Edpsy 590BAY: Multiple Regression in jags

C. J. Anderson

September 21, 2019

This document illustrates multiple regression in jag using 1 school from the nels data. Included here are the following:

Exploratory analysis: descriptive statistics histograms of math correlations and graphic of correlation all bi-variate of numeric variables tables of discrete create dummy coded variables

Model 1: All predictors: not mixing well and high auto-correlations extend.jags (better mixing) thinning (lower auto-correlations and better mixing) auto.run – out of curiousity

Model 2: Drop ses, gender and white Thin=5

Model 3: Put interactions (using centered predictor variables)

Model 4: math ~ centers homework + centered parent education Model evaluation: posterior predictive via monte carlo Model evaluation: posterior predictive output from rjags

Model 5: multiple t-distribution regression

Model evaluation: posterior predictive output from rjags

Bayes Factor: considers all possible models from model 1 (with centered)

### Step up

The packages that will be used

library(ellipse)  
library(ggplot2)  
library(coda)  
library(runjags)  
library(rjags)  
library(BayesFactor)  
  
set.seed(19843)

Set my working directory and read in and check the data

setwd("C:/Users/cja/Dropbox/edps 590BAY/Lectures/8 Multiple Linear Regression")  
nels <- read.table("nels\_school\_62821.txt",header=TRUE)  
head(nels)

## schid studid sex race homework schtyp ses paredu math clastruc size  
## 1 62821 0 1 4 4 4 1.63 6 62 3 3  
## 2 62821 1 2 4 5 4 0.48 4 68 3 3  
## 3 62821 3 1 4 5 4 0.69 5 56 3 3  
## 4 62821 4 2 4 5 4 0.60 4 68 3 3  
## 5 62821 6 2 4 2 4 0.77 4 61 3 3  
## 6 62821 7 2 4 2 4 1.19 5 61 3 3  
## urban geograph perminor rati0  
## 1 1 2 3 10  
## 2 1 2 3 10  
## 3 1 2 3 10  
## 4 1 2 3 10  
## 5 1 2 3 10  
## 6 1 2 3 10

We should always do some exploratary data analysis (e.g., basic descriptive statistics). I put them in a little table for future reference

# Descriptive statistics  
ybar<- mean(nels$math)  
std <- sd(nels$math)  
s2 <- var(nels$math)  
n <- length(nels$math)  
descriptive <- c(n,ybar,std,s2)  
names(descriptive) <- c("N","Ybar","std","s^2")  
descriptive

## N Ybar std s^2   
## 67.000000 62.820896 5.675373 32.209860

summary(nels$math)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 43.00 60.00 63.00 62.82 67.50 71.00

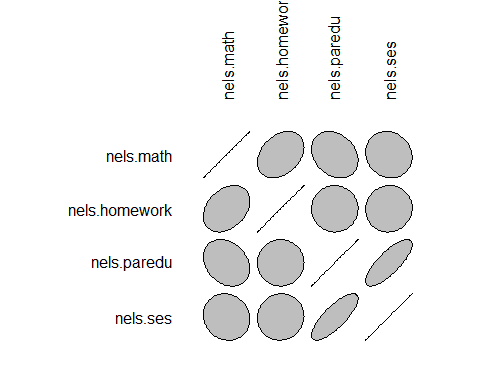
Since we’re adding more predictor variables we should also comput correlations. The following gets the correlation matrix

# Correlations between potential predictors ---> see www.cookbook-r.com  
xs <- data.frame(nels$math,nels$homework,nels$paredu,nels$ses)  
ctab <- cor(xs)  
round(ctab,2) # round correlations to 2 decimials for easier viewing

## nels.math nels.homework nels.paredu nels.ses  
## nels.math 1.00 0.33 -0.26 -0.10  
## nels.homework 0.33 1.00 0.00 0.04  
## nels.paredu -0.26 0.00 1.00 0.79  
## nels.ses -0.10 0.04 0.79 1.00

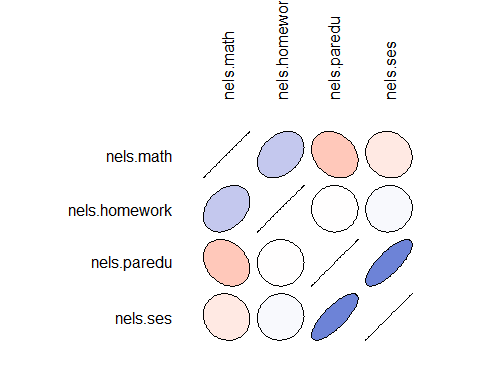
Or if you prefer a figure of a correlation matrix

