Stat 506 Final Exam

Leslie Gains-Germain

Contents

1 Computer Coding

1.0.1 R style guide

My R style guide is adopted from Google's R style guide as well as Hadley Wickham's and my own personal preferences. First, I address naming conventions because this is a topic I have vacillated on in the past. In the future, variable and function names will be all lowercase and I will use a period to separate words. Names will be short and descriptive.

```
# Good
area.one

# Bad
area_one
AreaOne
```

When indenting code, two spaces will be used. The exception is when a line break occurs. When this is the case, the wrapped line will be aligned with the first character inside the parentheses. I will indent any line of code that does not stand alone and depends on previous lines to run. Effort will be made to make lines short so that line breaks do not occur.

```
# Good
ggplot(int.area, aes(stat_area, Proportion)) +
  geom_boxplot(aes(colour=stat_area)) +
  geom_point() +
  theme(axis.title.x=element_blank(), legend.position="none",
        axis.text.x=element_text(size=15, angle=45))

# Bad
ggplot(int.area, aes(stat_area, Proportion)) +
    geom_boxplot(aes(colour=stat_area)) +
  geom_point()+
    theme(axis.title.x=element_blank(), legend.position="none",
        axis.text.x=element_text(size=15, angle=45))
```

When using ggplot, each layer will go on a new line.

```
# Good
ggplot(int.area, aes(stat_area, Proportion)) +
  geom_boxplot(aes(colour=stat_area))

# Bad
ggplot(int.area, aes(stat_area, Proportion))+geom_boxplot(aes(colour=stat_area))
```

A line of whitespace will be used to separate chunks of code, just like a paragraph break is used in ordinary english. If I think lines of code belong in the same "paragraph", there will be no whitespace between them. If I think a paragraph break is appropriate, I will include a line of whitespace. I will use code chunks in knitr like whitespace in R scripts so that common blocks of code go into the same code chunk.

```
# Good
linear.bay <- lm(proportion~year, data=counts)
curve.bay  <- lm(proportion~year+I(year^2), data=counts)

glm.bay <- glm(cbind(trips,no)~year, data=subset(ind.census, area=="resbay"),
               family='quasibinomial')

# Bad
linear.bay <- lm(proportion~year, data=counts)
curve.bay  <- lm(proportion~year+I(year^2), data=counts)
glm.bay    <- glm(cbind(trips,no)~year, data=subset(ind.census, area=="resbay"),
               family='quasibinomial')
```

I will use <-, not = for assignment.

```
# Good
compare <- read.csv("~/Documents/WritingProject/report/datafiles/compare.csv")

# Bad
compare = read.csv("~/Documents/WritingProject/report/datafiles/compare.csv")
```

Comments will begin with # and one space. Comments will be written before a code block if explaining the purpose of the block of code. If a comment explains the purpose of a specific line within a code block, the comment will be placed on that line, preceded by two spaces, #, and then one space. I will also use comments with a series of dashes to break the file into readable chunks.

```
# Good ---------------------------------------------------------------------
# Select rows of the compare dataset corresponding to area Cape Cleare
cleare.true <- compare %>% filter(area=="Cape Cleare")

ggplot(int.area, aes(stat_area, Proportion))+
  geom_boxplot(aes(colour=stat_area))+
  geom_point()+
  scale_colour_manual(values=getPalette(colourCount))+  # Specify color scheme
  ylab("Proportion of Use")+
  theme(axis.title.x=element_blank(), legend.position="none",
        axis.text.x=element_text(size=15, angle=45))

# Bad ----------------------------------------------------------------------
# read in data
compare <- read.csv("~/Documents/WritingProject/report/datafiles/compare.csv")

ggplot(int.area, aes(stat_area, Proportion))+
  geom_boxplot(aes(colour=stat_area))+
  geom_point()+
  scale_colour_manual(values=getPalette(colourCount))+#Specify color scheme
```

```
  ylab("Proportion of Use")+
  theme(axis.title.x=element_blank(), legend.position="none",
        axis.text.x=element_text(size=15, angle=45))
```

An opening curly brace should never go on it's own line, and a closing curly brace should always go on it's own line, unless it's followed by else. An opening curly brace should always be followed by a new line. For single statement blocks, I will always use curly braces.

```
# Good
for (i in 1:10) {
  x[i] <- 2*i
}

# Bad
for (i in 1:10) {x[i] <- 2*i  # opening brace should be followed by a new line
}
for (i in 1:10) x[i] <- 2*i  # use curly brackets even for single line statements
```

All binary operators should be surrounded by spaces.

```
# Good
x <- (3 + 4 + 5) / 2

# Bad
x<- (3 + 4 + 5) / 2  # No space after x
x <- (3 + 4 +5) / 2  # No space after the last plus sign
```

Adding extra spaces for alignment is permitted.

```
# Good
linear.bay <- lm(proportion~year, data=counts)
curve.bay  <- lm(proportion~year+I(year^2), data=counts)

# Not as good
linear.bay <- lm(proportion~year, data=counts)
curve.bay <- lm(proportion~year+I(year^2), data=counts)
```

There should be a space after commas but not before.

```
# Good
matrix(rep(0, 6), nrow = 2, ncol = 3)
compare[, 1:10]

# Bad
matrix(rep(0,6),nrow = 2,ncol = 3)
compare[ , 1:10]  # don't need a space before the comma
```

1.0.2 Homework Redos

I chose Stat 505 homework 11 to redo. I started thinking about my code for this homework assignment a few weeks ago when a 408 student came into the math learning center and was asking me questions about it. As I was helping her, I remembered how horrible and clunky my code was for this problem. When I found out that 'code cleaning' was our task for the final exam, I immediately thought of the Wald vs. Wilson coverage problem. I am most interested in improving the code that I used to loop through the Wald and Wilson coverage functions for different sample sizes, proportions, and confidence levels.

