculated deviance, DIC and Bayes factor measures given below.

Parts of the output showing the deviance, DIC and Bayes factor statistics output given here.

The whole code I used for this question is given at the end in the appendix, in case needed.

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

**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**

**MODEL 1**

> names(temp)=c("Deviance","2.5%","97.5%","mean","SD");temp **#MODEL 1 Deviance**

Deviance 2.5% 97.5% mean SD

46.204683 30.400000 62.880000 45.288958 8.305494

> names(temp)=c("ChiDev2","2.5%","97.5%","mean","SD");temp **# MODEL 1 CHIDEV2**

ChiDev2 2.5% 97.5% mean SD

16.963055 6.884000 28.825250 16.047275 5.662972

> rownames(temp)=c("SelfProgramed:","Openbugs Made:");temp

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

> c(xxx$mean$deviance,xxx$mean$dev) **# SELF PROGRAMMED and OPENBUGS MEANS for DEviance**

[1] 46.20517 46.20468

> DIC2<-Dbar+pd2; DIC2 **# DIC**

[1] 49.4639

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

[1] 50.11571

**MODEL 2**

> names(temp)=c("Deviance","2.5%","97.5%","mean","SD");temp **#DEVIANCE**

Deviance 2.5% 97.5% mean SD

24.109448 7.946900 41.525250 23.232516 8.572244

> names(temp)=c("ChiDev2","2.5%","97.5%","mean","SD");temp **#CHEDEV2**

ChiDev2 2.5% 97.5% mean SD

16.905456 6.950850 28.970000 16.028472 5.649806

> 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) **#SELF PROGRAMMED AND OPENBUGS DEVIANCE**

[1] 24.10992 24.10945

> DIC2<-Dbar+pd2; DIC2 **# DIC**

[1] 30.41021

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

[1] 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 n.eff

mod[1,1] 0.94136 0.23495 0.00000 1.000 1.000 1.000 1.00000 1.41373 17

mod[1,2] 0.00509 0.07116 0.00000 0.000 0.000 0.000 0.00000 1.29955 200

mod[2,1] 0.00351 0.05916 0.00000 0.000 0.000 0.000 0.00000 1.29684 280

mod[2,2] 0.05004 0.21802 0.00000 0.000 0.000 0.000 1.00000 1.39201 20

del[1] 0.94645 0.22513 0.00000 1.000 1.000 1.000 1.00000 1.40069 19

del[2] 0.94487 0.22823 0.00000 1.000 1.000 1.000 1.00000 1.40468 18

beta[1] 5.56211 1.93939 -0.37327 5.163 5.935 6.564 7.67200 1.24923 32

beta[2] -5.36073 1.89902 -7.46200 -6.345 -5.719 -4.943 0.43983 1.24044 34

deviance 24.25350 6.14401 19.36000 20.660 22.300 25.000 44.99000 1.21817 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.