# Figure of correlations --- grey scale  
plotcorr(ctab,mar=c(0.1,0.1,0.1,0.1))



I like color, so

# Figures of correlations in color   
colorfun <- colorRamp(c("#CC0000","white","#3366CC"), space="Lab")  
plotcorr(ctab, col=rgb(colorfun((ctab+1)/2), maxColorValue=255),  
 mar = c(0.1, 0.1, 0.1, 0.1))



The following a closer look at potential predictor variables, starting with how to treate “race”.

# What about race?   
r.tab <- table(nels$race)  
tot <- sum(r.tab)  
w <- r.tab[4]

Since 60 out of 67 are whites, I decided to recode this as a dichotomous variable

# Dichotomize race into white/not-white  
nels$white <- ifelse(nels$race==4, 1, 0)  
nels$white <- as.factor(nels$white)

We’ll treate gender as a catorical/factor variable

nels$gender <- as.factor(nels$sex)

### Ordinary Least Squares Regression

Even thoush we don’t expect all of these to be important, we will use all student level predictors: gender, ses, pareduc, homework and white

# All predictors  
nels$gender <- as.factor(nels$sex)  
summary(lm(math~gender + ses + paredu + homework + white, data=nels))

##   
## Call:  
## lm(formula = math ~ gender + ses + paredu + homework + white,   
## data = nels)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -13.8831 -2.4426 0.3711 3.4205 8.9577   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 70.3048 5.0930 13.804 < 2e-16 \*\*\*  
## gender2 1.5486 1.3083 1.184 0.24115   
## ses 3.6973 2.5384 1.457 0.15037   
## paredu -3.0370 1.2020 -2.527 0.01413 \*   
## homework 1.0629 0.3746 2.838 0.00616 \*\*   
## white1 -0.7322 2.2943 -0.319 0.75072   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 5.227 on 61 degrees of freedom  
## Multiple R-squared: 0.2161, Adjusted R-squared: 0.1518   
## F-statistic: 3.363 on 5 and 61 DF, p-value: 0.009523

Ones I think will be important

summary(lm(math~ paredu + homework, data=nels))

##   
## Call:  
## lm(formula = math ~ paredu + homework, data = nels)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -15.6051 -1.8546 0.5524 3.8143 9.2089   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 66.8935 3.6633 18.260 < 2e-16 \*\*\*  
## paredu -1.5636 0.6899 -2.266 0.02682 \*   
## homework 1.0930 0.3735 2.926 0.00475 \*\*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 5.23 on 64 degrees of freedom  
## Multiple R-squared: 0.1766, Adjusted R-squared: 0.1508   
## F-statistic: 6.862 on 2 and 64 DF, p-value: 0.001995

### Bayesian Model 1: all predictors

The data list with all the predictors

dataList <- list(  
 y =nels$math,  
 pared =nels$pared,  
 hmwk =nels$homework,  
 ses =nels$ses,  
 gender=nels$gender,  
 white = nels$white,  
 N=length(nels$math),  
 sdY = sd(nels$math)  
 )

The model with all predictors is basically the same as simple linear regression, but there are just more variables. (I’m not sure what all this is showing up green).

model1 = "model { for (i in 1:N){  
 y[i] ~ dnorm(mu[i] , precision)  
 mu[i] <- b0 + b1\*pared[i] + b2\*hmwk[i] + b3\*ses[i]+ b4\*gender[i]  
 + b5\*white[i]   
 }  
 b0 ~ dnorm(0 , 1/(100\*sdY^2) )   
 b1 ~ dnorm(0 , 1/(100\*sdY^2) )   
 b2 ~ dnorm(0 , 1/(100\*sdY^2) )   
 b3 ~ dnorm(0 , 1/(100\*sdY^2) )   
 b4 ~ dnorm(0 , 1/(100\*sdY^2) )   
 b5 ~ dnorm(0 , 1/(100\*sdY^2) )   
 sigma ~ dunif( 1E-3, 1E+30 )  
 precision <- 1/sigma^2   
 } "  
  
writeLines(model1, con="model1.txt")