I first updated the coverage functions to fit the standards of my style guide. I show the coverage function for the single proportion Wald confidence interval here. You can see that I added spacing around operators (=,*). I also added spacing around the assignment signs to improve alignment of the lines within my function, and I indented the lines within the function. I made the same improvements to the `wilson.cover` function and the coverage functions for the difference in proportions. I don't show those functions here because it would take up too much space (they are shown in the R code appendix). I made a few improvements to the `cover` function itself. I added default values for the inputs `confidence` and `Nreps`. I also made the `wald.ci.l` and `wald.ci.u` lines shorter by adding a line for the margin of error (`me`). Lastly, I shortened the function by returning only the coverage proportion. Originally, I had saved and returned the individual confidence intervals, but these are not actually asked for in the assignment description.

```
# Old code
cover <- function (n,p,Nreps,confidence){
phat.vec<-rbinom(Nreps,n,p)/n
wald.ci.l<-phat.vec+qnorm((1-confidence)/2)*sqrt(phat.vec*(1-phat.vec)/n)
wald.ci.u<-phat.vec-qnorm((1-confidence)/2)*sqrt(phat.vec*(1-phat.vec)/n)
cis <- matrix(c(wald.ci.l,wald.ci.u),nrow=Nreps,ncol=2)
prop.cover <- sum(ifelse(cis[,1]<p & cis[,2]>p,1,0))/Nreps
finals <-list("waldcis"=cis,"prop.cover"=prop.cover)
return(finals)
}
cover(20,0.5,10,0.95)
```

```
# new code
# write a function to find coverage rates of wald cis
cover      <- function(p, n, confidence = 0.95, Nreps = 10000) {
  phat.vec   <- rbinom(Nreps, n , p) / n
  me         <- qnorm((1 - confidence) / 2) * sqrt(phat.vec*(1 - phat.vec) / n)
  wald.ci.l  <- phat.vec + me
  wald.ci.u  <- phat.vec - me
  prop.cover <- sum(ifelse(wald.ci.l < p & wald.ci.u > p, 1, 0)) / Nreps
  return(prop.cover)
}
cover(0.5, 20)
```

Then, I started working towards making my code more efficient. The old code used to find coverage rates for different p's and sample sizes was really clunky! In the new code, I wrote another function called `wald.loop` that would loop through values of p from 0.01 to 0.99. I then used sapply to run this function for several sample sizes. The old and new code is shown below, and the new code is much slicker!

```
# old code
Nreps <- 10000
waldcover.5 <- rep(0,99)
for(i in seq(0.01,0.99,by=0.01)){
waldcover.5[100*i]<-cover(5,i,Nreps,0.95)$prop.cover
}
waldcover.10 <- rep(0,99)
for(i in seq(0.01,0.99,by=0.01)){
waldcover.10[100*i]<-cover(10,i,Nreps,0.95)$prop.cover
}
waldcover.20 <- rep(0,99)
for(i in seq(0.01,0.99,by=0.01)){
waldcover.20[100*i]<-cover(20,i,Nreps,0.95)$prop.cover
}

waldcover.5.h <- rep(0,99)
```

```
for(i in seq(0.01,0.99,by=0.01)){
waldcover.5.h[100*i]<-cover(5,i,Nreps,0.99)$prop.cover
}
waldcover.10.h <- rep(0,99)
for(i in seq(0.01,0.99,by=0.01)){
waldcover.10.h[100*i]<-cover(10,i,Nreps,0.99)$prop.cover
}
waldcover.20.h <- rep(0,99)
for(i in seq(0.01,0.99,by=0.01)){
waldcover.20.h[100*i]<-cover(20,i,Nreps,0.99)$prop.cover
}
```

```
# new code
# Write a function to loop through lots of p's
wald.loop <- function(n, confidence) {
  wald.cover <- c(rep(0, 99))
  for (i in seq(0.01, 0.99, by = 0.01)) {
    wald.cover[zapsmall(100 * i)] <- cover(i, n, confidence)
  }
  return(wald.cover)
}

# evaluate the function at several sample sizes
waldc.95 <- sapply(list(5, 10, 20), wald.loop, confidence = 0.95)
waldc.99 <- sapply(list(5, 10, 20), wald.loop, confidence = 0.99)
```

Then, I streamlined the plotting process. I first organized the simulated coverage rates into one large dataframe, with columns for the sample sizes, proportions, and the coverage rates of the different intervals. Once the the simulated data was organized, plotting it was much easier. I used `ggplot` with the `facet_wrap()` option to plot coverage rates by proportion over different sample sizes. I made two graphs, for confidence levels of 0.95 and 0.99.

```
# old code
par(mfrow=c(3,3))
p <- seq(0.01,0.99,by=0.01)
plot(p,waldcover.5, ylim=c(0.7,1), type="l", lty=3, main="95% CIs, n=5")
abline(h=0.95)
lines(p,wilsoncover.5)

plot(p,waldcover.10, ylim=c(0.7,1), type="l", lty=3, main="95% CIs, n=10")
abline(h=0.95)
lines(p,wilsoncover.10)

plot(p,waldcover.20, ylim=c(0.7,1), type="l", lty=3, main="95% CIs, n=20")
abline(h=0.95)
lines(p,wilsoncover.20)

plot(p,waldcover.5.h, ylim=c(0.7,1), type="l", lty=3, main="99% CIs,n=5")
abline(h=0.99)
lines(p,wilsoncover.5.h)

plot(p,waldcover.10.h, ylim=c(0.7,1), type="l", lty=3, main="99% CIs,n=10")
abline(h=0.99)
lines(p,wilsoncover.10.h)

plot(p,waldcover.20.h, ylim=c(0.7,1), type="l", lty=3, main="99% CIs,n=20")
abline(h=0.99)
lines(p,wilsoncover.20.h)
```

```r
# new code
# organize coverage rates into a dataframe with sample sizes and p's
p        <- rep(seq(0.01, 0.99, by = 0.01), 3)  # vector of proportions
n        <- c(rep(5, 99), rep(10, 99), rep(20, 99))  # vector of sample sizes
frame <- cbind.data.frame(p, wald95   = as.vector(waldc.95),
                             wilson95 = as.vector(wc.95), n,
                             wald99   = as.vector(waldc.99),
                             wilson99 = as.vector(wc.99))

# make two plots for confidence levels of 0.95 and 0.99
ggplot(data = frame, aes(p, wald95)) +
  geom_hline(yintercept=0.95, linetype = "dotted") +
  geom_line(linetype = "dashed") +
  geom_line(aes(p, wilson95)) +
  facet_wrap(~n) +
  ggtitle("Coverage for 95% confidence intervals")

ggplot(data = frame, aes(p, wald99)) +
  geom_hline(yintercept=0.99, linetype = "dotted") +
  geom_line(linetype = "dashed") +
  geom_line(aes(p, wilson99)) +
  facet_wrap(~n) +
  ggtitle("Coverage for 99% confidence intervals")
```

I made similar improvements to the code used to produce the plot comparing coverage rates between Wald and Wilson intervals for the difference in proportions scenario. The improvements made were similar to what I have already described, so this code is shown in the appendix.