**#2-Part b**

**#Only for MODEL 1**

**#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,] -1.41951555 -2.076000 2.056050 0.0062646463 1.038620

[2,] -1.45344210 -2.033000 2.033525 -0.0020013294 1.031708

[3,] 0.36756767 -2.038525 2.027000 -0.0070168085 1.030768

[4,] 0.59419123 -2.035000 2.046100 0.0092610334 1.032335

[5,] -0.33851842 -2.047000 2.046000 0.0076667835 1.027518

[6,] -0.07842876 -2.076000 2.055000 -0.0058168860 1.040222

[7,] 0.60475868 -2.082525 2.053050 -0.0029422121 1.038744

[8,] 1.46244013 -2.093525 2.059000 -0.0075481816 1.039961

[9,] 1.26118949 -2.074000 2.043000 -0.0048090592 1.037507

[10,] 0.78026164 -2.051525 2.035525 0.0008590868 1.030584

[11,] 0.87063322 -2.047000 2.058000 0.0007587071 1.031630

[12,] -0.08359047 -2.044525 2.031000 -0.0036833377 1.031083

[13,] 0.29840863 -2.062525 2.046525 0.0013316613 1.040289

[14,] -0.41916809 -2.074050 2.062050 -0.0063430047 1.037049

[15,] -0.80686984 -2.106525 2.033000 -0.0166460989 1.037925

[16,] -1.64635690 -2.059000 2.044000 -0.0058286962 1.033080

**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,] -1.45168092 -1.986000 1.952000 0.0055035023 1.0046636  [2,] -1.48643190 -1.961525 1.942000 -0.0014370468 0.9980588  [3,] 0.37596596 -1.983000 1.939000 -0.0070272732 0.9987307  [4,] 0.60769990 -1.953525 1.974525 0.0098946483 0.9985751  [5,] -0.34628132 -1.940050 1.965000 0.0093059947 0.9923961  [6,] -0.08032162 -2.005050 1.939000 -0.0057495479 1.0057053  [7,] 0.61836528 -1.990525 1.970050 -0.0040182117 1.0048078  [8,] 1.49551422 -1.980050 1.961000 -0.0063808651 1.0102707  [9,] 1.28963568 -1.970000 1.946000 -0.0042359123 1.0024274  [10,] 0.79771618 -1.958525 1.961525 0.0011302946 0.9989464  [11,] 0.89009846 -1.946000 1.967000 -0.0002209465 0.9955796  [12,] -0.08588642 -1.980000 1.944000 -0.0035628062 0.9997934  [13,] 0.30475705 -1.978525 1.963000 0.0006958645 1.0058822  [14,] -0.42919464 -1.983000 1.970525 -0.0053971334 1.0032588  [15,] -0.82576844 -2.032050 1.934000 -0.0176827528 1.0069579  [16,] -1.68440287 -1.977000 1.961000 -0.0055120920 0.9973165 |
| **PROBABILITY of extreme values**  > apply(xxx$sims.list$p.smaller,2,mean)  [1] 0.10326165 0.09258065 0.63867384 0.71333333 0.36870968 0.47121864 0.72397849  [8] 0.91788530 0.89028674 0.77763441 0.80197133 0.46677419 0.61347670 0.34659498  [15] 0.23025090 0.07064516 |
| |  | | --- | |  | |

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

**QUESTION 3**

**#3-Part 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] 0.49470406 0.03940186

**#3-Part b**

**Importance sampling using 1 uniform**

|  |
| --- |
| > 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] 0.4861274 0.1143912 |

**#3-Part c**

|  |  |
| --- | --- |
| **Acceptance-rejection method**  .  **R-code**   |  | | --- | | > 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]<- 888  + }  + }  > mean(accept == "Yes") **#Acceptance rate**  [1] 0.516  > c(mean(Y[Y<888]),var(Y[Y<888])) **#Mean and variance of y's that were accepted**  [1] 0.48247326 0.04577221  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 | |

**#3-Part 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"  + }  + }  > mean(x) **# Mean**  [1] 0.495785  > var(x) **# Variance**  [1] 0.04359647  > mean(accepted=="Yes",na.rm=TRUE) **# Acceptance rate**  [1] 0.6766767 |

**Appendix**

**Code for 2a**

#Question 2

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)

mean(x); sd(x)

mean(y); sd(y)

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

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)

sd(x^2)

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)

print(cropMod2.sim,dig=3)

#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

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

##### non calibrated ("pval-stats")

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

xxx$mean$chidev2.pval

xxx$mean$dev.pval

###########################

###########################

# 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

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

Dbar<-mean(xxx1);Dbar

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

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

pd1<-Dbar-Dhat;pd1

DIC1<-Dbar+pd1; DIC1

DIC2<-Dbar+pd2; DIC2

#bayes factor

xxx<-cropMod1.sim

xxx$mean$p.inv

# 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

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

#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

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

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

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

Dbar<-mean(xxx1);Dbar

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

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

pd1<-Dbar-Dhat;pd1

DIC1<-Dbar+pd1; DIC1

DIC2<-Dbar+pd2; DIC2

#bayes factor

xxx<-cropMod2.sim

xxx$mean$p.inv

# 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

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