Need some starting values. I used sample mean and standard deviation for one, but used random draws from normal or uniform for rest of them.

initsList = list(  
list("b0"=mean(nels$math), "b1"=0, "b2"=0, "b3"=0, "b4"=0, "b5"=0, "sigma"=sd(nels$math)),   
list("b0"=rnorm(1,50,5),"b1"=rnorm(1,-2,1),"b2"=rnorm(1,2,1),"b3"=rnorm(1,0,1), "b4"=rnorm(1,1,1), "b5"=rnorm(1,0,1),"sigma"=sd(nels$math)),   
list("b0"=rnorm(1,50,5), "b1"=rnorm(1,1,1),"b2"=rnorm(1,1,1), "b3"=rnorm(1,1,1), "b4"=rnorm(1,1,1), "b5"=rnorm(1,1,1), "sigma"=runif(1,0,10)),   
list("b0"=rnorm(1,50,6), "b1"=rnorm(1,0,1),"b2"=rnorm(1,0,1), "b3"=rnorm(1,0,1), "b4"=rnorm(1,0,1),"b5"=rnorm(1,0,1),"sigma"=runif(1,0,10))   
 )

Run simulatons of drawing samples from posterior.

mod1.runjags <- run.jags(model=model1,   
 monitor=c("b0","b1","b2","b3","b4","b5","sigma","dic"),  
 data=dataList,   
 n.chains=4,   
 inits=initsList)

## module dic loaded

## Compiling rjags model...  
## Calling the simulation using the rjags method...  
## Adapting the model for 1000 iterations...  
## Burning in the model for 4000 iterations...  
## Running the model for 10000 iterations...  
## Simulation complete  
## Calculating summary statistics...  
## Calculating the Gelman-Rubin statistic for 7 variables....  
## Finished running the simulation

No error messages but should check convergence. At the very miniumn

plot(mod1.runjags)

You can now look at the results, which have further diagnostic information.

print(mod1.runjags)

##   
## JAGS model summary statistics from 40000 samples (chains = 4; adapt+burnin = 5000):  
##   
## Lower95 Median Upper95 Mean SD Mode MCerr MC%ofSD  
## b0 55.388 68.483 83.207 68.817 7.1468 -- 0.52296 7.3  
## b1 -5.3471 -2.9615 -0.3236 -2.9635 1.2824 -- 0.080801 6.3  
## b2 0.29358 1.0744 1.7916 1.0736 0.38266 -- 0.0064109 1.7  
## b3 -1.652 3.6306 8.7055 3.624 2.6529 -- 0.11211 4.2  
## b4 -0.94956 1.6074 4.2249 1.6194 1.3239 -- 0.029458 2.2  
## b5 -5.4176 -0.56581 4.059 -0.6073 2.3995 -- 0.11405 4.8  
## sigma 4.4364 5.3022 6.36 5.3403 0.50073 -- 0.0046004 0.9  
##   
## SSeff AC.10 psrf  
## b0 187 0.90824 1.0201  
## b1 252 0.89121 1.012  
## b2 3563 0.1826 1.0008  
## b3 560 0.70558 1.0058  
## b4 2020 0.37673 1.0015  
## b5 443 0.80544 1.0076  
## sigma 11847 0.045405 1.0004  
##   
## Model fit assessment:  
## DIC = 420.3118  
## [PED not available from the stored object]  
## Estimated effective number of parameters: pD = 7.21805  
##   
## Total time taken: 5.9 seconds

# Adding Iterations

In this case, we probably are OK; however, if you want to get more samples and not re-run what you already have you can use the following. I didn’t run is here because if might take a long time.

mod1.extend <- extend.jags(mod1.runjags,   
 burnin=0,   
 sample=500000)  
plot(mod1.extend)  
print(mod1.extend)

# Thinning

When auto-correlations are large, thinning will often help. Only every 10th sample will be kept. This will increase computation time.

mod1.thin10 <- run.jags(model=model1,   
 monitor=c("b0","b1","b2","b3","b4","b5","sigma","dic"),  
 data=dataList,   
 n.chains=4,   
 inits=initsList,   
 thin=10)  
plot(mod1.thin10)  
gelman.plot(mod1.thin10)  
print(mod1.thin10)