Then, I considered the basketball freethrow problem on the same homework assignment. My old code is shown below. First, I improved the code style. I indented the lines within the for loop and the readability of the code was greatly improved. In the original code I was inconsistent with naming conventions, so I changed `propMade` to `prop.made`. To decide if a shot was made within the loop, I revised my code to use `rbinom`. This way is cleaner because `runif` requires an `ifelse` statement. Lastly, I added meaningful comments to the code.

```r
# old code
n.shots <- rep(0,1000)
propMade <- rep(0,1000)
for(i in 1:1000){
keep.going <- 0
score <- 0
shots <- 0
miss <- 0
prev.miss <- 0
 while(keep.going<2){
   shots <- shots+1
   prev.miss <- miss
   miss <- ifelse(runif(1,0,1)>.6,1,0)
   score <- score+ifelse(miss==1,0,1)
   keep.going <- prev.miss+miss
   #keep.going <- ifelse(miss+prev.miss==2,FALSE,TRUE)
    ## set keep.going to FALSE when condition is met
 }
n.shots[i]<-shots
propMade[i] <- score/shots
#print(unlist( list( n.shots= shots, propMade = score / shots)))
}
```

```
# new code
n.shots <- rep(0, 1000) # empty vectors to store # shots and proportion made
prop.made <- rep(0, 1000)
for (i in 1:1000) {
  keep.going <- 0  # these variables reset to 0 every time we run thru the loop
  score <- 0
  shots <- 0
  miss <- 0
  prev.miss <- 0
  while (keep.going < 2) { # keep going until 2 shots are missed
    shots <- shots + 1  # keep track of number of shots
    prev.miss <- miss  # save outcome of the previous shot
    miss <- rbinom(1, 1, 0.6)  # 60\% chance of making the shot
    score <- score + ifelse(miss == 1, 0, 1)
    keep.going <- prev.miss + miss  # equals 2 if 2 shots missed in a row
  }
  n.shots[i] <- shots
  prop.made[i] <- score / shots
}
```

The next homework I chose to revisit is 506 homework 7. I chose this assignment because I feel like JAGs knowledge will be very useful in my future, but JAGs code can be very hard to write and JAGs output can be hard to manipulate. In all the JAGs homework assignments, I used the `jags()` function to run the models. I think the output from the `jags()` function is hard to work with, and I remember Wilson, Claire, and Kevin telling me that they use the `coda.samples()` function. They emphasized that `coda.samples` is nice because it allows you to run the burn-in period before the actual model, and discard the mcmc output from the burn in period. It also sounded like the structure of the mcmc output from the `coda.samples` function is easier to work with. I am using this final exam as an opportunity to learn how to use and work with output from `coda.samples`.

I first look at the question where we are asked to fit a random intercept and slope model to the Oxboys data with correlation a priori. I adapt the model to fit my style guidelines. This code wasn't nearly as messy as the code from the 505 assignment! I reordered a few lines, added some spaces, and aligned a few assignment signs. I also added comments and renamed the model to fit my naming convention. The old and new code is shown below.

```
# old code
#Random slope and intercept with correlation a priori

setwd("~/Documents/Stat506/Homework/HW7")

cat("
model {
  for(i in 1:n){
    y[i] ~ dnorm (y.hat[i], tau.y)
    y.hat[i] <- B[subject[i], 1] + B[subject[i],2] * x[i]
  }
  tau.y <- pow(sigma.y, -2)
  sigma.y ~ dunif (0, 100)
  for(j in 1:J){
    B[j, 1:2] ~ dmnorm(B.hat[j, ], Tau.B[,])
    B.hat[j,1] <- mu.a
    B.hat[j,2] <- mu.b
  }
  mu.a ~ dnorm (0, .0001)
  mu.b ~ dnorm (0, .0001)
  Tau.B[1:2,1:2] <- inverse(Sigma.B[,])
  Sigma.B[1,1] <- pow(sigma.a, 2)
  sigma.a ~ dunif (0, 100)
```

```
  Sigma.B[2,2] <- pow(sigma.b, 2)
  sigma.b ~ dunif (0, 100)
  Sigma.B[1,2] <- rho*sigma.a*sigma.b
  Sigma.B[2,1] <- Sigma.B[1,2]
  rho ~ dunif (-1, 1)}", file="CorrSlopInt.jags")
```

```
# new code
setwd("~/Documents/Stat506/Homework/HW7")

# Random slope and intercept with correlation a priori
cat("
model {
  for (i in 1:n) {
    y[i] ~ dnorm (y.hat[i], tau.y)
    y.hat[i] <- B[subject[i], 1] + B[subject[i],2] * x[i]
  }
  tau.y <- pow(sigma.y, -2)
  sigma.y ~ dunif (0, 100)
  for(j in 1:J){
    B[j, 1:2] ~ dmnorm(B.hat[j, ], Tau.B[,])  # multivariate normal distribution
    B.hat[j,1] <- mu.a
    B.hat[j,2] <- mu.b
  }
  mu.a ~ dnorm (0, .0001)
  mu.b ~ dnorm (0, .0001)
  Tau.B[1:2,1:2] <- inverse(Sigma.B[, ])  # precision matrix
  Sigma.B[1,1]   <- pow(sigma.a, 2) # first entry of variance covariance matrix
  sigma.a ~ dunif (0, 100)  # variance of random intercept
  sigma.b ~ dunif (0, 100)  # variance of random slope
  rho ~ dunif (-1, 1)
  Sigma.B[2, 2] <- pow(sigma.b, 2)
  Sigma.B[1, 2] <- rho * sigma.a * sigma.b
  Sigma.B[2, 1] <- Sigma.B[1, 2]}", file = "correlation.jags")
```

Next, I revised the code used to run the JAGs model. Instead of using the `jags()` function, I run a warmup of 1000 iterations, and then I use `coda.samples` to sample after the warmup.

```
# old code
##jags call
library(R2jags)
set.seed(52)

oxboys.data <-with(Oxboys, list(n=nrow(Oxboys), J=length(unique(Subject)),
                                subject=as.integer(Subject),
                                y= height, x=age))

oxboys.init2 <- function (){
  list (mu.a=rnorm(1), mu.b=rnorm(1), rho = runif(1, -1, 1),
        sigma.y=runif(1), sigma.a=runif(1) )
}
oxboys.param2 <- c ("B", "mu.a", "mu.b", "sigma.y", "sigma.a", "sigma.b", "rho")

withCorM2 <-  jags(data = oxboys.data, inits = oxboys.init2, oxboys.param2,
                   model.file = "CorrSlopInt.jags", n.chains=4, n.iter=5000)
```

```
# new code
library(R2jags)
require(coda)
set.seed(52)

#set up data to feed to the jags model
oxboys.data <- with(Oxboys, list(n       = nrow(Oxboys),
                                 J       = length(unique(Subject)),
                                 subject = as.integer(Subject),
                                 y       = height, x=age))
# specify starting values
oxboys.init2 <- function() {
  list(mu.a = rnorm(1), mu.b = rnorm(1), rho = runif(1, -1, 1),
       sigma.y = runif(1), sigma.a = runif(1))
}
# parameters to save
oxboys.param2 <- c ("B", "mu.a", "mu.b", "sigma.y", "sigma.a", "sigma.b", "rho")

# burn in period
warmup <- jags.model("correlation.jags", oxboys.data, inits = oxboys.init2(),
                     n.chains = 4, n.adapt = 1000)
# sampling after completed warmup
samples <- coda.samples(warmup, oxboys.param2, n.iter = 4000)
```