# Auto-run

When you don’t want to think about what you’re doing and want to let the software make more decisions, you can use autorun mod1b.runjags <- autorun.jags(model=model1, {r eval=FALSE} monitor=c("b0","b1","b2","b3","b4","b5","sigma","dic"), data=dataList, n.chains=4, inits=initsList, thin=10) plot(mod1b.runjags)

### Model 2: Refinement

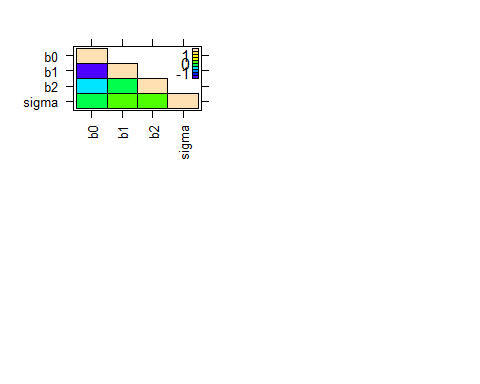
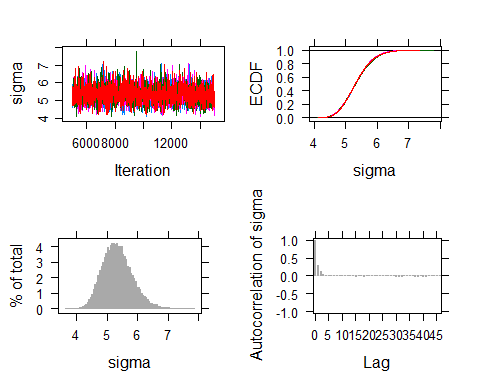
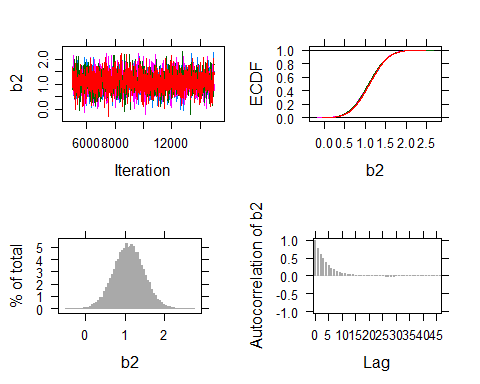
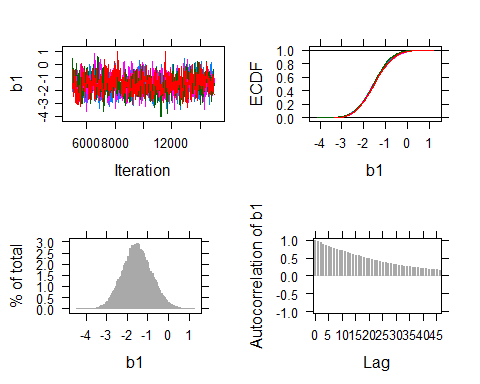
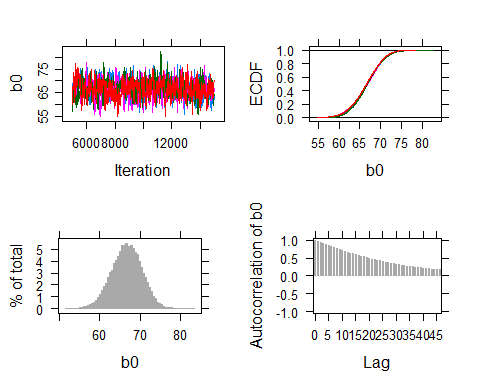
I will drop non-significant predcitors, which means change data list, model, starting values, and call to run jags.

dataList <- list(  
 y =nels$math,  
 pared =nels$paredu,  
 hmwk =nels$homework,  
 N=length(nels$math),  
 sdY = sd(nels$math)  
 )  
model2 = "model { for (i in 1:N){   
 y[i] ~ dnorm(mu[i] , precision)  
 mu[i] <- b0 + b1\*pared[i] + b2\*hmwk[i]   
 }  
 b0 ~ dnorm(0 , 1/(100\*sdY^2) )   
 b1 ~ dnorm(0 , 1/(100\*sdY^2) )   
 b2 ~ dnorm(0 , 1/(100\*sdY^2) )   
 sigma ~ dunif( 1E-3, 1E+30 )  
 precision <- 1/sigma^2   
 } "  
writeLines(model2, con="model2.txt")  
  