I then figure out how to make traceplots with the `coda.samples` output. Previously, I used `traceplot()`, but I didn't really like this function because it was hard to specify which random effects to plot. For example, I had trouble picking out the traceplot for the random intercept for boy 10. I looked into the mcmcplots package, and I found a function called `traplot()` that makes traceplots. This function is really nice, because I can directly call `B[10,1]`, the traceplot for the random intercept for boy 10.

```
# old code
traceplot(withCorM2, mfrow=c(1,3), varname=c("mu.a", "mu.b", "rho"))
#traceplot(withCorM2, mfrow=c(1,2), varname=c("B"))
traceplot(withCorM2, mfrow=c(1,4), varname=c("deviance","sigma.a", "sigma.b", "sigma.y"))
```

```
# new code
require(mcmcplots)
# Make traceplots for specified parameters
traplot(samples, parms = c("mu.a", "mu.b", "rho"))
traplot(samples, parms = c("sigma.a", "sigma.b", "sigma.y"))
traplot(samples, parms = c("B[10,1]", "B[10,2]", "B[26,1]", "B[26,2]"))
```

I also thought that pulling off summary measures from a `jags()` model was confusing. The `summary()` function isn't very useful when used on a model fit with `jags()`, but when used on a `coda.samples` model, it provides a nice summary of model output. I subset the output in the code below to show details for the parameters of interest.

```
# old code
require(xtable)
print(xtable(withCorM2$BUGSoutput$summary[c(1,10,27,36,53:59),c(1:3,7:9)]), table.placement = getOption("xtable.table.placement"
```

```
# new code
require(xtable)
# summarize coda.samples model output
print(xtable(summary(samples)$statistics[c(1, 10, 27, 36, 53:58), ]),
      table.placement = getOption("xtable.table.placement", "H"))
```

I also improved the code for the Oxboys inverse wishart model. The code improvements I made were really similar to what I show above, and this code can be found in the appendix. Next, I updated code for the owls model. I updated the old model code to fit my style guidelines. I mostly fixed the indentation.

```
# old code
cat("
model
{
for(i in 1:n){
  y[i] ~ dpois(lambda[i])
  log(lambda[i]) <- offset[i]+inprod(b.0[], X.0[i,])+a[nest[i]]+b[nest[i]]*x[i]
}
for(k in 1:K){
  b.0[k]~dnorm(0,0.0001)
}
for(j in 1:J) {
a[j] ~ dnorm(0, tau.a)
b[j] ~ dnorm(0, tau.b)
}
tau.a <- pow(sigma.a, -2)
sigma.a~dunif(0,1000)
tau.b <- pow(sigma.b, -2)
sigma.b~dunif(0,1000)
}", file="poisson4.jags")
```

```
# new code
cat("
model {
  for (i in 1:n) {
    y[i] ~ dpois(lambda[i])
    log(lambda[i]) <- offset[i]+inprod(b.0[], X.0[i,])+a[nest[i]]+b[nest[i]]*x[i]
  }
  for(k in 1:K){
    b.0[k]~dnorm(0,0.0001)  # priors on fixed effects parameters
  }
  for(j in 1:J) {
    a[j] ~ dnorm(0, tau.a)  # prior on random intercepts for nests
    b[j] ~ dnorm(0, tau.b)  # prior on random slopes for nests
  }
  tau.a <- pow(sigma.a, -2)
  tau.b <- pow(sigma.b, -2)
  sigma.a ~ dunif(0,1000)
  sigma.b ~ dunif(0, 1000)}", file="poisson4.jags")
```

Next, I use `coda.samples()` instead of `jags()` to run the model.

```
# old code
library(R2jags)
set.seed(52)

owls.data4 <-with(owls,
                  list(n=nrow(owls), J=length(unique(Nest)), K=14,
                       nest=as.integer(Nest), y=Ncalls, x=cTime,
                       offset=log(BroodSize),
                       X.0=model.matrix(~foodsat*sexmale+sexmale*bs(cTime,5))))

owls.param4 <- c("b.0","a","b", "sigma.a", "sigma.b")

owls.model4 <-  jags(data = owls.data4, parameters.to.save=owls.param4,
                     model.file = "poisson4.jags", n.chains=4, n.iter=5000)
```

```
# new code
library(R2jags)
set.seed(52)

# data to feed to jags model
owls.data4 <-with(owls,
                  list(n      = nrow(owls),
                       J      = length(unique(Nest)),
                       K      = 14,
                       nest   = as.integer(Nest),
                       y      = Ncalls,
                       x      = cTime,
                       offset = log(BroodSize),
                       X.0    = model.matrix(~foodsat * sexmale +
                                             sexmale * bs(cTime, 5))))
# parameters to save
owls.param4 <- c("b.0", "a", "b", "sigma.a", "sigma.b")

# burn in period
warmup.owls <- jags.model("poisson4.jags", owls.data4, n.chains = 4,
                          n.adapt = 1000)
# sampling after completed warmup
samples.owls <- coda.samples(warmup.owls, owls.param4, n.iter = 4000)
```

Lastly, I figure out how to show convergence criteria with `coda.samples` output using the `gelman.diag()` function.

```
# old code
require(mosaic)
xtable(favstats(owls.model4$BUGSoutput$summary[,8]))
xtable(favstats(owls.model4$BUGSoutput$summary[,9]))
```

```
# new code
require(mosaic)
xtable(favstats(gelman.diag(samples.owls)$psrf))  # summary of gelman diagnostics
```

```
# old code
require(xtable)
print(xtable(owls.model4$BUGSoutput$summary[c(69:71), 1:2]), table.placement = getOption("xtable.table.placement", "H"))
```

```
# new code
require(xtable)
# summary of model output - parameter estimates and SDs
print(xtable(summary(samples.owls)$statistics[68:70,]), table.placement =
      getOption("xtable.table.placement", "H"))
```

1.0.3 Reflection

I think one of my biggest coding challenges this year was learning how to learn code. I came to the realization this year that I will never *ever* know everything about coding. But, if I know where to turn when I'm unsure of how to code something, then I will always be able to figure out the code. I think this skill came slowly for me throughout the semester. Early on, I was using google a lot to answer coding questions, but I later learned that it was often much more efficient and straightforward to read helpfiles and/or package documentation. The other resource I started using was example code from Jim's notes and the book. The example code wasn't always exactly what I needed, but it often gave me ideas and a starting point. Then, I would turn to google and/or helpfiles to answer more specific questions. Once

I started to make use of all the resources available, coding came a lot easier.

In 411 and 412, we did some coding, but all of the code we used was given to us or available in very similar examples. I didn't really gain coding independence in 411 or 412 because of the way it was structured. I think I gained a lot of coding independence this year because I was forced to figure out how to code things on my own. I had to think about the task at hand (i.e. make a specific plot or write a function to find coverage rates of a confidence interval), and then I had to start from scratch and build my own code entirely. I think this experience taught me how to learn code on my own.

The second challenge I faced this year was debugging code. This may sound silly, but before this year I didn't really know how to read error messages. I didn't know that the red text that pops up can actually be helpful in figuring out what is wrong! Once I learned this, it became a lot easier to find the errors in my code. I still have trouble debugging JAGs code, but I think I will get a lot more practice with this in Bayes next semester.

I would advise a new student to work with others, especially in the beginning of the course. Alison, Claire, and I worked on 505 homeworks together, and I think this experience really helped me become more independent at coding. I remember in one of the early assignments we had to do some data cleaning, and I remember specifically discussing how to delete a row from the dataset. Now, this seems so silly and easy. At the time, however, it was so helpful to put the task into words. After putting it into words, it was easier to find the coding solution. Working with Alison and Claire also motivated me. I would notice that Claire had made a really cool plot, and it would make me want to figure out how to make an equivalent plot. I would advise a new student to spend time in the couch room, go to the help sessions, and work with others.

I would also advise a new student to not spend too much time beating your head against the wall. Looking back, I would often spend hours and hours trying to figure out some small section of code. I think it would have been much more productive to stop and ask for help. Coding can be *really* time consuming, and I think it's important to realize when it's time to take a break and get some help.

I have many goals for my future in stat computing. First, I'd like to start contributing more to github. I just learned how to push files to github in a seminar this semester, and I'd like to start using github as my main backup system and a place to share files with others. Second, I'd like to learn more about database management. I started using `dplyr` this year and it blows my mind how useful it is for data management. I'd like to learn mySQL as well because I think it's a useful tool and a marketable skill.

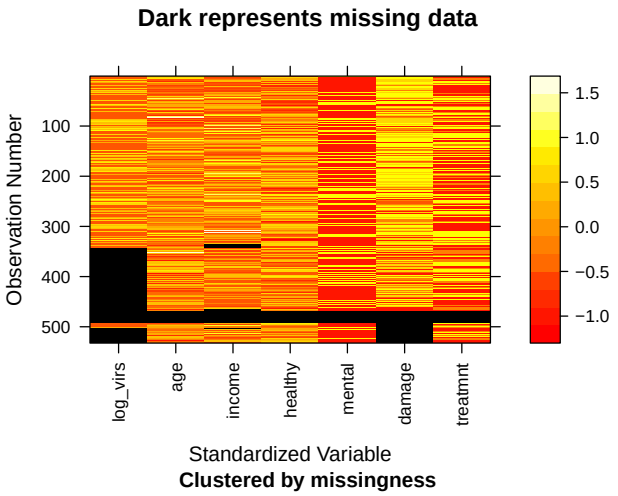
I'd also like to learn more about coding to make presentations and displays. I made my writing project presentation in slidify this spring. I'd like to explore slidify more, and I'd like to experiment with making markdown presentations.

2 Multiple Imputation

To use `mi`, we assume the data are Missing-At-Random (MAR), meaning that the probability of going missing depends on observed covariates. For each covariate, we assume that the probability of going missing follows a $Bernoulli(p_i)$ distribution, where p_i varies across covariates.

When I run `missing_data.frame` on the CHAIN dataset, it warns us that "some observations are missing on all included variables." It tells us that a more complicated model may be needed, because the missingness mechanism may not be MAR.

The image plot shows that the most of the missing data (across all covariates) occurs for observations $475 - 500$. This suggests that the missing rows of data did not occur at random. For some reason, there must have been a group of people with HIV who refused to participate in the interviews. For example, maybe all of the women with HIV living in the bronx refused to be interviewed, and the characteristics for each one of these people show up as missing data across all covariates. In this case, the Missing-At-Random assumption does not hold because the missing data probably depends on some other unmeasured covariate.



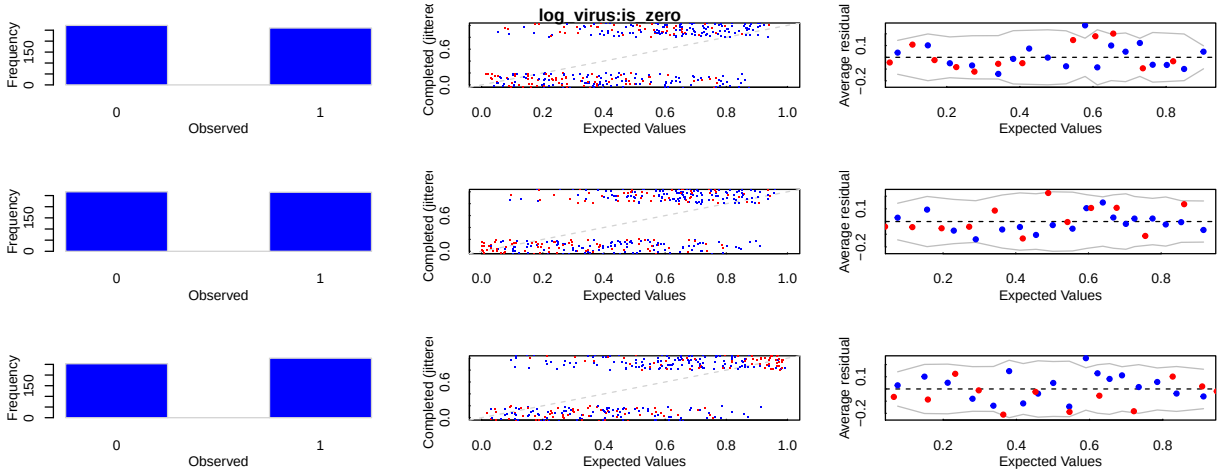
Next I look at the variable types. I changed `log_virus` from a continuous variable to nonnegative-continuous. I tried to change `age` and `healthy` to nonnegative-continuous variables, but I couldn't get it to work. I considered changing income to an ordered-categorical variable, but I decided to leave it as a continuous variable so that model interpretation would be easier. After I change `log_virus` to a nonnegative-continuous variable, the positive log virus values are separated from the zero log virus values in the missing data frame, and they are imputed separately. I think this is a reasonable thing to do because over half of the log virus values are zero!