initsList = list(list("b0"=mean(nels$math), "b1"=0, "b2"=0, "sigma"=sd(nels$math)),   
 list("b0"=rnorm(1,50,5), "b1"=rnorm(1,-2,1), "b2"=rnorm(1,2,1), "sigma"=sd(nels$math)),   
 list("b0"=rnorm(1,50,5), "b1"=rnorm(1,1,1), "b2"=rnorm(1,1,1), "sigma"=runif(1,0,10)),   
 list("b0"=rnorm(1,50,6), "b1"=rnorm(1,0,1), "b2"=rnorm(1,0,1), "sigma"=runif(1,0,10))   
 )  
  
mod2.runjags <- run.jags(model=model2,   
 monitor=c("b0","b1","b2","sigma","dic"),  
 data=dataList,   
 n.chains=4,   
 inits=initsList)

## Compiling rjags model...  
## Calling the simulation using the rjags method...  
## Adapting the model for 1000 iterations...  
## Burning in the model for 4000 iterations...  
## Running the model for 10000 iterations...  
## Simulation complete  
## Calculating summary statistics...  
## Calculating the Gelman-Rubin statistic for 4 variables....  
## Finished running the simulation

plot(mod2.runjags)

## Generating plots...



print(mod2.runjags)

##   
## JAGS model summary statistics from 40000 samples (chains = 4; adapt+burnin = 5000):  
##   
## Lower95 Median Upper95 Mean SD Mode MCerr MC%ofSD  
## b0 59.516 66.755 74.019 66.688 3.7194 -- 0.14332 3.9  
## b1 -2.8823 -1.5372 -0.13196 -1.5244 0.70382 -- 0.026553 3.8  
## b2 0.3449 1.0953 1.8412 1.0955 0.38007 -- 0.0054803 1.4  
## sigma 4.4342 5.3045 6.3038 5.3384 0.48372 -- 0.0034558 0.7  
##   
## SSeff AC.10 psrf  
## b0 673 0.71718 1.0014  
## b1 703 0.70277 1.0013  
## b2 4810 0.08213 1.0003  
## sigma 19593 0.0033715 1.0001  
##   
## Model fit assessment:  
## DIC = 417.0809  
## [PED not available from the stored object]  
## Estimated effective number of parameters: pD = 4.14291  
##   
## Total time taken: 5 seconds

This model look pretty good.

### Model 3: Interactions

Since adding interactions, often leads to multicoliniarity, I centered the predictors bewteen adding the interaction.

nels$chomework <- nels$homework -mean(nels$homework)  
nels$cparedu <- nels$paredu -mean(nels$paredu )

Below is the rest of the code for this model.

dataList <- list(  
 y =nels$math,  
 cpared =nels$cparedu,  
 chmwk =nels$chomework,  
 N =length(nels$math),  
 sdY = sd(nels$math)  
 )  
model3 = "model { for (i in 1:N){  
 y[i] ~ dnorm(mu[i] , precision)  
 mu[i] <- b0 + b1\*cpared[i] + b2\*chmwk[i]   
 + b3\*cpared[i]\*chmwk[i]   
 }  
 b0 ~ dnorm(0 , 1/(100\*sdY^2) )   
 b1 ~ dnorm(0 , 1/(100\*sdY^2) )   
 b2 ~ dnorm(0 , 1/(100\*sdY^2) )   
 b3 ~ dnorm(0 , 1/(100\*sdY^2) )   
 sigma ~ dunif( 1E-3, 1E+30 )  
 precision <- 1/sigma^2   
 } "  
writeLines(model3, con="model3.txt")  
  
initsList = list(list("b0"=mean(nels$math), "b1"=0, "b2"=0, "b3"=0, "sigma"=sd(nels$math)),   
 list("b0"=rnorm(1,50,5), "b1"=rnorm(1,-2,1), "b2"=rnorm(1,2,1), "b3"=.3, "sigma"=sd(nels$math)),   
 list("b0"=rnorm(1,50,5), "b1"=rnorm(1,1,1), "b2"=rnorm(1,1,1), "b3"=.5, "sigma"=runif(1,0,10)),   
 list("b0"=rnorm(1,50,6), "b1"=rnorm(1,0,1), "b2"=rnorm(1,0,1), "b3"=-.3, "sigma"=runif(1,0,10))   
 )  
  
mod3.runjags <- run.jags(model=model3,  
 monitor=c("b0","b1","b2","b3","sigma","dic"),  
 data=dataList,   
 n.chains=4,   
 thin=5,   
 inits=initsList)

## Compiling rjags model...  
## Calling the simulation using the rjags method...  
## Adapting the model for 1000 iterations...  
## Burning in the model for 4000 iterations...  
## Running the model for 50000 iterations...  
## Simulation complete  
## Calculating summary statistics...  
## Calculating the Gelman-Rubin statistic for 5 variables....  
## Finished running the simulation

I assume that you next check convergence and then you can further look at the results. Since as your will see, the 95% interval for the interaction include 0, so we will delete it. This takes us back to model 2; however, in what follows I will go back to un-centered variables.

### Model 4: Final model with various derived statistics

I will add some values that I want computed and some derived varaibles. For some of the analyses done here, I find it more convient to use rjags than runjags, so I make the switch here

Since I’m switching back to UNCENTERED I

dataList <- list(  
 y =nels$math,  
 pared =nels$paredu,  
 hmwk =nels$homework,  
 N =length(nels$math),  
 sdY = sd(nels$math)  
 )  
  
nels\_final = "model { for (i in 1:N){  
 y[i] ~ dnorm(mu[i] , precision)  
 mu[i] <- b0 + b1\*pared[i] + b2\*hmwk[i]   
  
 res[i] <- y[i] - mu[i] # Data residuals  
 emp.new[i] ~ dnorm(mu[i],precision) # New Predicted y   
 res.new[i] <- emp.new[i] - mu[i] # New Predicted residuals  
 }  
 b0 ~ dnorm(0 , 1/(100\*sdY^2) )   
 b1 ~ dnorm(0 , 1/(100\*sdY^2) )   
 b2 ~ dnorm(0 , 1/(100\*sdY^2) )   
 sigma ~ dunif( 1E-3, 1E+30 )  
 precision <- 1/sigma^2   
# Derived quantities   
 fit <- sum(res[]) # sum data residuals this iterations  
 fit.new <- sum(res.new[]) # sum predicted residuals   
 fit.mean <- mean(emp.new[]) # mean of posterior predictions  
 } "  
writeLines(nels\_final, con="nels\_final.txt")

I added res[ ] as raw residuals, emp.new[ ] are predicted y, and res.new[ ] are the residuals of these new predictions. The predicted y will have the same values of cpared[ ] and chmwk[ ].

I used the same initial or starting values as above, so we don’t need to re-run this here. Which brings us to the simulation itself.

b0Init = mean(nels$math)  
b1Init = 0  
sigmaInit = sd(nels$math)  
initsList = list(b0=b0Init, b1=b1Init, sigma=sigmaInit )  
  
nels\_final <- jags.model(file="nels\_final.txt",   
 data=dataList,  
 inits=initsList,  
 n.chains=4,  
 n.adapt=1000)

## Compiling model graph  
## Resolving undeclared variables  
## Allocating nodes  
## Graph information:  
## Observed stochastic nodes: 67  
## Unobserved stochastic nodes: 71  
## Total graph size: 453  
##   
## Initializing model

update (nels\_final, n.iter=1000)   
  
# gets samples from all chains for 2000 iterations  
Samples <- coda.samples(nels\_final,   
 variable.names=c("b0","b1","sigma","fit","fit.new","fit.mean","emp.new"),   
 thin=5,  
 n.iter=10000)

When you get plots and print Samples, you will get something for every value in the variable names list. Note that this include 67 values of fit. This is one reason that I don’t include all the these “extras” when working on model refinement.