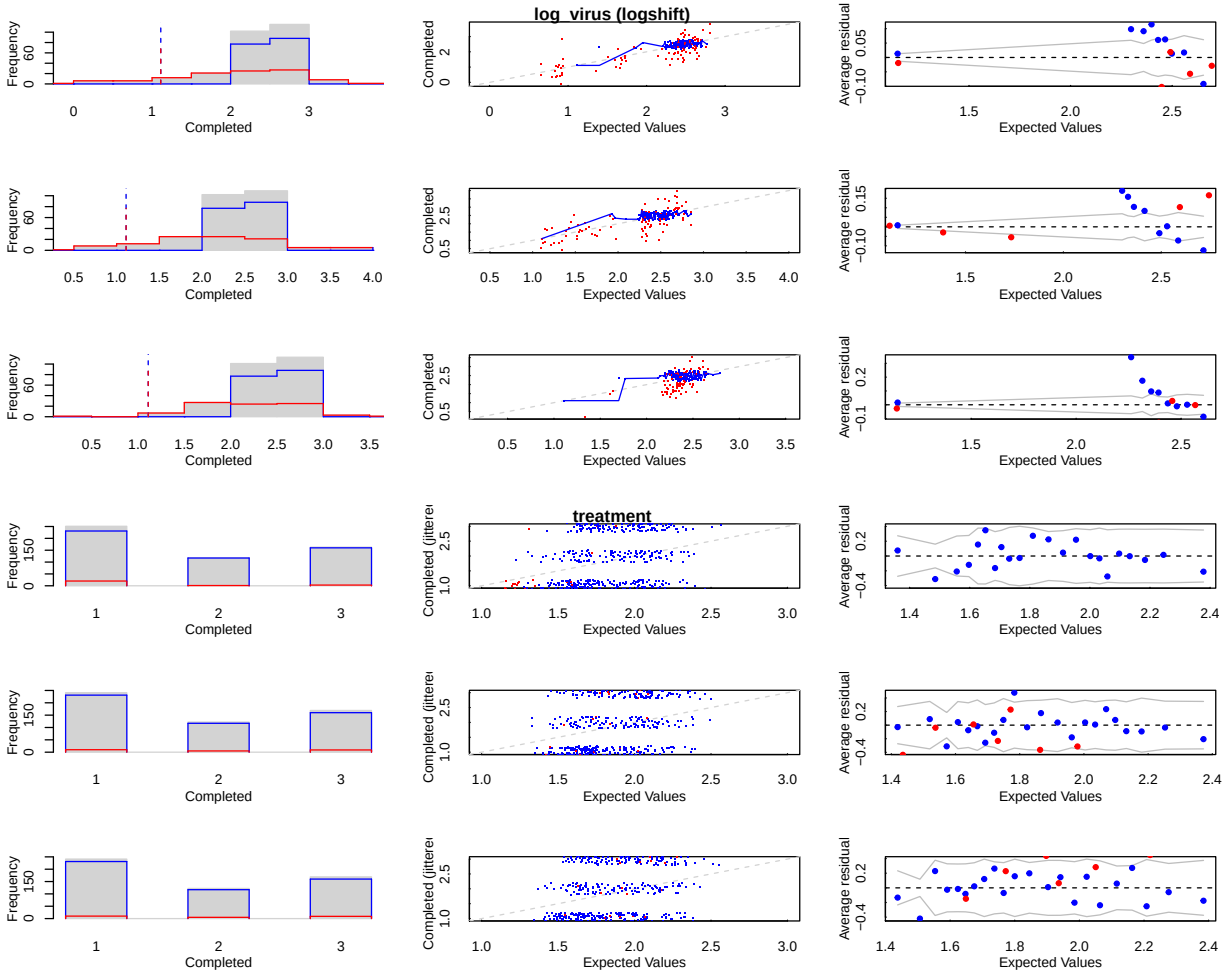
Next I ran the imputations to fill in missing values. I ran 4 chains with 30 iterations at first, but there were some large $\hat{R}$ values, indicating convergence was not reached. So I ran 150 more imputations and the $\hat{R}$'s looked better, although convergence didn't look great for the `healthy`, `age`, and `income` variables. The means still varied slightly across chains, and the $\hat{R}$ values for these three variables were greater than 1.14. I'll keep this in mind when I

interpret the results. I plotted the observed values, imputed values, and complete dataset for the variables `log_virus`, `log_virus:is zero`, and `treatment`.

	1	2	3	4
log_virus	1.73	1.73	1.72	1.72
age	0.01	-0.02	0.02	0.01
income	-0.08	0.01	-0.14	-0.07
healthy	-0.02	0.00	-0.01	-0.01
mental	1.28	1.28	1.29	1.26
damage	3.50	3.53	3.66	3.46
treatment	1.83	1.86	1.86	1.84
missing_log_virus	0.34	0.34	0.34	0.34
missing_age	0.04	0.04	0.04	0.04
missing_income	0.07	0.07	0.07	0.07
missing_healthy	0.04	0.04	0.04	0.04
missing_mental	0.04	0.04	0.04	0.04
missing_damage	0.12	0.12	0.12	0.12
missing_treatment	0.04	0.04	0.04	0.04

```
## mean_log_virus       mean_age    mean_income   mean_healthy
##          1.004          1.003          1.247          1.031
##    mean_mental    mean_damage mean_treatment   sd_log_virus
##          1.038          1.147          1.003          1.007
##         sd_age      sd_income     sd_healthy      sd_mental
##          1.001          1.227          1.006          1.037
##      sd_damage   sd_treatment
##          1.096          0.999
```





I then fit an additive model to `log_virus` with all other variables as predictors. To fit the model, we pool over five imputed datasets to estimate the model parameters. I don't think this model is appropriate, however, considering how many zero values there are for the `log_virus` variable. I think it would be more appropriate to model the zeros separately from the non-zero values. But, for exploration purposes, I'll fit the naive model and compare it to lm output. The `mi` output is shown below.

```
## bayesglm(formula = log_virus ~ age + income + healthy + mental +
##     damage + treatment, data = imputations, m = 5)
##             coef.est coef.se
## (Intercept) 10.97     1.98
## age         -0.11     0.03
## income      -0.01     0.24
## healthy     -0.05     0.02
## mental1      1.42     0.81
## damage.L    -3.11     1.01
## damage.Q     0.58     1.29
## damage.C    -0.91     0.86
## damage^4     0.08     0.62
## treatment.L -1.13     0.43
## treatment.Q  0.70     0.56
## n = 521, k = 11
## residual deviance = 12695.3, null deviance = 15377.2 (difference = 2682.0)
## overdispersion parameter = 24.4
## residual sd is sqrt(overdispersion) = 4.94
```

According to the `mi` output, an increase in age, income, and health is estimated to be associated with a decrease in the mean of log virus, after controlling for all other variables in the model. I am hesistant to trust the coefficients on age, income, and health, however, because the convergence criteria suggested disagreement among the four chains for these variables.