Do be able to use the samples, I need to do some data maninuplation. In “Samples”“, the data are in separate chaings. The object”Samples“” is an mcmc object which is difficult for me to manipulate, so I changed the object type.

ffit <- as.array(Samples) # change from mcmc to array object   
b <- rbind(ffit[,,1],ffit[,,2],ffit[, , 3], ffit[, , 4]) # combine the chains

The following takes Samples and breaks out specific estimates into different objects. To know what the “numbers” for the indices, I had to use what I know about the sample size of nels data and the number of other parameters (looking at the samples helps).

b0 <- b[,1]  
b1 <- b[,2]  
pred <- b[,3:69] # emp.new's  
fit <- b[,70]  
fit.mean <- b[,71]  
fit.new <- b[,72]  
sigma <- b[,73]

A little checking to see if things make sense. I expect the following two values to be very similar, because one is the mean of the data and the other is the mean of the postierior distribution of the fitted values.

mean(nels$math)

## [1] 62.8209

mean(pred)

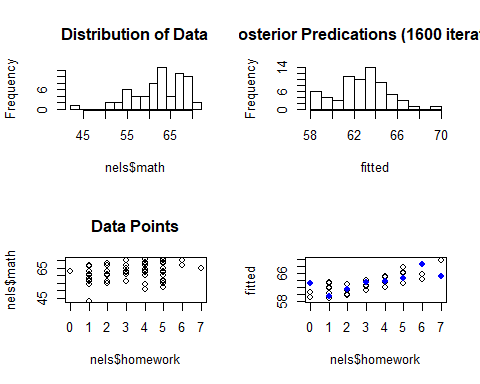
## [1] 62.79597

I wanted to look at the data and postierior distribution in different ways. In the first case, I collapsed over students so that I have one fitted value for each replications. This is like finding means for pred and nels$math, but doesn’t collapse as much.

fitted <- apply(pred,2,mean)

And some graphics

par(mfrow=c(2,2))  
hist(nels$math,breaks=10,main="Distribution of Data")  
hist(fitted,breaks=10,main="Posterior Predications (1600 iterations)")  
  
plot(nels$homework,nels$math,main="Data Points")  
plot(nels$homework,fitted, ylim=c(58,70))  
# Get the means to add to plot  
schMathHmwk <- aggregate(nels$math, list(nels$homework), mean)  
lines(schMathHmwk$Group.1,schMathHmwk$x,type='p',pch=19,col="blue")



I know collapse over replications (which makes more sense), and get a single mean value for each student.

# Compute statistics over rows (students)  
ymeans <- apply(pred,1,"mean")  
ymin <- apply(pred,1,"min")  
ymax <- apply(pred,1,"max")  
ysd <- apply(pred,1,"sd")

And the Bayesian p-values which gives another way to look at how well the model is doing for different aspects of the data.

pmean <- mean(mean(nels$math) < ymeans)  
pmin <- mean(min(nels$math) < ymin)  
pmax <- mean(max(nels$math) < ymax)  
psd <- mean(sd(nels$math) < ysd)

Understanding what these are is a bit easier by looking at them relative to data.

par(mfrow=c(2,2))  
   
hist(ymin,breaks=15,main=paste("Simulated within Jags: Minimums", "\nBayesian P-value =", round(pmin, 2)))  
lines(c(min(nels$math),min(nels$math)),c(0,2000),col="blue",lwd=2)  
2  
hist(ymax,breaks=15,main=paste("Simulated within Jags: Maximums", "\nBayesian P-value =", round(pmax, 2)))  
lines(c(max(nels$math),max(nels$math)),c(0,2000),col="blue",lwd=2)  
  
hist(ymeans,breaks=15,main=paste("Simulated within Jags: Means", "\nBayesian P-value =", round(pmean, 2)))  
lines(c(mean(nels$math),mean(nels$math)),c(0,2000),col="blue",lwd=2)  
  
hist(ysd,breaks=15,main=paste("Simulated within Jags: SDs", "\nBayesian P-value =", round(psd, 2)))  
lines(c(sd(nels$math),sd(nels$math)),c(0,2000),col="blue",lwd=2)

### Robust Regression: Student’s t-distribution

We swap out “dnorm” by “dt”, add priors for degrees of freedom (nu), and add nu to list of parametes to montor.  
Only the parts of the code that change are given belows.

The model,

nels\_t = "model { for (i in 1:N){  
 y[i] ~ dt(mu[i] , precision, nu)  
 mu[i] <- b0 + b1\*pared[i] + b2\*hmwk[i]   
 }  
 b0 ~ dnorm(0 , 1/(100\*sdY^2) )   
 b1 ~ dnorm(0 , 1/(100\*sdY^2) )   
 b2 ~ dnorm(0 , 1/(100\*sdY^2) )   
 sigma ~ dunif( 1E-3, 1E+30 )  
 precision <- 1/sigma^2   
 nuMinusOne ~ dexp(1/29)   
 nu <- nuMinusOne+1   
 } "

and the call for samples,

Samples <- coda.samples(nels\_t,   
 variable.names=c("b0","b1","sigma","nu"),   
 thin=5,  
 n.iter=10000)

### Bayes factor

If you feel compelled to compute these, it’s pretty if you use the BayesFactor package. This package does more then just linear regressions. See <http://bayesfactorpcl.r-forge.r-project.org/> for more information about the package.