For nonadherent patients taking HAART, the mean of log virus levels is estimated to be 1.13 (SE= 0.43) lower than patients not currently taking HAART after accounting for all other variables in the model. The estimated effect is very different for adherent patients. For adherent patients taking HAART, the mean of log virus levels is estimated to be 0.70 (SE=0.56) higher than patients not currently taking HAART after accounting for all other variables. Another interesting variable is `damage`. Damage is an ordered variable from 1 to 5, depending on how much damage HIV has caused the immune system. The log virus level of patients with a damage level of 2 is estimated to be 3.11 (SE=1.01) lower than patients with a damage level of 1, after controlling for the other variables. I believe a damage level of 2 indicates less damage than a damage level of 1. The standard errors on the damage coefficients are relatively large, however, so the coefficient estimates aren't very trustworthy. Lastly, patients with poor mental health are estimated to have 1.42 (SE= 0.81) higher log virus levels than patients with good mental health, after controlling for the other variables.

I then compared the `mi` output to the `lm` output (shown below). The coefficient estimates do change, although the standard errors are similar. So, the actual estimate varies across methods, but the uncertainty around that estimate does not vary as much. The estimated age, income, and mental effects are pretty similar between `mi` and `lm`. The estimated income effect is greater in the lm output, and the standard error is smaller. In the lm output, the estimated damage effects are all negative. Each damage coefficient is more negative than the previous one, meaning that a decrease in damage caused to the immune system is estimated to be associated with a decrease in log viral loads, after accounting for the other variables. I assume here that a higher number for the damage variable means that they have less damage (and a higher CD4 count). I think this makes a lot more sense than the results for the mi model. The other major difference is the estimated coefficient for adherent patients taking HAART. In the lm model, both non-adherent *and* adherent patients taking HAART are estimated to have lower log virus levels than patients not taking HAART after controlling for other variables (opposite of what was predicted by the `mi` model for adherent patients). Again, I think this result makes more sense than the `mi` model.
I think I actually trust the lm output more in this case. First of all, I think the results of the lm output are a lot more reasonable. Second, we knew from the beginning that the assumptions of this multiple imputation procedure are not met because the missing data are not Missing-At-Random. So, in this case I think the imputed values are not helping us estimate the linear regression model. I think a better way to deal with the missing data in this case is to try to identify characteristics of those individuals who did not participate in the survey, and impute missing data based on those variables. If this is not possible, I think this is a good example of a situation where it's more appropriate to leave missing data out and recognize that the results could be biased.

The people in the study were not randomly selected from all people living with HIV in New York City, so inference does not extend beyond the patients in this study. Patients were

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	17.1105	1.8651	9.17	0.0000
age	-0.1061	0.0291	-3.65	0.0003
income	-0.3838	0.1128	-3.40	0.0007
healthy	-0.0324	0.0197	-1.64	0.1019
mental1	1.0590	0.5350	1.98	0.0486
damage2	-3.6705	1.0411	-3.53	0.0005
damage3	-3.9742	0.9459	-4.20	0.0000
damage4	-4.2219	0.8689	-4.86	0.0000
damage5	-5.9428	0.8438	-7.04	0.0000
treatment1	-2.0928	0.5942	-3.52	0.0005
treatment2	-2.1165	0.5327	-3.97	0.0001

Table 1: lm output

not randomly assigned ages, incomes, treatment, and the other characteristics, so we cannot infer that the observed variables cause changes in log viral loads.

The vignette and the pdf in CRAN were the most useful in getting to know this package. The vignette gave me a basic idea of how to use the package, and then the pdf document helped me figure out specific details and more information about certain functions. Google searches did not help in this case because there aren't many threads about this package on Stack Overflow.

3 R Code Appendix

3.1 Computer Coding

3.1.1 505 Homework Problem 11

```
# new code
# write a function to find coverage rates of wilson cis
cover.wilson <- function(p, n, confidence = 0.95, Nreps = 10000) {
  ptilde.vec   <- (rbinom(Nreps, n, p) + 2) / (n + 4)
  se           <- sqrt(ptilde.vec * (1 - ptilde.vec) / (n + 4))
  me           <- qnorm((1 - confidence) / 2) * se
  wald.ci.l    <- ptilde.vec + me
  wald.ci.u    <- ptilde.vec - me
  cis          <- matrix(c(wald.ci.l, wald.ci.u), nrow = Nreps, ncol = 2)
  prop.cover   <- sum(ifelse(cis[, 1] < p & cis[, 2] > p, 1, 0)) / Nreps
  finals       <- list("wilsoncis" = cis, "prop.cover" = prop.cover)
  return(finals)
}
cover.wilson(0.5, 20)
```

```
# new code
# Write a function to loop through lots of p's
wilson.loop <- function(n, confidence) {
  wilson.cover <- c(rep(0, 99))
  for(i in seq(0.01, 0.99, by = 0.01)) {
    wilson.cover[zapsmall(100 * i)] <- cover.wilson(i, n, confidence)$prop.cover
  }
  return(wilson.cover)
}
```

```r
# evaluate the function at several sample sizes
wc.95 <- sapply(list(5, 10, 20), wilson.loop, confidence = 0.95)
wc.99 <- sapply(list(5, 10, 20), wilson.loop, confidence = 0.99)
```

```r
# function to calculate coverage for a wald interval for diff in means
cover.diff <- function(p1, p2, confidence = 0.95, n1 = 20, n2 = 20,
                       Nreps = 10000) {
  phat1.vec  <- rbinom(Nreps, n1, p1) / n1
  phat2.vec  <- rbinom(Nreps, n2, p2) / n2
  se         <- sqrt(phat1.vec * (1 - phat1.vec) / n1+phat2.vec * (1 - phat2.vec)
                     / n2)
  me         <- qnorm((1-confidence)/2) * se
  wald.ci.l  <- phat1.vec - phat2.vec + me
  wald.ci.u  <- phat1.vec - phat2.vec - me
  cis        <- matrix(c(wald.ci.l, wald.ci.u), nrow = Nreps, ncol = 2)
  diff       <- p1-p2
  prop.cover <- sum(ifelse(cis[, 1] < diff & cis[, 2] > diff, 1, 0)) / Nreps
  finals     <- list("waldcis" = cis, "prop.cover" = prop.cover)
  return(finals)
}
```

```r
# function to calculate coverage for a wilson interval for diff in means
coverwilson.diff <- function(p1, p2, confidence = 0.95, n1 = 20, n2 = 20,
                             Nreps = 10000) {
  ptilde1.vec <- (rbinom(Nreps, n1, p1) + 1) / (n1 + 2)
  ptilde2.vec <- (rbinom(Nreps, n2, p2) + 1) / (n2 + 2)
  se          <- sqrt(ptilde1.vec * (1 - ptilde1.vec) / (n1 + 2) +
                      ptilde2.vec * (1 - ptilde2.vec) / (n2 + 2))
  me          <- qnorm((1 - confidence)/2) * se
  wald.ci.l   <- ptilde1.vec - ptilde2.vec + me
  wald.ci.u   <- ptilde1.vec - ptilde2.vec - me
  cis         <- matrix(c(wald.ci.l, wald.ci.u), nrow = Nreps, ncol = 2)
  diff        <- p1-p2
  prop.cover  <- sum(ifelse(cis[, 1] < diff & cis[, 2] > diff, 1, 0)) / Nreps
  finals      <-list("waldcis" = cis, "prop.cover" = prop.cover)
  return(finals)
}
```

```r
# Write a function to loop through lots of p1's
wald.diffloop <- function(p2, confidence) {
  wald.diffcover <- c(rep(0, 99))
  for (i in seq(0.01, 0.99, by = 0.01)) {
    wald.diffcover[zapsmall(100 * i)] <- cover.diff(i, p2, confidence)$prop.cover
  }
  return(wald.diffcover)
}

# evaluate the function at several p2's
walddiffc.95 <- sapply(list(0.1, 0.3, 0.5), wald.diffloop, 0.95)
```

```r
wilson.diffloop <- function(p2, confidence) {
  wilson.diffcover <- c(rep(0, 99))
  for (i in seq(0.01, 0.99, by = 0.01)) {
    wilson.diffcover[zapsmall(100 * i)] <-
      coverwilson.diff(i, p2, confidence)$prop.cover
  }
  return(wilson.diffcover)
}

# evaluate the function at several p2's
wcdiff.95 <- sapply(list(0.1, 0.3, 0.5), wilson.diffloop, confidence = 0.95)
```

```
# organize into dataframe
p1           <- rep(seq(0.01, 0.99, by = 0.01), 3)
p2           <- c(rep(0.1, 99), rep(0.3, 99), rep(0.5, 99))
level        <- c(rep(0.95, 297))
diffframe.95 <- cbind.data.frame(p1, wald = as.vector(walddiffc.95),
                                 wilson  = as.vector(wcdiff.95), p2, level)

ggplot(data = diffframe.95, aes(p1, wald)) +
  geom_hline(yintercept = 0.95, linetype = "dotted") +
  geom_line(linetype = "dashed") +
  geom_line(aes(p1, wilson)) +
  facet_wrap(~p2) +
  ggtitle("Coverage for 95% confidence intervals")
```

3.1.2 506 Homework Problem 7

```
cat("
model {
  for (i in 1:n) {
    y[i] ~ dnorm (y.hat[i], tau.y)
    y.hat[i] <- B[subject[i], 1] + B[subject[i], 2] * x[i]
  }
  tau.y <- pow(sigma.y, -2)
  sigma.y ~ dunif (0, 100)

  for (j in 1:J) {
    B[j, 1:2] ~ dmnorm(mu, Tau.B[,])
  }
  mu[1] ~ dnorm (0, .0001)
  mu[2] ~ dnorm (0, .0001)
  Tau.B[1:2, 1:2] ~ dwish(W[, ], 3)
  Sigma.B[1:2, 1:2] <- inverse(Tau.B[, ])
  sigma.a <- sqrt(Sigma.B[1, 1])
  sigma.b <- sqrt(Sigma.B[2, 2])
  rho     <- Sigma.B[1, 2]/sqrt(Sigma.B[1, 1] * Sigma.B[2, 2])}",
file="wishart.jags")
```

```
# data to feed to JAGs model
oxboys.data.wish <-with(Oxboys, list(n = nrow(Oxboys),
                                     J = length(unique(Subject)),
                                     subject = as.integer(Subject),
                                     y = height,
                                     x = age,
                                     W = diag(2)))
# parameters to save
oxboys.param3 <- c ("B", "mu", "sigma.y", "sigma.a", "sigma.b", "rho")

# burn in period
warmup.wish <- jags.model("wishart.jags", oxboys.data.wish, n.chains = 4,
                          n.adapt = 1000)
# sampling after completed warmup
samples.wish <- coda.samples(warmup.wish, oxboys.param3, n.iter = 4000)
```

```
require(mcmcplots)
# Make traceplots for specified parameters
traplot(samples.wish, parms = c("mu", "rho"))
traplot(samples.wish, parms = c("sigma.a", "sigma.b", "sigma.y"))
```

```
require(xtable)
# summarize coda.samples model output
print(xtable(summary(samples.wish)$statistics[c(1, 10, 27, 36, 53:58), ]),
      table.placement = getOption("xtable.table.placement", "H"))
```

3.2 Multiple Imputation

```
suppressMessages(library(mi))
data(nlsyV, package = "mi")
mdf <- missing_data.frame(nlsyV)
show(mdf)
mdf <- change(mdf, y = c("income", "momrace"), what = "type", to = c("non", "un"))
show(mdf)
summary(mdf)
image(mdf)
hist(mdf)
rm(nlsyV)
# good to remove large unnecessary objects to save RAM
options(mc.cores = 2)
imputations <- mi(mdf, n.iter = 30, n.chains = 4, max.minutes = 20)
show(imputations)
round(mipply(imputations, mean, to.matrix = TRUE), 3)
Rhats(imputations)
imputations <- mi(imputations, n.iter = 5)
plot(imputations)
plot(imputations, y = c("ppvtr.36", "momrace"))
hist(imputations)
image(imputations)
summary(imputations)
analysis <- pool(ppvtr.36 ~ first + b.marr + income + momage + momed + momrace,
data = imputations, m = 5)
display(analysis)
dfs <- complete(imputations, m = 2)
```

```
suppressMessages(library(mi))
data(CHAIN, package="mi")
cdf <- missing_data.frame(CHAIN)
#Warning message:
#In .local(.Object, ...) :
 #Some observations are missing on all included variables.
#Often, this indicates a more complicated model is needed for this missingness mechanism
```

```
image(cdf)
```

```
show(cdf)
head(CHAIN)
cdf <- change(cdf, y = c("log_virus"), what = "type", to = c("non"))
show(cdf)
summary(CHAIN$log_virus)
```

```
set.seed(119)
rm(CHAIN)
# good to remove large unnecessary objects to save RAM
options(mc.cores = 2)
imputations <- mi(cdf, n.iter = 30, n.chains = 4, max.minutes = 20)
imputations <- mi(imputations, n.iter = 150)
xtable(round(mipply(imputations, mean, to.matrix = TRUE), 3))
```

```
analysis <- pool(log_virus ~ age + income + healthy + mental + damage
                 + treatment, data = imputations, m = 5)
display(analysis)
dfs <- complete(imputations, m = 2)
```

```r
data(CHAIN, package="mi")
CHAIN$mental <- as.factor(CHAIN$mental)
CHAIN$damage <- as.factor(CHAIN$damage)
CHAIN$treatment <- as.factor(CHAIN$treatment)
lm.fit <- lm(log_virus ~ age + income + healthy + mental + damage +
             treatment, data = CHAIN)
require(xtable)
xtable(summary(lm.fit), caption="lm output")
```