You basically, ask for a regression using the regressionBF command with (in this case) all our possible variables. The following code gives Bayes factor for all possible models where the comparison model (default) is model with only an intercept.

library(BayesFactor)  
nels$xwhite <- as.numeric(nels$white)  
(bf <- regressionBF(math ~ paredu + homework + ses + xwhite + sex, data=nels))

## Bayes factor analysis  
## --------------  
## [1] paredu : 1.736485 ±0%  
## [2] homework : 6.983524 ±0%  
## [3] ses : 0.3338801 ±0%  
## [4] xwhite : 0.2545051 ±0%  
## [5] sex : 0.3031462 ±0%  
## [6] paredu + homework : 17.83269 ±0%  
## [7] paredu + ses : 1.247574 ±0%  
## [8] paredu + xwhite : 0.5765412 ±0%  
## [9] paredu + sex : 0.741968 ±0%  
## [10] homework + ses : 3.116336 ±0%  
## [11] homework + xwhite : 2.160394 ±0%  
## [12] homework + sex : 2.687732 ±0%  
## [13] ses + xwhite : 0.1284928 ±0.01%  
## [14] ses + sex : 0.1397701 ±0.01%  
## [15] xwhite + sex : 0.110552 ±0.01%  
## [16] paredu + homework + ses : 11.84489 ±0%  
## [17] paredu + homework + xwhite : 6.19577 ±0%  
## [18] paredu + homework + sex : 8.491123 ±0%  
## [19] paredu + ses + xwhite : 0.4952247 ±0%  
## [20] paredu + ses + sex : 0.7544532 ±0%  
## [21] paredu + xwhite + sex : 0.294487 ±0%  
## [22] homework + ses + xwhite : 1.276575 ±0%  
## [23] homework + ses + sex : 1.388366 ±0%  
## [24] homework + xwhite + sex : 1.006036 ±0%  
## [25] ses + xwhite + sex : 0.06318426 ±0.01%  
## [26] paredu + homework + ses + xwhite : 4.475333 ±0%  
## [27] paredu + homework + ses + sex : 7.718168 ±0%  
## [28] paredu + homework + xwhite + sex : 3.32971 ±0%  
## [29] paredu + ses + xwhite + sex : 0.3445106 ±0%  
## [30] homework + ses + xwhite + sex : 0.6339349 ±0%  
## [31] paredu + homework + ses + xwhite + sex : 3.246735 ±0%  
##   
## Against denominator:  
## Intercept only   
## ---  
## Bayes factor type: BFlinearModel, JZS

If you have lots and lots of possible models, you may want just the top ones (again compared against model with only an intercept), the add the line

#The best 5  
head(bf,n=5)

## Bayes factor analysis  
## --------------  
## [1] paredu + homework : 17.83269 ±0%  
## [2] paredu + homework + ses : 11.84489 ±0%  
## [3] paredu + homework + sex : 8.491123 ±0%  
## [4] paredu + homework + ses + sex : 7.718168 ±0%  
## [5] homework : 6.983524 ±0%  
##   
## Against denominator:  
## Intercept only   
## ---  
## Bayes factor type: BFlinearModel, JZS

or compare the best with the top competitors and put the best model in the denominator.

# Compare the best with top 5  
(top\_models <- head(bf)/max(bf) )

## Bayes factor analysis  
## --------------  
## [1] paredu + homework : 1 ±0%  
## [2] paredu + homework + ses : 0.6642233 ±0%  
## [3] paredu + homework + sex : 0.4761549 ±0%  
## [4] paredu + homework + ses + sex : 0.4328101 ±0%  
## [5] homework : 0.3916136 ±0%  
## [6] paredu + homework + xwhite : 0.3474389 ±0%  
##   
## Against denominator:  
## math ~ paredu + homework   
## ---  
## Bayes factor type: BFlinearModel, JZS